Coverage Planning for Unmanned Aerial Vehicles

Kevin L. Yu

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Engineering

Pratap Tokekar, Co-chair
Ryan K. Williams, Co-chair
A. Lynn Abbott
Daniel J. Stilwell
Craig A. Woolsey

May 7, 2021
Blacksburg, Virginia

Coverage Planning for Unmanned Aerial Vehicles

Kevin L. Yu

(ABSTRACT)

This dissertation investigates how to plan paths for Unmanned Aerial Vehicles (UAV) for the task of covering an environment. Three increasingly complex coverage problems based on the environment that needs to be covered are studied. The dissertation starts with a 2D point coverage problem where the UAV needs to visit a set of sites on the ground plane by flying on a fixed altitude plane parallel to the ground. The UAV has limited battery capacity which may make it infeasible to visit all the points. A novel symbiotic UAV and Unmanned Ground Vehicle (UGV) system where the UGV acts as a mobile recharging station is proposed. A practical, efficient algorithm for solving this problem using Generalized Traveling Salesperson Problem (GTSP) solver is presented. Then the algorithm is extended to a coverage problem that covers 2D regions on the ground with a UAV that can operate in fixed-wing or multi-rotor mode. The algorithm is demonstrated through proof-of-concept experiments. Then this algorithm is applied to covering 2D regions, not all of which lie on the same plane. This is motivated by bridge inspection application, where the UAV is tasked with visually inspecting planar regions on the bridge. Finally, a general version of the problem where the UAV is allowed to fly in complete 3D space and the environment to be covered is in 3D as well is presented. An algorithm that clusters viewpoints on the surface of a 3D structure and has an UAV autonomously plan online paths to visit all viewpoints is presented. These online paths are re-planned in real time as the UAV obtains new information on the structure and strives to obtain an optimal 3D coverage path.

Coverage Planning for Unmanned Aerial Vehicles

Kevin L. Yu

(GENERAL AUDIENCE ABSTRACT)

This dissertation investigates how to plan paths for Unmanned Aerial Vehicles (UAV). Three increasingly complex coverage problems based on the environment that needs to be covered are studied. The dissertation starts with a 2D point coverage problem where the UAV needs to visit a set of sites on the ground by flying at a fixed altitude. The UAV has limited battery capacity which may make it impossible to visit all the points. A novel symbiotic UAV and Unmanned Ground Vehicle (UGV) system where the UGV acts as a mobile recharging station is proposed. A practical, efficient algorithm for solving this problem using Generalized Traveling Salesperson Problem (GTSP) solver is presented. Then the algorithm is extended to coverage of 2D regions on the ground with a hybrid UAV. The algorithm is demonstrated through proof-of-concept experiments. Then this algorithm is applied to covering 2D regions on 3D structures. This is motivated by bridge inspection application, where the UAV is tasked with visually inspecting regions on the bridge. Finally, a general version of the problem where the UAV is allowed to fly in 3D space and the environment to be covered is in 3D as well is presented. An algorithm that clusters points on the surface of a 3D structure and has an UAV autonomously plan online paths to visit all viewpoints is presented. These online paths are re-planned in real time as the UAV obtains new information on the structure and strives to obtain an optimal 3D coverage path.

Dedication

To Bob Francis.

Acknowledgments

I would like to thank Virginia Polytechnic Institute and State University for providing me the opportunity to study here and providing a space that helped me excel in my research, make timeless memories and finish my graduate career.

I am grateful for my advisor, Pratap Tokekar. He has guided me throughout my Ph.D. career by helping me tackle complicated research problems, inspired me to dive into new areas and provided me with a lab where I was surrounded by peers who pushed me to do better.

I would like to acknowledge my committee Ryan K. Williams, A. Lynn Abott, Daniel J. Stilwell, and Craig A. Woolsey. My committee members helped me achieve my goals and pushed me to think outside of the box when conducting my final research.

I like to acknowledge my previous mentor Bob Francis and old colleague Ron Crammer. Bob Francis made a great impact on my in choosing to pursue a graduate career and mentored me through my internships. Ron Crammer taught me to be always caring, never grow old, and continues to be an inspiration.

My friends and faculty who have helped me throughout graduate school. Without my friends, old and new, I would not have made it through graduate school. They have provided me a great outlet from research and we made many unforgettable memories. Thank you to all the faculty that have helped me, provided guidance and continue to help me into my future careers.

I am forever thankful for my girlfriend Wei Wang. She has supported me since the day we meet and continues to do so. She continues to make me laugh and smile. I always feel supported by her and she has made much of my graduate career possible through her continued support and love.

Finally my family, Bo Herman, Tod Herman and Lucia Yu have made everything possible. Without my parents I would have never made it this far. They have pushed me to always do my best and have supported me throughout my life. They continue to inspire me, even after earning my degree and I continue to feel their love. I could not be more proud of my sister Lucia Yu. She has always achieved greatness and continues to do so. It is great to see her starting her career as a dentist and is also graduating with me at the same time, making it even more special. I will always aspire to be like her.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

UAVs are starting to be used in a plethora of applications such as package delivery [110], surveillance [76], environmental monitoring [38, 70, 87], precision agriculture [30, 111], search and rescue [100, 107], and infrastructure inspection [83]. UAVs can benefit such applications by making the operations safer, faster, and more cost-efficient [59, 78]. Current methods for integrating UAVs into these applications require skilled pilots to manually fly the UAV. However to fully exploit the advantages we need autonomous UAV operations [29, 82]. Specifically, high-level planners and coordination algorithms are needed for UAVs working with other autonomous vehicles or even humans. In this dissertation, planning and coordination algorithms are developed with a particular focus on coverage tasks that occur in the previously mentioned applications.

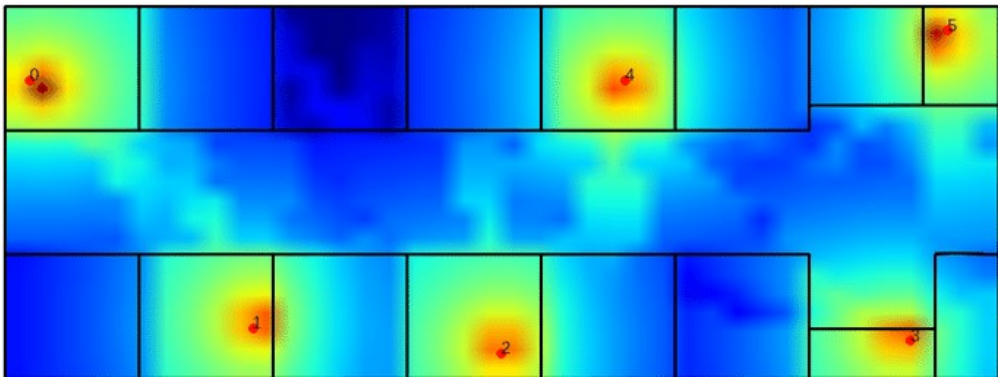


Figure 1.1: 2D point coverage: In surveillance applications, UAVs may be tasked with visiting a set of points (e.g., hotspots) so that they can monitor the environment [39].

Figure 1.2: 2D area coverage: In environmental monitoring and precision agriculture applications, the UAV may be tasked with covering areas on the ground to gather scientific data [117].

Consider the following applications where autonomous planning with UAVs can be beneficial for area coverage. In environmental monitoring applications, UAVs can be equipped with a variety of sensors such as thermal camera, RGB camera, and air pollutant sensors to monitor the temperature, wild life, and pollutants in the air [68]. UAVs can gather data by flying over the regions of interest. We may be able to cover larger areas which can be critical in applications such as precision agriculture and search-and-rescue, as shown in Figures 1.1 and 1.2. Unlike manual data collection, UAVs can operate persistently and collect data at unprecedented spatiotemporal scales [33, 114].

Depending on the application, the sites to be covered by the UAV may be specific points of interest on the ground or 2D regions on the ground. Figure 1.1 shows an example of surveillance where the UAV is tasked with monitoring a specific set of points of interest on the ground plane [76, 110, 122]. Figure 1.2 shows an example of precision agriculture where a UAV can be used to monitor a set of plots in a farm. In these applications, the UAV is covering points or areas on the ground plane. However, there are other applications

Figure 1.3: UAV conducting 3D visual inspection of a bridge.

where UAVs may be required to conduct 3D coverage. For example, consider the application of UAVs for inspecting infrastructure, e.g., bridges, for defects such as cracks, spawls, and rust [5]. UAVs equipped with sensor suites containing thermal cameras and RGB cameras can help inspect the bridges for such defects. This requires 3D coverage from multiple angles and not only an overhead view. 3D coverage planning requires taking into account the visibility constraints due to the environment itself.

Motivated by these use cases, in this dissertation three types of coverage problems are studied: 2D point coverage, 2D area coverage, and 3D visual coverage. While similar problems have been studied in the past [68, 83, 110], this dissertation focuses on addressing specific challenges as described in the next section.

1.1 Challenges

In the following, the challenges associated with deploying UAVs for coverage are highlighted. Particularly this dissertation is motivated by applications such as environmental monitoring that may require persistent operations and infrastructure inspection that may require operations in cluttered, complex, and partially unknown environments.

A Limited Battery Capacity

Most small multi-rotor UAVs have a limited battery capacity (typically < 30 minutes) which prevents them from being used for long-term or large-scale missions. When conducting some of the applications mentioned it is unrealistic to expect that they will be finished within this 30 minute time frame. This would require frequent battery swapping which is undesirable in many of these applications, since it requires a human operator to supervise the UAV. Manually swapping the batteries will need round-the-clock human presence for persistent operations. Alternatively, multiple UAVs may be needed which adds to the cost and maintenance requirements. There has been significant work that focuses on extending the lifetime of UAVs through new energy harvesting designs [80], automated battery swapping [112], stationary charging stations [8, 60, 97], swapping UAVs [60], low-level energy-efficient controllers [51], and low-level path planning [79].

An alternative is to place stationary recharging stations where the UAVs can periodically land to recharge. This dissertation shows how to plan the coverage paths when stationary recharging stations are placed in the environment. This is further extended to study how to plan persistent monitoring paths when an UGV is used as a mobile recharging station. The UAV can land on the UGV and the UGV can transport it to the next take-off site while recharging along the way. This behavior has been termed as symbiotic UAV-UGV

system [111]. This dissertation shows how to plan coverage paths with such a system.

B Lack of Prior Information

Often, planning efficient coverage paths requires prior knowledge of the environment. A major challenge is planning in environments where there is no prior information. Online planning algorithms to conduct 3D coverage of environments using information gathered in real-time are needed. A typical approach for online exploration uses the notion of frontiers [31, 127]. A frontier is the boundary between the known and the unknown regions in the environment. Various criteria such as nearest frontier and entropy/mutual information can be used to decide which frontier to drive to next. While frontier-based exploration can conduct 3D coverage, they do not focus coverage on regions-of-interest. For example, when inspecting a bridge, the UAV needs to focus only on the part of the environment that contains the bridge and ignore regions that do not contain the bridge. These algorithms also typically do not give guarantees on the performance. This dissertation investigates how to fuse the image data with the LiDAR data to conduct 3D semantic segmentation of the structure, so that exploration of regions of interest are the only focus. Better guarantees on the performance of the algorithm are given as well.

C Safe Operations in Practice

There are practical challenges for large-scale area coverage. These include flying without line-of-sight, especially along bridges and under the deck; hovering in place for long periods of time in windy conditions; and operating without GPS and compass measurements [42]. In many scenarios the GPS reception is typically noisy, if not completely absent. Furthermore, compass can be unreliable due to infrastructure surfaces or erroneous magnetic fields making

it even harder for operators to fly the UAV. The focus of this dissertation is on high-level path planning problems. Nevertheless, we design several algorithms that complement and use low-level routines that can address these practical challenges. The performance of the algorithms is demonstrated through proof-of-concept deployments.

1.2 Contributions

Table 1.1: Overview of the dissertation.

	Dissertation Overview	
	Region of Coverage	UAV Workspace
Ch. 2 [121, 122]	Points on the ground plane	Fixed altitude plane parallel to the ground
Ch. 3 [123]	Regions on the ground plane	Fixed altitude plane parallel to the ground
Ch. 4 [98, 124]	Surface area in 3D	3D planes parallel to the surface area to be covered
Ch. 5 [125]	Surface area in 3D	3D space

In this dissertation the three problems of point coverage, area coverage, and 3D coverage are studied. In the first two problems, point coverage and area coverage, covering sites of interest or 2D areas of interest on the ground plane using UAVs flying above them is the main focus (e.g., using downwards-facing sensors). In these problems, we focus specifically on overcoming the energy limitations of UAVs. The third problem that we study is 3D coverage. Here we focus on overcoming the challenges associated with lack of prior knowledge of the environment and safety constraints, when performing coverage in cluttered environments.

In Chapter 2, this dissertation starts with the problem of covering points of interest on the ground plane with a UAV with limited battery capacity. The limited battery capacity may prevent the UAV from covering all the points. The scenario where a UGV can act as a

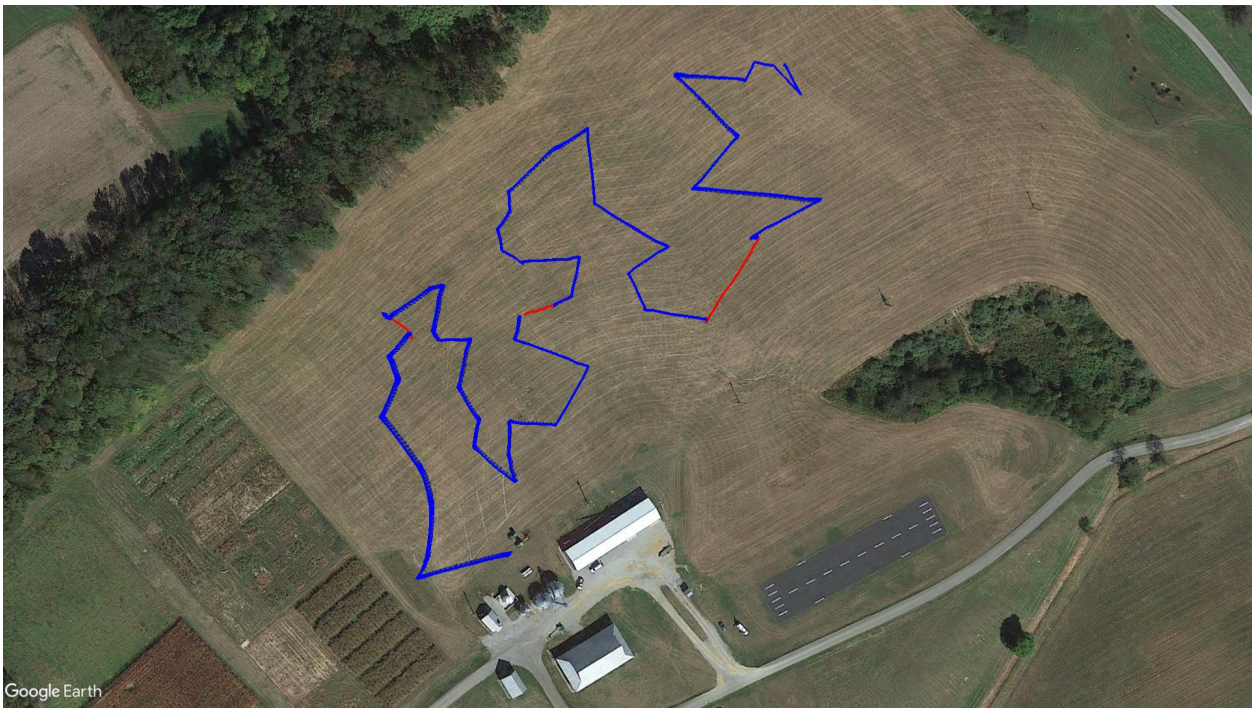


Figure 1.4: Point coverage of an environment.

mobile charging station for the UAV is considered. We design an algorithm that chooses when, where, and how much to recharge the battery of a UAV. The algorithm minimizes the total time it takes for the UAV to cover all points. This includes flight time, time to land and take-off, and recharging time. The problem is reduced to the GTSP [122] and a GTSP solver, GLNS [102], is used to find the optimal solution. We study variants of the algorithm where a stationary charging stations can be placed and multiple mobile recharging UGVs can be used. We carry out proof-of-concept experiments that involve fully autonomous UAV landings on the UGV and last over 30 minutes (Figure 1.4). The work reported in this chapter was presented at ICRA '18 [121] and later published in the Journal of Field Robotics [122].

In Chapter 3, we extend the work in Chapter 2 to UAVs conducting 2D area coverage. The input consists of a set of *boustrophedon cells* — rectangular strips whose width is equal to the field-of-view of the sensor on the UAV. The goal is to find a coordinated strategy for

Figure 1.5: Area coverage of an environment.

the UAV and UGV that visits and covers all cells in minimum time, while optimally finding how much to recharge, where to recharge, and when to recharge the battery. This includes flight time for visiting and covering all cells, recharging time, as well as the take-off and landing times. Similar to the point coverage problem, we show how to reduce this to a GTSP instance. Given an optimal GTSP solver, the approach finds the optimal coverage paths for the UAV and UGV. The formulation models multi-rotor UAVs as well as hybrid UAVs that can operate in fixed-wing and Vertical Take-off and Landing modes. We evaluate the algorithm through simulations and proof-of-concept experiments (Figure 1.5). This work was presented at ICRA '19 [123] and is in submission at a journal.

In Chapter 4, we expand the work to 3D surface area coverage, with infrastructure inspection as the motivation. We first show how to solve the inspection problem, when the model of the environment is known, by reducing it to the 2D area coverage problem. Instead of all boustrophedon cells on the ground, here, the cells are planar surfaces on the bridge that

Figure 1.6: 3D surface for coverage of a structure.

need to be covered. Not all cells are on the same plane, thus the planner needs to take into account the 3D flight cost. When navigating close to a bridge, GPS and compass data may be unreliable. Therefore, a low-level controller that uses LiDAR data to cover a boustrophedon cell is developed. We show how to adapt the 2D area coverage high-level planner to use these low-level routines. Lastly, an algorithm that flies along planner surfaces of infrastructures to allow for autonomous flight along planes on the infrastructure is implemented (Figure 1.7). This work was presented in ISER '19 [98] and a journal version is in revision.

In Chapter 5, the work in Chapter 4 is expanded upon to investigate a more general approach to 3D inspection. Unlike previous cases, the model of the environment is not known a priori. Thus, an online planner is developed. As described earlier, the coverage planner must focus only on regions that are of interest. We study the problem of visually inspecting the surface of a bridge using an UAV for defects. We do not assume that the geometric model of the bridge is known. The UAV is equipped with a LiDAR and RGB sensor that is used to build a 3D semantic map of the environment. Our planner, termed *GATSBI*, plans in an online fashion a path that is targeted towards inspecting all points on the surface of the bridge.

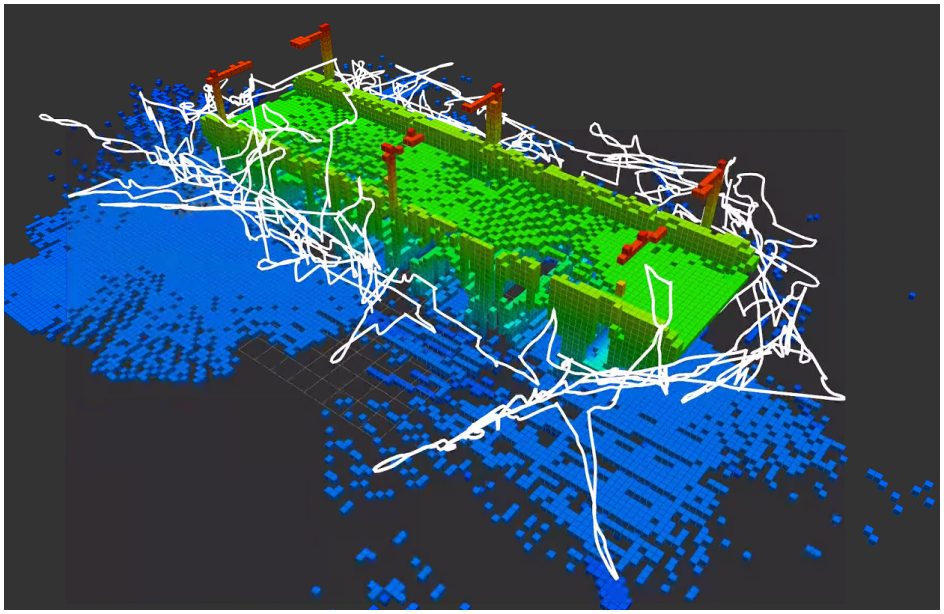


Figure 1.7: 3D surface for inspection of a bridge.

The input to GATSBI consists of a 3D occupancy grid map of the part of the environment seen by the UAV so far. We use semantic segmentation to segment the voxels into those that are part of the bridge and the surroundings. Inspecting a bridge voxel requires the UAV to take images from a desired viewing angle and distance. We then create a GTSP instance to cluster candidate viewpoints for inspecting the bridge voxels and use an off-the-shelf GTSP solver to find the optimal path for the given instance. As more parts of the environment are seen, we replan the path. We evaluate the performance of GATSBI through high-fidelity simulations conducted in Gazebo. We compare the performance of GATSBI with a frontier exploration algorithm. Our evaluation reveals that targeting the inspection to only the segmented bridge voxels and planning carefully using a GTSP solver leads to more efficient inspection than the baseline algorithms. This work is under review at IROS '21 [125].

Chapter 2

2D Point Coverage with a Symbiotic UAV+UGV System

We study the problem of planning a tour for an energy-limited UAV to visit a set of sites in the least amount of time. We envision scenarios where the UAV can be recharged at a site or along an edge either by landing on stationary recharging stations or on UGVs acting as mobile recharging stations. This leads to a new variant of the Traveling Salesperson Problem (TSP) with mobile recharging stations. We present an algorithm that finds not only the order in which to visit the sites but also when and where to land on the charging stations to recharge. The algorithm plans tours for the UGVs as well as determines the best locations to place stationary charging stations.

We study three variants for charging: multiple stationary charging stations; single mobile charging station; and multiple mobile charging stations. While the problems we study are NP-Hard, we present a practical solution using Generalized TSP that finds the optimal solution that minimizes the total time, subject to discretization of battery levels. If the UGVs are slower than the UAVs, the algorithm also finds the minimum number of UGVs required to support the UAV mission such that the UAV is not required to wait for the UGV. The simulation results show that the running time is acceptable for reasonably sized instances in practice. We evaluate the performance of the algorithm through simulations and proof-of-concept field experiments with a fully autonomous system of one UAV and UGV.

Figure 2.1: DJI F450 with Pixhawk autopilot running APM with onboard Jetson TX1 along with the Clearpath Husky UGV used for the field experiments.

The rest of the chapter is organized as follows. We begin with a discussion of the related work in Section 2.1. We describe the problem formulation and assumptions in Section 2.2. The algorithm is described in Section 2.3. We evaluate the performance of the algorithm through simulations reported in Section 2.4 and through proof-of-concept experiments reported in Section 2.5. We conclude with a summary of this chapter.

2.1 Related Work

In this section we briefly describe the work related to UAV recharging stations.

A Recharging and Replacing UAV Batteries

A number of solutions for autonomous charging of UAVs have been proposed in the recent past. Cocchioni et al. presented a vision system to align the UAV with a stationary charging station [23]. A similar design was presented by Mulgaonkar and Kumar including magnetic contact points which improved the landing of the UAV [81]. There are also commercial

products (*e.g.*, the SkySense system [10]) that provide similar capabilities.

The alternative to recharging a discharged battery is to swap it with a charged one. Swieringa et al. presented a "cold" swap system for exchanging the batteries for one or more helicopters, which is where the system is turned off and the battery is swapped out [109]. The authors evaluated their system through simulations with three helicopters where they demonstrated an increase in system lifetime from six minutes to thirty two minutes. Toksoz et al. presented the design of a stationary battery swapping station for multi-rotor systems [112]. Their design has a "dual-drum structure" that can hold a maximum of eight batteries which can be "hot" swapped. "Hot" swapping being the changing of batteries without shutting down the system.

Kemper et al. investigated the needs of consumers and design constraints for battery recharging stations for helicopters [58]. They performed a cost-analysis of existing designs and consumer needs for recharging and swapping of batteries. Suzuki et al. followed up on this work [108] by further analyzing two types of landing platforms proposed in [58].

The work presented in this chapter is complementary to these hardware designs — any of the existing systems could be leveraged. Instead we show how to optimize the performance by careful placement of charging stations or planning of paths for mobile charging stations.

B Planning for Energy Limited UAVs

A typical strategy to deal with limited battery capacity of UAVs is to use multiple robots with possible redundancy built in. Derenick et al. presented a control strategy to carry out persistent coverage missions with robot teams which balances a weighted sum of mission performance and the safety of the UAVs [32]. The UAVs reconfigure based on their energy levels and coverage performance. Mitchell et al. presented an online approach for maintaining

formations while substituting UAVs running low on charge with recharged UAVs [77].

There have been work on planning the paths for UAVs using multiple stationary recharging stations [8, 60, 97]. Kim et al. used Mixed Integer Linear Programing (MILP) to allow teams of robots to trade off the task objective within the team when some UAVs are low on energy [60]. Shakhatreh et al. studied the problem of maintaining persistent coverage of an area by partitioning the coverage tour amongst the UAVs [97]. Ahmed et al. presented a method to discretize the state space which allows for an algorithm to minimize the energy used by UAVs for longer missions [8]. Liu and Michael presented a matching algorithm for assigning UAVs with UGVs acting as recharging stations [69].

Our work falls broadly in the category of area coverage with energy-limited UAVs. Franco et al. presented an algorithm that reduces the energy consumption of a UAV while satisfying coverage and resolution requirements [34]. Sipahioglu et al. considered a similar problem [101]. Their algorithm first finds a route that ensures complete coverage and then partitions the route among multiple robots by considering respective energy capacities. Most recently, Wei and Isler presented a approximation algorithm for covering a polygonal grid environment where the UAV is allowed to return back to a fixed basestation multiple times for recharging [115]. However, they do not address the problem of optimizing the recharging locations.

In the previous work [111], it is shown how to plan tours for a symbiotic UAV+UGV where the UGV can mule the UAV between two deployment locations such that the UAV does not spend any energy. However, the previous work did not model the capability of UGV recharging the UAV along the way. Consequently, the goal was to maximize the number of sites that can be visited in a single charge. This results in a variant of an NP-Hard problem known as orienteering [16]. In the current work, a more general model which has the added complication of keeping track of the energy level of the UAV as well as deciding where and

how much to charge along the tour is allowed.

The work by Rathinam et al. studies how to plan paths for mobile charging stations [106]. The authors of this paper study the problem of planning UAV paths with fuel constraints and stationary refueling locations. The algorithms that the author presents allow for the planning of multiple UAVs to visit every site. This paper is slightly different than the work in this chapter in the sense that the refueling stations are stationary and the UAV has to adjust to the refueling station. In this chapter we allow the refueling station to be mobile and adjust for the UAV allowing for shorter tour times.

The work most closely related to ours is that of Maini and Sujit. They present an algorithm that plans paths for one UAV and one recharging UGV to carry out surveillance in an area [72]. The UGV moves on a road network. The authors create an initial path for the UGV and then create a path for the UAV. In this chapter, we simultaneously create paths for the UAV and UGV. We compare the empirical performance of the algorithm with that from [72] in Section 2.4. We find that the presented algorithm outperforms this baseline. Additionally, we guarantee that the algorithm finds the optimal solution for the problem, conditioned on the discretization unlike that from [72].

C Vehicle Routing Problems

The algorithm is based on a solution to the Generalized Traveling Salesperson Problem. Both, GTSP and TSP, belong to the class of vehicle routing problems [63]. Variants of TSP have been used for many robotic applications such as routing [63], surveillance [96], and patrolling [88]. Solutions to GTSP have been used in applications such as location-routing, loop material flow system design, post-box collection, stochastic vehicle routing and arc routing [64].

Both, TSP and GTSP are NP-Hard problems. If the costs are Euclidean distances, then there exists an efficient polynomial time approximation scheme for solving TSP [85]. However, there does not exist a constant-factor approximation for solving Euclidean GTSP. Nevertheless, a number of solvers have been developed that can find good solutions for TSP and GTSP in practice [11, 102]. In this chapter, we use two state-of-the-art solvers, concorde [11] and GLNS [102], for solving TSP and GTSP respectively.

D Autonomous UAV Landing

There are many studies on controls and planning techniques for autonomous landing of aerial vehicles. Kong et al. surveyed various autonomous landing methods for five classes of aerial vehicles (full-scale, medium-scale, small-scale, mini-scale, and micro) [61]. A common approach is to use vision-based autonomous landing [47, 65, 86]. Oh et al. presented a technique for landing on a swaying ship in the ocean [86]. Kim et al. presented the design of a system that can land on a mobile platform using color-based detection with an RGB camera [61]. This approach may not be robust due to inconsistent lighting conditions. Instead, we use an IR-camera along with IR beacon for detection and landing on the recharging station. This approach is robust to poor lighting conditions, indifferent to color, and can deal with other IR sources when the exposure is adjusted correctly.

Goldin and Dougherty investigated the ability for quadrotors to land on inclined surfaces (i.e., perching) [37, 43]. The system proposed by Dougherty used downward-facing laser sensors to measure the distance to the landing platform and to detect the angle of the platform [37]. The system was demonstrated to work for platforms with up to 30° incline. Goldin presented a completely autonomous system that allows for real-time simultaneous localization and mapping for perching with UAVs [43].

Our main novelty is in jointly optimizing the routes for the UAV as well as the mobile recharging stations (equivalently, placement of stationary recharging stations). The exact formulation is defined in the next section.

2.2 Problem Formulation

The input to the algorithm is a set of n sites, x_i, that must be visited by the UAV. We use x_i to refer to a site as well as the vector representing its coordinates. We start with a list of common assumptions:

1. unit rate of discharge (1% per second);
2. UAV has an initial battery charge of 100%;
3. UAV and UGVs start at a common *depot*, d;
4. all the sites are at the same altitude;
5. UGVs have unlimited fuel/battery capacity;
6. UAV can fly between any two sites if it starts at 100% battery level.

All but assumption 5 are only for the sake of convenience and ease of presentation and can be easily relaxed. Although UGVs cannot have unlimited operational time, it is a reasonable assumption since UGVs can have much larger batteries or can be refueled quickly. In this work, we only allow for the UAV to be recharged at a site or while being ferried between two sites by the UGV. It is possible to extend the algorithm to also allow for recharging at arbitrary sites, as we discuss in Section 2.6.

We also provide a list of standard terminology that will be used throughout this chapter:

- x_i denotes the i^{th} site that must be visited[1] by flying to a fixed altitude;
- r represents the time required to recharge the battery by a unit %;
- t_{TO} is the time it takes to take off from the UGV;
- t_L is the time it takes to land on the UGV;
- $t_{UAV}(x_i, x_j)$ and $t_{UGV}(x_i, x_j)$ give the time taken by the UAV and UGV to travel from x_i to x_j.

Suppose Π is a path that visits the sites in the order given by $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$ where $\sigma(j) = i$ implies x_i is the j^{th} point visited along Π. The cost of an edge from $x_{\sigma(j)}$ to $x_{\sigma(j+1)}$ along Π depends on whether the UAV flies between the two sites or if it is muled by the UGV between the two sites while being recharged (denoted as a **TYPE II** edge, defined formally in Section A). Let k and k' be the battery levels at $\sigma(j)$ and $\sigma(j+1)$. Therefore,

$$T(j, j+1) = \begin{cases} t_{UAV}(x_{\sigma(j)}, x_{\sigma(j+1)}) \\ \max\{t_{UGV}(x_{\sigma(j)}, x_{\sigma(j+1)}), r(k' - k)\} \end{cases} \tag{2.1}$$

In addition, there are non-zero node costs if the UAV is charged from battery level k to k' at a site x_i rather than along an edge:

$$T(j) = r(k' - k). \tag{2.2}$$

Therefore, the total path cost is given by,

$$T(\Pi) = T(1) + \sum_{j=1}^{n-1} T(j+1) + T(j, j+1) \tag{2.3}$$

[1]Note that x_i does not mean that is the i^{th} point that will be visited. The order of visiting the points is determined by the algorithm.

We are now ready to define the problems studied in this chapter.

Problem 1 (Multiple Stationary Charging Stations (MSCS))**.** Given a set of sites, x_i, to be visited by the UAV, find a path Π^* for the UAV that visits all the sites as well as selects one or more sites (if needed) to place recharging stations so as to minimize the total time given by Equation 2.3 under the assumptions 1–6 given above.

Problem 2 (Single Mobile Charging Station (SMCS))**.** Given a set of sites, x_i, to be visited by the UAV, find a path Π^* for the UAV that visits all the sites as well as another path for the UGV acting as a mobile basestation so as to minimize the total time given by Equation 2.3 under the assumptions 1–5 given above. Assume that the UAV and UGV travel at the same speed and must travel the same distance between any two sites.

The assumption that the UGV is as fast as the UAV is not necessary to find a solution; it is required to guarantee optimality for one UGV. If the UGV is slower than the UAV, we can still use the paths returned by the algorithm for one UGV, but the UAV may have to wait. An alternative is to minimize the number of UGVs required to ensure the UAV never has to wait for a recharging station.

Problem 3 (Multiple Mobile Charging Stations (MMCS))**.** Given a path, Π^*, for a UAV and a set of charging sites as well as **Type II** edges for the UGVs, find the minimum number of slower UGVs necessary to service the UAV, without the UAV having to wait for a UGV under the assumptions 1–5 given above. The input to MMCS is obtained by solving SMCS, without assuming the single UGV in SMCS is as fast as the UAV. The paths for the UGVs are allowed to start at any site.

Our main contribution is a GTSP-based algorithm that solves the first two problems optimally and an Integer Linear Programming (ILP)-based algorithm that solves Problem 3. As mentioned previously, Problems 1 and 2 are NP-Hard and consequently finding optimal

algorithms with running time polynomial in n is infeasible under standard assumptions. Instead, we provide a practical solution that is able to solve the three problems in reasonable time (quantified in Section 2.4).

2.3 GTSP-Based Algorithm

In this section we show how to formulate Problems 1 and 2 as GTSP instances [85]. We first formally define GTSP. Let G be a graph (can be directed) with vertices V and edges E. Each edge $e \in E$ has a corresponding cost c. The vertices are grouped into m mutually exclusive clusters. An edge exists only between vertices belonging to different clusters. The Generalized Traveling Salesperson Problem asks for a minimum cost cycle which includes exactly one vertex from each cluster. When each cluster contains only one vertex, the GTSP reduces to TSP.

Solving GTSP is at least as hard as solving TSP. However, Noon and Bean [85] presented a technique to convert any GTSP input instance into an equivalent TSP instance on a modified graph such that finding the optimal TSP tour in the modified graph yields the optimal GTSP tour in the original graph. We can solve GTSP by solving TSP optimally using a numerical solver and we use *concorde* [11], which is the state-of-the-art TSP solver or GLNS [102], that is a heuristics-based GTSP solver. The results in Section 2.4 show that GLNS is significantly faster than *concorde*. However, only the *concorde* approach is guaranteed to find the optimal solution.

We start by showing how to formulate the SMCS and MSCS problems as GTSP instances. After obtaining an output, we can convert the TSP solution back into a GTSP solution, then into a solution for the SMCS or MSCS problems. The process of converting SMCS and MSCS into TSP is the same. Only the process of converting the solution of TSP to solutions

of SMCS and MSCS differ.

A Transforming SMCS/MSCS to GTSP

Given an SMCS or MSCS instance, we show how to create a GTSP instance consisting of a directed graph where the vertices are partitioned into non-overlapping clusters. We create one cluster, g_i, for each input site x_i. Each cluster, g_i has m vertices, each one corresponding to a discretized battery level. That is, $g_i = \{x_i^k \mid \forall i \in [1:n], \forall k \in \{1, 2, \ldots, m\}\}$. x_i^k represents the UAV reaching site x_i with $k\frac{100\%}{m}$ battery remaining. m is an input discretization parameter. Figure 2.2 shows the six clusters for six input sites with $m = 5$.

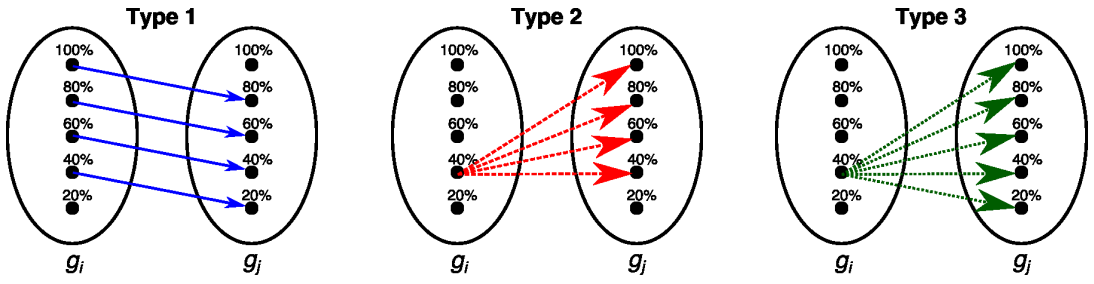


Figure 2.2: Different types of edges that are created. **Type I** are the edges where the UAV flies directly between the two sites and the battery level strictly decreases. **Type II** are the edges where the UGV carries and recharges the UAV. Lastly, **Type III** are the edges where the UAV and UGV meet up to charge at a site. The depot station d is shown on the graph. Note that only a subset of all possible edges are shown.

Next we describe how to create the edges amongst the vertices in the n clusters. We create three types of edges. **Type I** edge between x_i^k and $x_j^{k'}$ models the case where the UAV directly flies from x_i to x_j. The cost of a **Type I** edge is given by:

$$T_{\mathrm{I}}(x_i^k, x_j^{k'}) = t_{UAV}(x_i, x_j)$$

A **Type I** edge exists between x_i^k and $x_j^{k'}$ if and only if $k - k'$ equals the distance between x_i

and x_j. For ease of exposition, we assume that taking-off and landing energy consumption is negligible. Nevertheless, we can easily incorporate this in the edge definitions. These types of edges are shown in Figure 2.2-left.

A **TYPE II** edge from x_i^k to $x_j^{k'}$ models the UAV landing on the UGV at x_i and recharging while being muled to x_j by the UGV. The cost of a **TYPE II** edge is given by:

$$T_{\text{II}}(x_i^k, x_j^{k'}) = \max(r(k' - k), t_{UGV}(x_i, x_j)) + t_{TO} + t_L$$

The cost is the maximum of the time taken to recharge from k to k' and the time it takes the UGV to travel from x_i to x_j. Note that a **TYPE II** edge exists only if $k' \geq k$. **TYPE II** edges are shown in Figure 2.2-middle.

Finally, there are **TYPE III** edges that represent the UAV flying from x_i to x_j and then landing on the UGV at x_j and recharging up to k' battery level. The cost of a **TYPE III** edge is given by:

$$T_{\text{III}}(x_i^k, x_j^{k'}) = t_{UAV}(x_i, x_j) + r(k' - k + ||x_i - x_j||_2) + t_{TO} + t_L$$

A **TYPE III** edge exists if and only if $k' \geq k - ||x_i - x_j||_2$. Figure 2.2-right shows the **TYPE III** edges.

Only **TYPE I** and **TYPE III** edges exist when solving MSCS whereas all three edges are possible when solving MMCS. Note that **TYPE II** and **TYPE III** edges require the UAV to take off and land at every site. This prevents the UAV from not taking off between two consecutive **TYPE II** edges. This is because in order to visit a site it must fly to a fixed altitude to consider the site visited.

There are certain pairs of vertices for which more than one type of edge may be allowed. In such a case, we pick the minimum of the three edge costs (assuming the edge cost is ∞ if

the edge does not exist) and assign the minimum cost for the edge. That is, the edge cost $T(x_i^k, x_j^{k'})$ is given by:

$$T(x_i^k, x_j^{k'}) = \min\{T_{\mathrm{I}}(x_i^k, x_j^{k'}), T_{\mathrm{II}}(x_i^k, x_j^{k'}), T_{\mathrm{III}}(x_i^k, x_j^{k'})\}$$

We also create an $n+1^{th}$ cluster containing a dummy vertex called the depot, d. We add a zero cost edge from d to all vertices, x_i^k, with $k = m$ and edges from all vertices back to d. The reason to create a depot node is that the TSP solver finds a closed tour whereas we are interested in finding paths.[2] The depot node serves to ensure that we can find a closed tour without charging for the extra edges.

B Converting Optimal TSP Tour to UAV and UGV Paths

An optimal TSP tour immediately yields an optimal GTSP solution. The order in which the clusters are visited gives the sequence of vertices on the UAV paths. What remains is deciding the UGV path for SMCS and recharging station placements for MSCS.

In MSCS, we only have **Type I** and **Type III** edges. If a **Type III** edge, say from x_i^k to $x_j^{k'}$, appears in the GTSP solution, then we will place a recharging station at the site x_j. No recharging stations are placed for **Type I** edges in the solution.

In MMCS, all three edges are possible, whereas only **Type I** and **Type II** in SMCS. We check the type of each edge in the GTSP solution, one by one. If a **Type I** edge appears in the GTSP solution, then it does not affect the UGV tour. If a **Type II** edge, say from x_i^k to $x_j^{k'}$, appears in the GTSP solution, we add x_i and x_j to the UGV path (in this order). If a **Type III** edge, say from x_i^k to $x_j^{k'}$, appears in the GTSP solution we add only x_j to the UGV path.

[2] A path visits a vertex exactly once whereas a tour has the same starting and ending vertices.

The UGV path, as a result, visits a subset of the input sites. If the UGV is slower than the UAV, then it is possible that the UAV will reach a site before the UGV does and will be forced to wait. We implement an ILP that allows me to solve for the minimum number of UGVs necessary to service the UAV without waiting (shown in Section C).

Theorem 2.1. *The GTSP-Based algorithm finds the optimal solution for SMCS and MSCS assuming that there exists an optimal TSP solution.*

The proof follows directly from the proof of optimality of the GTSP reduction given by Noon and Bean [85].

Problem 2 assumes that the UGV is as fast as the UAV, and therefore only one UGV suffices. Next, we show how to address the case when the UGV is slower than the UAV. Multiple UGVs may be required. We show how to minimize the number of UGVs.

C Solution for Problem 3

We present a solution to the MMCS problem based on an ILP formulation. The input is obtained by solving Problem 2, where a UAV path, a set of UGV sites and **TYPE II** edges are given. The UGV path visits only a subset of the sites in $\{x_i\}$. We denote these sites by $\{g_1, g_2, \ldots, g_l\}$, where $l \leq n$. For each directed edge from g_i to g_j, where $j > i$, we associate a binary decision variable y_{ij}. The solution to Problem 3 will assign binary values to all y_{ij}. If y_{ij} equals 1, then some UGV will travel from g_i to g_j. If y_{ij} equals 0, then no UGV will travel from g_i to g_j. Note that j does not necessarily have to be $i+1$ but must satisfy $j > i$.

If there is a site g_e such that a solution has one incoming edge $y_{de} = 1$ and one outgoing edge $y_{ef} = 1$, then the same UGV can be used for both edges (Figure 2.3). That is, the site g_e is in the middle of some UGV path. If there is a site g_c such that the solution has an

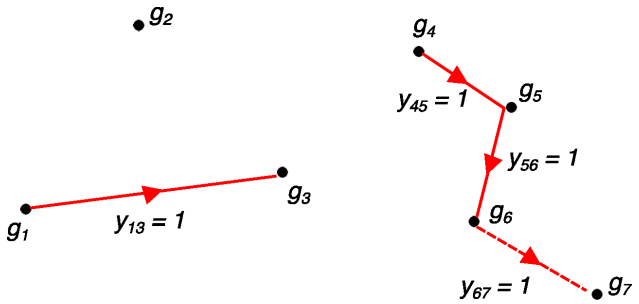


Figure 2.3: Example case of UGV path with site and edge labels. The binary decision variables that are equal to 1 are shown; all other y_{ij} are returned as 0 by the solver. g_2 has no incoming or outgoing edge selected and requires a separate UGV. g_1 and g_4 have only one outgoing edge selected and therefore indicate the start of a new UGV path. Similarly, g_3 and g_7 have only one incoming edge and are therefore the end of a UGV path. g_5 and g_6 have incoming and outgoing edges and are therefore in the middle of a path. The edge between g_6 and g_7 is **Type II** and therefore y_{67} is forced to be selected always.

incoming edge $y_{ac} = 1$ or a site g_a such that the solution has an outgoing edge $y_{ac} = 1$ but not both, then the corresponding UGV path either ends or starts at g_c or g_a respectively. If there is a site g_b such that a solution has no incoming or outgoing edge selected, then the corresponding UGV path visits only g_b and no other site. The number of distinct UGV paths selected is given by l minus the total number of edges selected. Therefore, to minimize the number of UGVs required, we maximize the number of edges selected.

Also associated with each edge is the time to come for the UAV and UGV denoted by $T^{\Pi^*}(g_i, g_j)$ and $t_{UGV}(g_i, g_j)$ respectively. Here $T^{\Pi^*}(g_i, g_j)$ is the time taken by the UAV to fly the subpath of Π^* from g_i to g_j, which may contain some intermediate sites. $t_{UGV}(g_i, g_j)$, on the other hand is the time for the UGV to directly go from g_i to g_j.

Using the above notation we present the following ILP formulation:

$$\max \sum_{i=1}^{l-1} \sum_{j=i+1}^{l} y_{ij} \tag{2.4}$$

subject to:

$$\sum_{i=1}^{j-1} y_{ij} \leq 1 \ \forall \ j, \tag{2.5}$$

$$\sum_{j=i+1}^{l} y_{ij} \leq 1 \ \forall \ j, \tag{2.6}$$

$$y_{ij} = 0 \text{ if } T^{\Pi^*}(g_i, g_j) < t_{UGV}(g_i, g_j), \text{ and} \tag{2.7}$$

$$y_{ij} = 1 \text{ if } y_{ij} \text{ is TYPE II edge.} \tag{2.8}$$

Equation 2.5 and 2.6 only allow a maximum of one incoming edge and a maximum of one outgoing edge. The constraint given by Equation 2.7 removes all UGV edges where the UAV would have to wait for the UGV. Lastly Equation 2.8 forces the problem to use TYPE II edges if present.

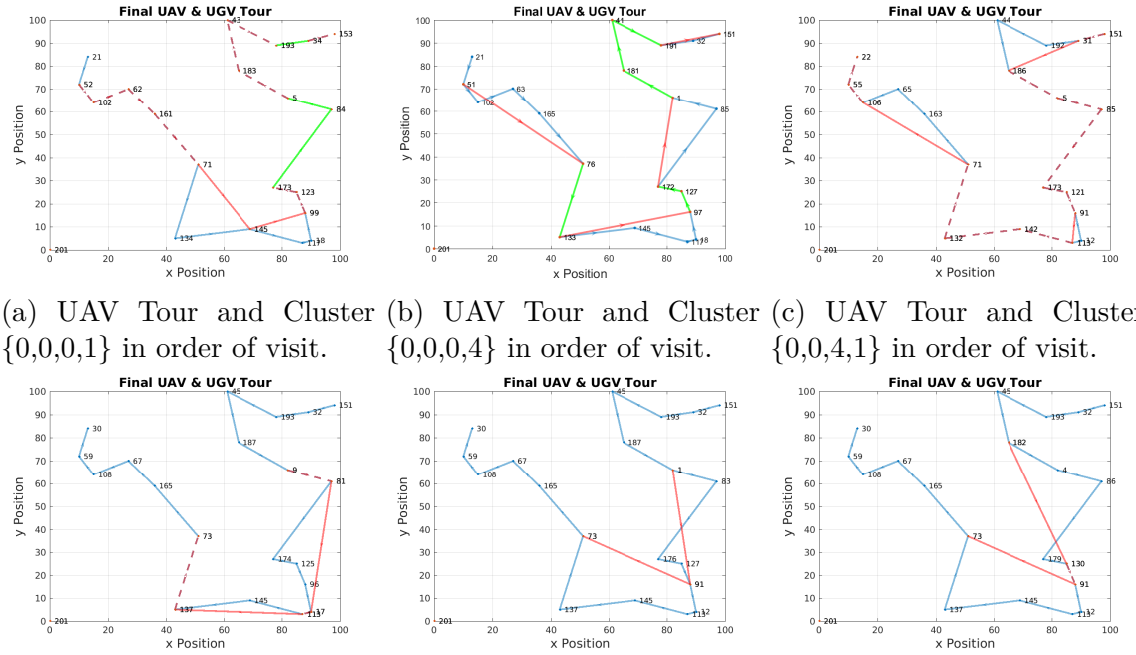
2.4 Simulations

In this section, we present simulation results using the proposed algorithm. The implementation is available online.[3] We conduct four types of simulation studies. First, study the effect of the input parameters and computational time for SMCS (MSCS has similar computational cost as SMCS). Next, we compare the performance of the algorithm for SMCS with that of [72]. Finally, we present simulation results for MMCS.

A Effect of the Parameters for SMCS

Figure 2.4 shows the outputs obtained for different configurations of the t_{TO}, t_L, r, t_{UGV} parameters for the same 20 input sites and with $m = 10$ battery levels. Each figure has the

[3] https://github.com/raaslab/Heterogeneous-Teams



(a) UAV Tour and Cluster {0,0,0,1} in order of visit.
(b) UAV Tour and Cluster {0,0,0,4} in order of visit.
(c) UAV Tour and Cluster {0,0,4,1} in order of visit.
(d) UAV Tour and Cluster {4,4,0,1} in order of visit.
(e) UAV Tour and Cluster {4,4,0,4} in order of visit.
(f) UAV Tour and Cluster {4,4,4,4} in order of visit.

Figure 2.4: The above figures are multiple runs using the same initial points. We use 20 sites with 10 battery levels. Each figure has the vertex number in the GTSP input graph and a set in the form {W,X,Y,Z} in the caption. This set {W,X,Y,Z} denotes: $t_{TO} = W$, $t_L = X$, $r = Y$, and $t_{UGV} = Zt_{UAV}$. The colors represent different edge types with blue being only UAV travel, red being only UGV travel, green being UAV and UGV travel separate, and dashed red being UAV+UGV travel together.

UAV+UGV tour with blue solid edges (only UAV), red solid edges (only UGV), green solid edges (UAV and UGV separate), and red dashed edges (UAV+UGV together).

The following intuitive observations using the six cases are made, shown in Figure 2.4:

- $t_{TO} = 0$, $t_L = 0$ and $r = 0$: UAV does not differentiate between the type of the edge because there is no penalty to recharge (Figure 2.4a);
- $t_{TO} + t_L > 0$: recharging has a penalty and as such the number of recharging stops are reduced (Figures 2.4d, 2.4e and 2.4f);
- $t_{TO} = 0$, $t_L = 0$, $r = 0$ and $t_{UGV} > t_{UAV}$: the UAV will use **Type III** edges for charging

because t_{UGV} will make **TYPE II** edges higher cost (Figure 2.4b and 2.4e);

- $t_{TO} = 0$, $t_L = 0$, $r > 0$ and $t_{UGV} = t_{UAV}$: the UAV will use **TYPE II** edges for charging instead of **TYPE III** edges (Figure 2.4c);

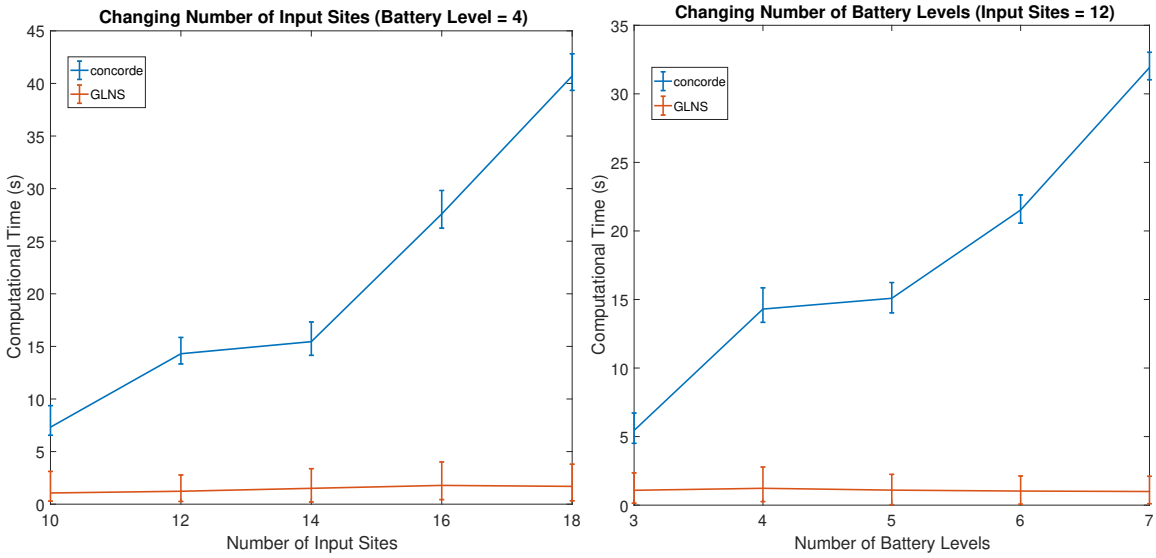
It is observed that the recharge time r and UGV speed t_{UGV} affect which type of edges are used. If the time it takes to recharge is much larger than t_{UGV} then the UAV will favor **TYPE II** edges and when the time it takes to recharge is much less than t_{UGV} then the UAV will favor **TYPE III** edges.

B Computational Time for SMCS

We use two solvers for the SMCS (and MSCS) problem. When using *concorde* an optimal solution is obtained, but with more computational time. Therefore, it can only solve smaller instances. The GLNS solver can solve larger instances, but cannot always guarantee optimality. Nevertheless, it is observed that the GLNS was able to find the optimal solution for all of the cases reported in Figure 2.5.

Figure 2.5 shows a direct comparison of the computational time of the two methods. We compared the two methods by first varying the amount of input sites (Figure 2.5a), with $m = 4$, and then comparing the two by varying the amount of battery levels (Figure 2.5b), given $n = 12$. We ran 10 trials with random input sites. We plot the average value along with the maximum and minimum value.

Due to limitations in *concorde*, larger instances were not able to be run, but GLNS could run larger instances, as shown in Figure 2.6. We show the effect of incrementing n from 20 to 50 and m from 50 to 150 in Figure 2.6. We ran 5 random instances and plot the average of the 5 instances with the maximum and minimum value. The simulations show that GLNS



(a) Computational time with battery levels set to 4.

(b) Computational time with input sites set to 12.

Figure 2.5: Comparison of computational times of the GTSP to TSP transformation [85] solved using *concorde* and the direct GTSP solver, GLNS [102] on the default "medium" setting. The output costs for the *concorde* and GLNS solutions were the same for the given instances, but may differ for larger instances. This was done over 10 trials.

is able to solve larger sized instances in reasonable amount of time.

C Comparison With Baseline Algorithm for SMCS

We compare the proposed GTSP-based algorithm with the method presented by Maini and Sujit [72]. We refer to the algorithm presented in [72] as the baseline method. There are some minor differences in the two problem formulations. The method proposed overcomes many of the limitations in the baseline method. In the baseline method, the UGV is restricted to navigate only on a given road network which is assumed to be a tree. The UGV must remain stationary while recharging the UAV in the baseline method whereas we allow for the UGV to recharge the UAV while carrying it to the next deploying site. Furthermore, it is implicitly assumed that the UAV's battery is fully recharged every time it lands on the UGV whereas

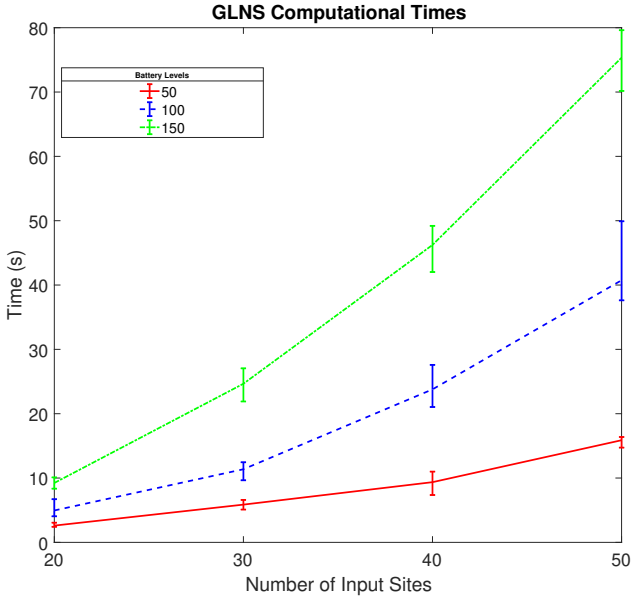
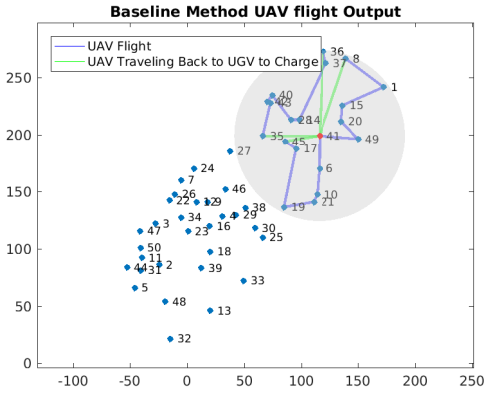


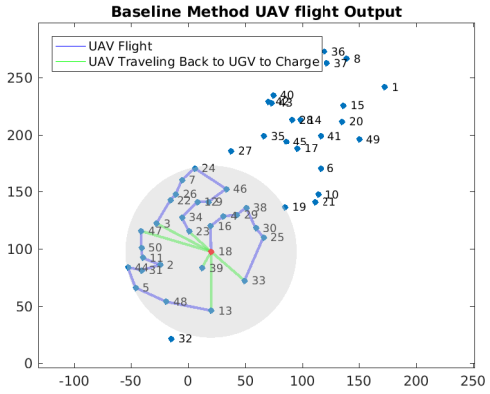
Figure 2.6: The computational time of GLNS for larger instances. This was done for 5 random instances with the average, max, and min plotted as well. These problem instances could not be solved by *concorde*.

we provide as output the charging schedule as well. The method proposed keeps track of the UAV's battery level, thereby allowing it to recharge at any level (not necessarily when fully depleted) and potentially not recharge to 100%, if it is not needed. As a result, the method proposed produces paths that visit the set of input sites in lesser time. We present the empirical results next.

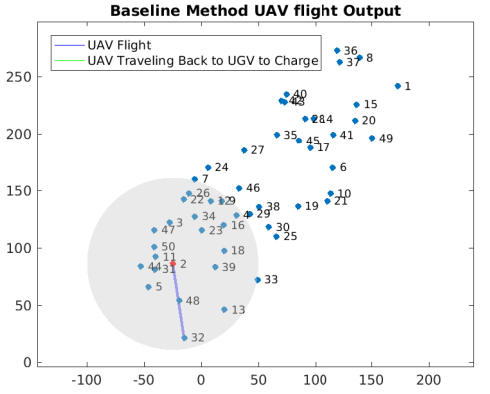
Figure 2.7 shows a sample output from the baseline method. The method starts by discretizing the edges of the UGV roadmap. A disk of radius $B/2$ is placed at each discretized point, where B is the energy budget (assuming unit rate of discharge and unit speed). A hitting set problem is then solved to find the minimum number of disks required to cover all the sites that need to be visited by the UAV. The UAV then visits all the sites within each chosen disk, while the UGV waits at the center of the disks. The UAV may return to the center multiple times, if the tour within a disk is longer than B. The UGV's path is found using depth-first search over the roadmap to visit all the chosen disk centers.



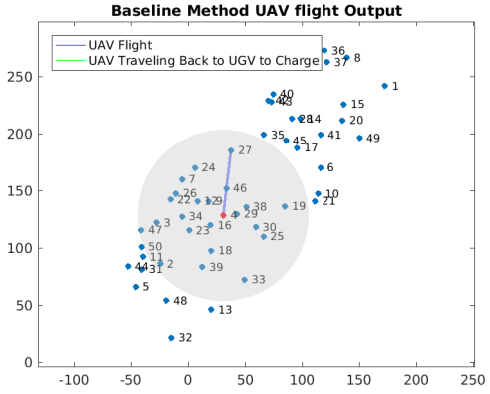
(a) First UAV flight.



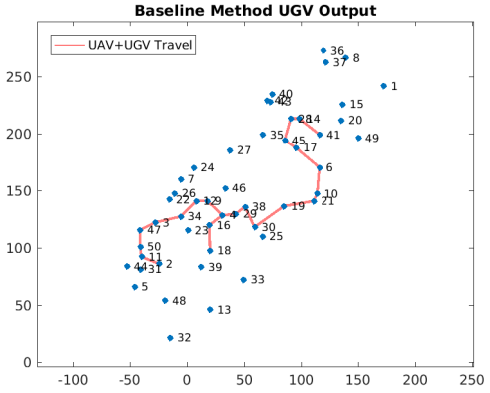
(b) Second UAV flight.



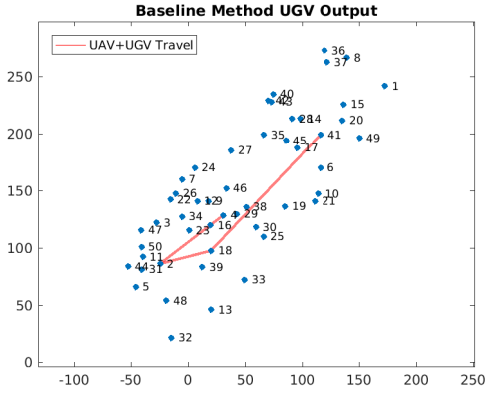
(c) Third UAV flight.



(d) Fourth UAV flight.

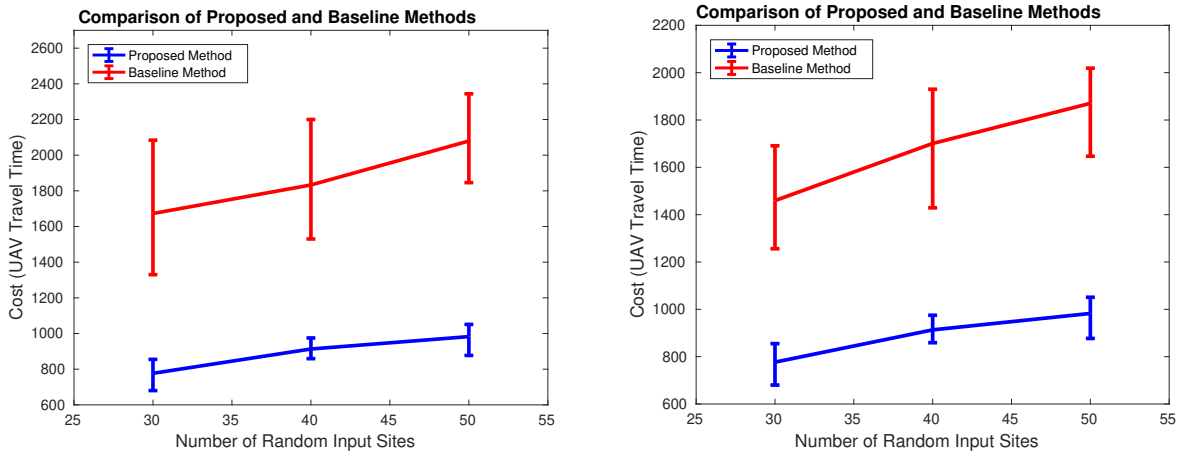


(e) UGV path using DFS.



(f) UGV path using TSP.

Figure 2.7: Representative output of the baseline method [72]. The number of sites randomly chosen were 50 with 100 battery levels and 150 meter budget (assuming unit rate of discharge and unit speed).



(a) Depth First Search method.

(b) TSP method.

Figure 2.8: Comparison between the method proposed in this chapter and the baseline method [72]. We also consider a variant of the baseline method that uses a TSP solver to calculate the UGV path instead of depth first search. All graphs show mean and variance of 10 random trials. We kept the number of battery levels the same at 100 and allow an energy budget of 150 meters.

Since the algorithm does not take a UGV roadmap as input, we create such a roadmap by constructing a minimum spanning tree of all the input sites. We use *concorde* to solve for the UAV tours within each disk and add detours to visit the center if the tour exceeds the budget B (green edges in Figure 2.7). We find the minimum hitting set by greedily choosing the disk that contains the most uncovered sites and continuing until all sites are covered. In addition to using depth-first search for finding the UGV path, we also find a TSP tour of the chosen disk centers which results in shorter paths. Nevertheless, the baseline method still results in longer tours as compared to the method proposed.

Figure 2.8 shows the cost of tours generated by the baseline method (with DFS and TSP) and the method proposed. The figures show the total cost of the UAV tour which includes the UAV flight time as well as the time it would take to return to the UGV, land, recharge, take-off, and then return to the path. The sites are generated randomly in an area of 300×110 meters. The energy budget is set to $B = 150$. The baseline method returns path that take

almost twice as long as the method proposed. We attribute this to the fact that we allow the UGV to recharge while carrying the UAV and not always recharge to 100% or recharge only when completely depleted.

D Number of UGVs required in MMCS

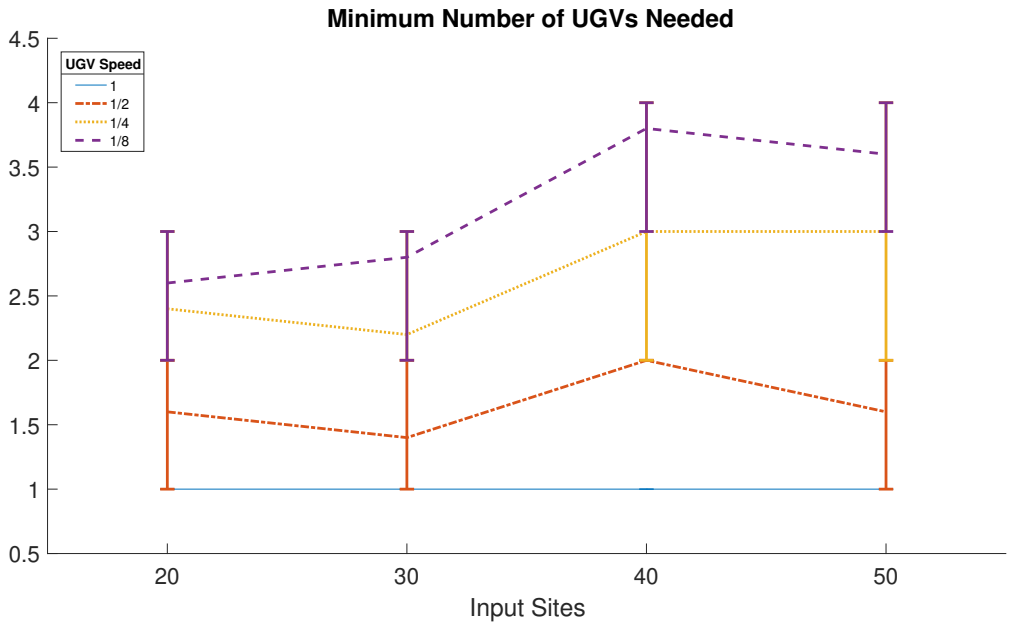


Figure 2.9: Minimum number of UGVs necessary to service an UAV with multiple UGV speeds. The UGV speed in this figure is in meters/second. The input was the same 5 random trails that were used for $m = 50$ data from Figure 2.6.

For Problem 3, we study the effects of a slower UGV on the number of UGVs required to service the same set of sites. The ILP was solved using the Integer Programming toolbox in MATLAB. Figure 2.9 shows the minimum number of UGVs necessary to service a single UAV as a function of the relative speeds. We used the same $m = 50$ data set that was used to create Figure 2.6. As expected, the ILP solution uses only one UGV when the UAV and UGV have equal speeds. As the relative speed of the UGV decreases, more UGVs are required on an average. The actual trend depends on the specific configuration of sites.

2.5 Field Experiments

We evaluated the proposed algorithm through proof-of-concept experiments using a custom UAV and a Clearpath Husky UGV (Figure 2.1). The UAV is capable of flying fully autonomous missions aided by GPS. The UAV is based on a DJI Flame Wheel F450 frame and uses the APM firmware running on a Pixhawk autopilot. The onboard computer (Nvidia Jetson TX1) interfaces with the autopilot using the mavros package [73] of the Robotic Operating System [94]. The Jetson TX1 runs the high-level mission algorithm and sends the list of waypoints to the autopilot.

The mission to be executed by the UAV and UGV is computed offline. The mission is decomposed into subpaths; each subpath consists of a single take-off, visiting one or more waypoints, and land sequence. While the UAV is executing the subpath, the UGV directly navigates to the landing site. Once at the final site on the subpath, the UAV lands on the UGV using a precision landing method that is implemented via IR-Lock (Figure 2.10a).

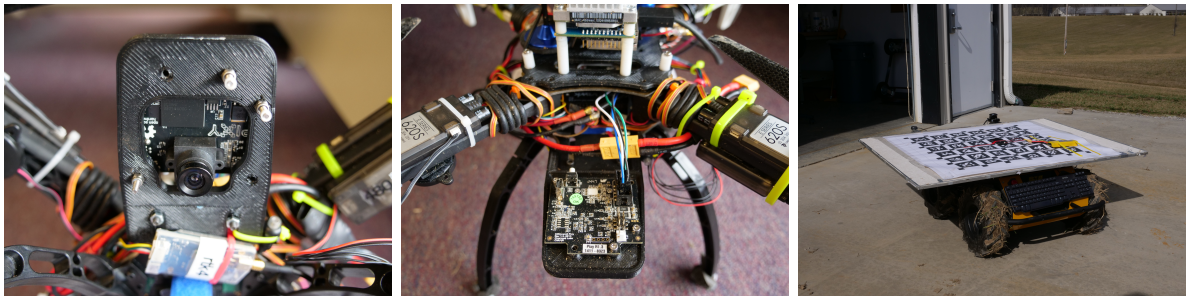
IR-Lock [53] is an off-the-shelf system integrated in APM firmware. It gives a landing accuracy of up to 30cm in ideal conditions. The experiments suggest that the UAV is capable of landing directly on top of the Husky (59cm x 73cm) in normal conditions. In windy conditions, a larger platform of size 91cm x 122cm (Figure 2.10c) was mounted on top of the Husky. The UAV was repeatedly successful in landing on the platform.

The IR-Lock sensor has 3.6mm lens giving horizontal and vertical viewing angles of approximately 60° and 35°. The MarkOne Beacon was used as the IR emitter, which has a detection range of 15m+ and a beam angle of 65°. During the experiments, the UAV flew at an altitude of 10m. Therefore, the UAV can detect the UGV within a 17m horizontal and 7m vertical footprint.

Once the UAV lands on the UGV, the UGV travel to the take-off site on the next subpath.

We manually "cold-swapped" the battery of the UAV everytime it landed on the UGV. This part of the system can be completely automated using existing solutions such as those presented in [10, 81].

This next take-off site could be a new site due to **Type II** edge or the same site due to **Type III** edge. The UAV then detects if it is at the next take-off point by comparing its current GPS position with the GPS coordinates of the next take-off point. This process loops until the UAV reaches its final site whereupon the UAV would autonomously land on the ground. A video of the system in action can be seen in the multimedia submission.



(a) IR-Lock camera mounted on the UAV, viewed from the bottom.

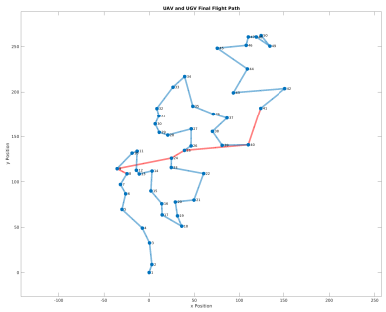
(b) IR-Lock camera mounted on the UAV, viewed from the top.

(c) IR-Beacon mounted on top of the landing platform on the UGV.

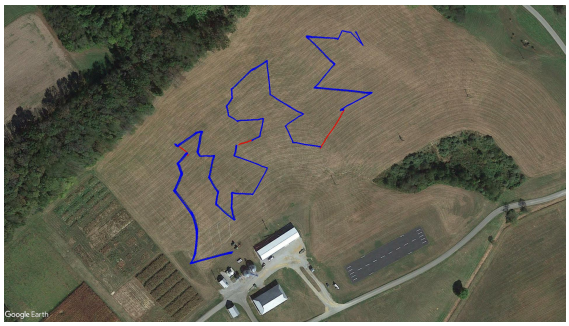
Figure 2.10: The IR-Lock system consists of a camera mounted on the UAV and an IR beacon mounted on the UGV. The system enables precision landing of the UAV on the UGV.

Figure 2.11a shows the 50 sites used for the proof-of-concept experiments of the SMCS problem. The sites were randomly generated in an area of 300×110 meters. The path was found using the GLNS solver with $m = 100$ battery levels, $t_{TO} = 4$, $t_L = 4$, $r = 0$, and $t_{UGV} = t_{UAV}$. For this particular path, a single UGV was sufficient even though it is slower than the UAV. Regardless, the implementation will make the UAV hover over the landing site until the UGV reaches it. In the experiments, the UGV always reached the landing site before the UAV. The outputs for the UAV tour and UGV tour are shown in Figure 2.11a. Figure 2.11b shows the output data from the Pixhawk flight controller.

The total time for the experiment was approximately 23 minutes. We conducted multiple sets of experiments with similar results (additional experiments can be found in [122]).



(a) Final MATLAB plot of UAV and UGV path.



(b) GPS trace of the UAV.

Figure 2.11: Experiments were performed at Kentland Farms in Blacksburg VA. The input was 50 sites in a 300×110 meters area with 100 battery levels. $t_{TO} = 4$, $t_L = 4$, $r = 0$, and $t_{UGV} = t_{UAV}$. For the experiment the UAV autonomously took off, traversed sites, and landed. The blue subpaths are UAV-only, the red subpaths are UAV+UGV edges. The UGV operated autonomously between landing and take-off sites.

2.6 Conclusion

In this chapter, we present an algorithm for routing a battery-limited UAV and a mobile recharging station to visit a set of sites of interest. We specifically consider scenarios where the UGV can recharge the UAV while being ferried to the next deployment location. We examine different combinations of input parameters that help dictate different methods to recharge in Section 2.4. We evaluate the assumptions through proof-of-concept experiments lasting about 30 minutes and rigorous simulations.

Chapter 3

2D Area Coverage with Symbiotic UAV+UGV System

In the previous chapter, we presented an algorithm for the 2D point coverage problem. In this chapter, we extend point coverage to 2D area coverage. The key challenge we address in this chapter is how to cover large environments with an energy-constrained UAV. Particularly, we are interested in scenarios where the coverage is expected to take longer than the battery capacity of the UAV.

One way of addressing energy constraints is by choosing a better platform. Multi-rotor UAVs have limited battery runtime, typically less than 30 minutes [40]. As a result, surveying large areas with a single vehicle may require frequent stops to recharge or replace batteries. Fixed-wing UAVs have longer runtimes, typically less than 90 minutes, but cannot take-off and land vertically or hover in place. The latter characteristic may be essential for visual coverage. Fixed-wing UAVs also have the steering constraints that limit their maneuverability. Hybrid UAVs seek to achieve the best of both worlds — higher maneuverability of a multi-rotor and longer endurance of a fixed-wing. Such hybrid UAVs are commercially available for coverage applications such as precision agriculture [6], environmental monitoring [116] reconnaissance [7] that involve visual coverage of large areas.

While fixed-wing and hybrid UAVs can mitigate some of the energy limitations, there may still be environments that are too large or need persistent monitoring beyond the runtime of

the UAV. To address this inherit limitation, we propose a solution that uses UGVs as mobile recharging stations similar to the previous chapter. In this chapter, we generalize this to area coverage using hybrid UAVs. Hybrid UAVs add the challenges of handling multiple modes of flight, Dubins' steering constraints when executing fixed-wing flight, and the different energy discharge rates dependent on the flight mode.

The input to the planner is a set of *boustrophedon cells* — rectangular regions whose width is equal to the footprint of the UAV's sensor. The boustrophedon cells can be obtained by decomposing regions that need to be covered [21, 74]. Figure 3.1 shows a motivating example of surveying crops in four fields. Here, each boustrophedon cells corresponds to a row of crops that need to be imaged by the UAV.

A boustrophedon cell can be covered by the UAV entering from either end and exiting from the other — the planner must find the optimal sequence in which to cover the boustrophedon cells as well as the corresponding entry and exit sites for each boustrophedon cells.

We consider scenarios where the UAV can land on the UGV and either recharge in-place or recharge while the UGV transports the UAV to the next take-off site. We present an algorithm that plans a tour for the UAV and a path for the UGV, such that the UAV can cover an area in the minimum time while never running out of charge. This includes not only the flight time of the UAV but also the time it takes to recharge as well as the taking-off and landing times. The output tour given by the planner specifies not only the order in which to cover the boustrophedon cells but also the charging schedule and flight mode that the UAV will be in. In particular, the planner outputs where and how to recharge the battery, how much to recharge by, and whether to fly in multi-rotor flight mode or fixed-wing flight mode.

This problem is a generalization of the NP-hard TSP [13]. The solution is based on reducing the coverage problem to the GTSP [85]. Specifically, we present an algorithm that is

guaranteed to find the optimal coverage tour for the UAV, as long the GTSP solver produces optimal tours. While no polynomial-time optimal algorithms are believed to exist for NP-hard problems, there are solvers that find optimal solutions to many instances in reasonable amounts of time in practice. One such solver, Generalized Large Neighborhood Search (GLNS) [102], that finds optimal solutions for GTSP instances is used. We empirically evaluate the performance of the algorithm using the GLNS solver.

This chapter, extends the method in the previous chapter for covering a set of points to covering a set of boustrophedon cells. We could use the previous method to cover boustrophedon cells by discretizing the whole environment and creating a set of points that need to be covered by the UAV. However, this becomes computationally expensive for larger areas and scales poorly. We also account for hybrid UAVs in this chapter. This chapter focuses on the task of a single hybrid UAV handling multiple modes of flight, Dubins steering constraints when executing fixed-wing flight, and the different energy discharge rates dependent on the flight mode.

The contributions of this chapter are as follows: we introduce a new area coverage problem formulation for a hybrid UAV using a UGV as a mobile charging platform; we design a coverage method that uses eighteen uniqe edges for UAV and UGV traversal of boustrophedon cells; and conduct an extensive evaluation of the method presented in this chapter.

3.1 Related Work

Environmental coverage is a well-studied problem in robotics [20, 41]. Choset [20] provides a survey of coverage using robots, including heuristic, approximate, partial-approximate, and exact cellular decomposition algorithms. The variant most closely related to the one we study is that of decomposing a known environment into various cells and then finding a route

to sweep (i.e., cover) each cell [17, 19, 44, 52, 120]. In this work, we assume that the first step (cell decomposition) has been solved and focus on the problem of routing the UAV to cover the cells in minimal time, while keeping track of the battery level and flight mode. We introduce a novel means of environmental coverage using a hybrid UAV, which can leverage aspects of a multi-rotor UAV, such as vertical take-off and landing, and fixed-wing UAV, such as long flight times. Specifically, we take as input a boustrophedon cell decomposition which can be found using techniques given by [74].

A number of algorithms have also been developed for the second step, i.e., routing to cover all cells, under various constraints. In particular, Karapetyan et al. presented a multi-robot coverage algorithm for boustrophedon cell decomposition for point robots [56]. Yu et al. presented a coverage algorithm for a single robot with Dubins steering constraints [126]. Lewis et al. later proved that the problem is NP-hard, and showed how to reduce the graph to obtain practical solutions [66]. Bochkarev and Smith present a method for decomposing an environment to minimize the number of turns needed to cover an area, but do not consider energy-limited robots and consequently, do not need to keep track of the battery level of the robot [17]. Most recently, Karapetyan et al. presented two techniques to cover a collection of cells with multiple Dubins vehicles [57].

The underlying ideas in the aforementioned works are similar — reduce the problem of covering all cells to a variant of the TSP, solve the TSP, and convert the resulting solution back to a tour for the robots. The approach extends these ideas for the case of a robot with limited energy capacity which can be recharged along the way and a robot with multiple flight modes, taking advantage of both multi-rotor and fixed-wing modalities. This requires me to keep track of energy level of the robot along the tour and the flight modes, which further complicates the problem. Nevertheless, we show that by reducing it to GTSP, we can obtain optimal solutions in reasonable amounts of time.

Many other works also analyze the coverage problem using fixed-wing UAVs, but do not have the advantages of a hybrid system. Paull et al. considered area coverage using onboard sensors for a fixed-wing UAV [89]. However the authors take an online approach, leading to solutions that are sub-optimal for environmental coverage. Xu et al. present an algorithm for optimal terrain coverage using fixed-wing UAV [118]. Their method considers minimizing overlapping areas of coverage, the method in this work does not allow for any overlapping coverage. Also experiments that utilize charging stations are provided as well as an implementation of a hybrid system. Coombes et al. use a fixed-wing UAV for survey coverage path planning in windy environments [25]. This approach studies the effects of wind on a fixed-wing system and proposes algorithms that solve for paths that take into account the wind patterns of the environment.

There have been algorithms for assigning and routing with one or more stationary recharging stations [8, 60, 69]. Kim et al. present a Mixed Integer Linear Programming approach for assigning UAVs to stationary recharging stations while taking into account the task objective [60]. Liu and Michael presented a method for assigning UAVs to UGVs acting as recharging stations [69]. Gramajo and Shankar [45] studied energy-constrained UAVs for search and coverage. That approach maximizes the area covered based on the amount of energy, but does not allow for recharging.

In previous works [111, 121, 123], the problem of visiting a set of boustrophedon cells (rectangles with the width of the field-of-view (FOV) of the sensor in a 2D plane) using an energy-limited UAV with only one mode of flight was studied. In [111], it was shown how to maximize the number of sites visited in a single charge when the UAV is allowed to land on the UGV and let the UGV transport it to the next take-off site, without recharging the UAV, and without the UAV expending energy. In [121], this is extended to also allow for the UGV to recharge the UAV either while stationary or while being transported to the next

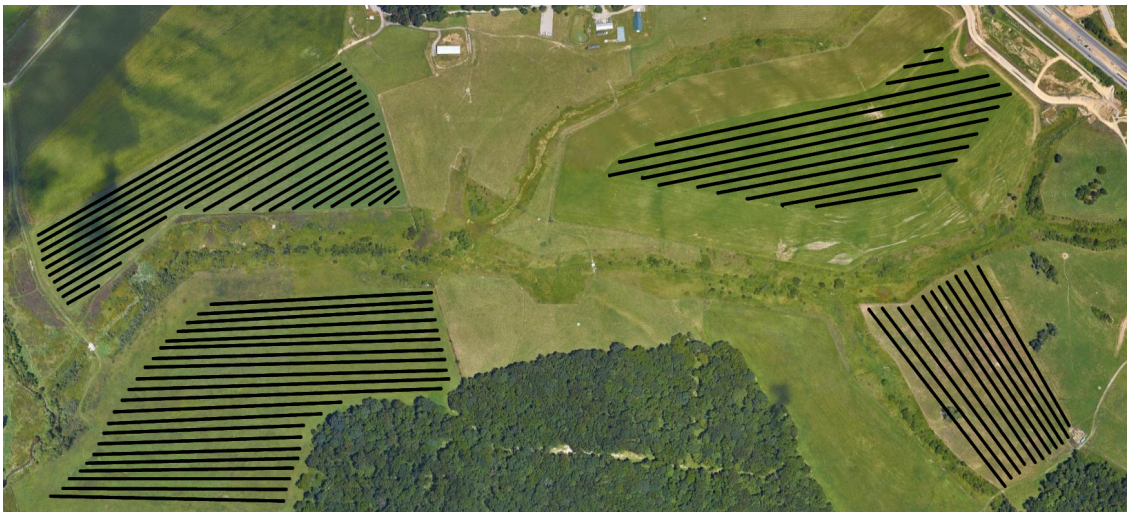


Figure 3.1: The problem of covering a set of boustrophedon cells using a UAV which has limited battery capacity is studied. In a precision agriculture scenario, a boustrophedon cell may correspond to a row of crops that must be monitored using a downward-facing camera on the UAV.

deployment site. In [123], the work is further extended from [121] to conduct coverage of boustrophedon cells. This chapter extends the prior chapter from merely having a single mode of flight to having multiple modes of flight. As a result, the planner must decide not only the order in which the boustrophedon cells should be visited but also the directions in which to cover them and what is the optimal flight mode to use.

3.2 Problem Formulation

The input to the algorithm proposed is a set of n boustrophedon cells that need to be covered by the UAV. We do not split the areas into boustrophedon cells, but instead rely on previous work [66] that optimize how areas are split into boustrophedon cells for coverage. A boustrophedon cell is a rectangular strip whose width is equal to the diameter of the FOV of the sensor onboard the robot. An example is shown in Figure 3.2 and a larger example is shown in Figure 3.1.

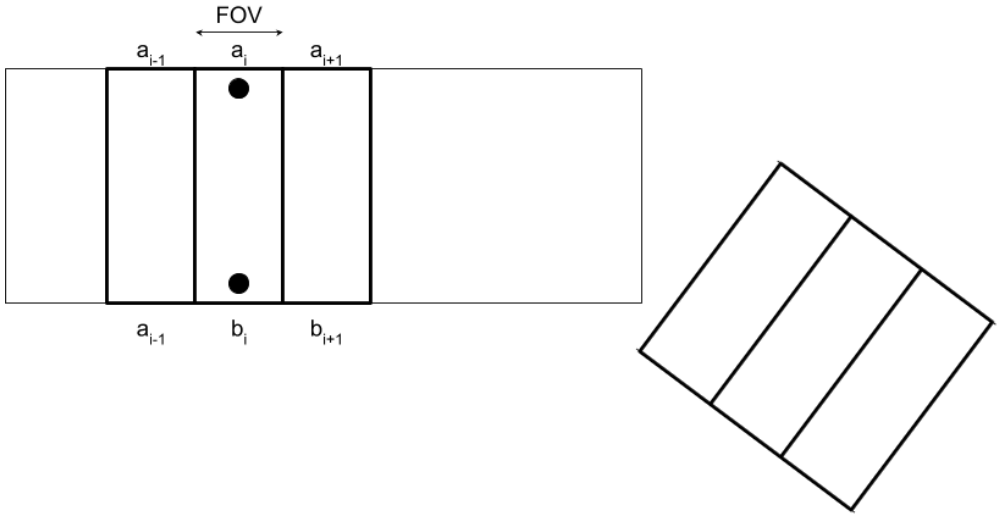


Figure 3.2: Example boustrophedon cells. Each boustrophedon cell is a rectangle whose width is equal to the footprint of the UAV's sensor. A boustrophedon cell i is characterized by two sites, a_i and b_i, on either end. The algorithm must choose which one acts as the entry site and how to traverse the boustrophedon cell.

Each boustrophedon cell is described by two sites, a_i and b_i, where i is the index of the boustrophedon cell. These sites are placed at two ends of the rectangular strip. A boustrophedon cell is said to be *covered* if the UAV travels in a straight line from a_i to b_i or from b_i to a_i. The UAV can enter a boustrophedon cell from either site, a_i or b_i. However, once the UAV has entered a boustrophedon cell it is required to cover the entire boustrophedon cell and exit from the other site. The coverage algorithm must choose one of the sites as the entry site as well as how to traverse the boustrophedon cell. We slightly generalize the traditional notion of a boustrophedon cell by allowing them to be oriented in different directions. That is, we do not require the boustrophedon cells to be parallel to each other.

We make the following assumptions:

- the UAV has an initial battery charge of 100%;
- the UAV flies at a fixed-altitude plane when covering a boustrophedon cell;

- the battery discharge at a unit rage (one unit per unit distance traveled);
- the UGV has unlimited fuel/battery capacity;
- boustrophedon cells do not overlap, however, paths may intersect while the UAV is transitioning;
- the UAV ignores transition costs between multi-rotor and fixed-wing mode;
- the UAV is fixed to one flight mode while covering a boustrophedon cell;
- edges can only conduct recharging if they correspond with sites the UGV can visit.

Note that the UGV can be a large vehicle such as the Clearpath Moose [4] which can run for 6 hours. This is significantly longer than the UAV mission time and therefore, it can be assumed that the UGV will not run out of charge during the mission. Even if the mission is longer than 6 hours, a quick battery swap for the UGV can double the run time. also note that the multi-rotor mode for the UAV is not required to be slower than the fixed-wing mode.

We use t_{TO} and t_L to represent the time taken by the UAV to take-off from the UGV to reach the fixed-altitude plane and land from this plane onto the UGV, respectively. $D_{\max}$ represents the total distance a UAV can travel with 100% battery capacity in multi-rotor mode.[1] We discretize the battery capacity into C levels. r represents the rate the battery gets recharged per unit time. $fRatio$ represents the ratio of multi-rotor to fixed-wing battery consumption. This is above 1 if the fixed-wing consumes less battery than the multi-rotor over a fixed distance. We define the turn radius of the fixed-wing as TR. When the UAV is in fixed-wing mode it will obey the constraints of a Dubins vehicle. This means that when

[1]Strictly speaking, we maintain a reserve battery capacity to take-off from ground to reach the fixed altitude plane and land from the fixed-altitude plane on the ground. In this work, when we refer to 100% battery capacity, it excludes this reserve battery for taking-off and landing.

the UAV is flying from the exit site of one boustrophedon cell to the entry site of another boustrophedon cell it will have to abide by the turn radius and therefore the cost to travel is dependent on the turn radius.

Let $\gamma(i) \in \{a_i, b_i\}$ denote the site chosen by a coverage algorithm to be the entry site of the i^{th} boustrophedon cell in the order in which they are to be visited. Correspondingly, $\overline{\gamma}(i)$ denotes the site chosen to be the exit site of the boustrophedon cell, i.e., $\overline{\gamma}(i) = \{a_i, b_i\} \backslash \gamma(i)$. $\sigma(j)$ denotes the order in which the boustrophedon cells are to be visited. That is, $\sigma(j) \in \{1, \ldots, n\}$ gives the j^{th} boustrophedon cell that is visited.

We use γ_i, $\overline{\gamma}_i$, and γ_{i+1} to denote $\gamma(\sigma(i))$, $\overline{\gamma}(\sigma(i))$, and $\gamma(\sigma(i+1))$, respectively. We denote by $t_G(\overline{\gamma}_j, \gamma_{j+1})$ the time it takes for the UGV to travel along the ground from the exit site of j^{th} boustrophedon cell to the entry site of the next visited boustrophedon cell. We use $t_M(\overline{\gamma}_j, \gamma_{j+1})$ to represent the time it takes for the UAV to travel in multi-rotor mode from the exit site of the j^{th} boustrophedon cell to the entry site of the next boustrophedon cell visited. Similarly, $t_M(\gamma_j, \overline{\gamma}_j)$ gives the time taken by the UAV in multi-rotor mode to cover the j^{th} boustrophedon cell. We use $t_F(\overline{\gamma}_j, \gamma_{j+1})$ to represent the time it takes for the UAV to travel in fixed-wing mode from the exit site of the j^{th} boustrophedon cell to the entry site of the next boustrophedon cell visited. Similarly, $t_F(\gamma_j, \overline{\gamma}_j)$ gives the time taken by the UAV in fixed-wing mode to cover the j^{th} boustrophedon cell. When calculating these times, we take into consideration the constraints on the UAV's motion. In particular, when in fixed-wing mode, we take into account the longer flight distance due to the turn radius constraint.

Suppose π is a path that visits every boustrophedon cell in the order given by σ and with entry and exit sites given by γ. The cost of the path depends on how the UAV travels between consecutive boustrophedon cells. Consider traveling from γ_j to $\overline{\gamma}_j$ and then on to γ_{j+1} along π. We have the following components for this part of the path:

- The UAV must fly from γ_j to $\overline{\gamma}_j$. The time taken is given by $t_M(\gamma_j, \overline{\gamma}_j)$ or $t_F(\gamma_j, \overline{\gamma}_j)$. Let $I_1(\gamma_j, \overline{\gamma}_j)$ be an indicator function that denotes whether the UAV chooses to fly in multi-rotor or fixed-wing mode.

- It can then choose to land on the UGV at $\overline{\gamma}_j$, recharge in-place, and take-off to reach the fixed-altitude plane at $\overline{\gamma}_j$. Let $I_2(\overline{\gamma}_j)$ be an indicator function that denotes whether the UAV chooses to do this or not.

- It can then choose to either fly from $\overline{\gamma}_j$ to γ_{j+1} or land on the UGV at $\overline{\gamma}_j$, recharge while being carried by the UGV to the next site, then take-off at γ_{j+1} to reach the fixed-altitude plane. Let $I_3(\overline{\gamma}_j)$ be an indicator function denoting whether the UAV travels with the UGV or not. Note that if UAV chooses flight then the indicator function $I_1(\overline{\gamma}_j, \gamma_{j+1})$ is also used to denote true for multi-rotor or false for fixed-wing flight.

- It can then choose to land on the UGV at γ_{j+1}, recharge in-place, and take-off to reach the fixed-altitude plane at γ_{j+1}. Let $I_2(\gamma_{j+1})$ be an indicator function that denotes whether the UAV chooses to do this or not.

Based on these choices, the cost of traveling from γ_j to γ_{j+1} is given by:

$$\begin{aligned} T(j, j+1) &= I_1(\gamma_j, \overline{\gamma}_j) t_M(\gamma_j, \overline{\gamma}_j) + (1 - I_1(\gamma_j, \overline{\gamma}_j)) t_F(\gamma_j, \overline{\gamma}_j) + I_2(\overline{\gamma}_j)(t_L + r \cdot b(\overline{\gamma}_j, \overline{\gamma}_j) + t_{TO}) \\ &+ (1 - I_3(\overline{\gamma}_j)) I_1(\overline{\gamma}_j, \gamma_{j+1}) t_M(\overline{\gamma}_j, \gamma_{j+1}) + (1 - I_3(\overline{\gamma}_j))(1 - I_1(\overline{\gamma}_j, \gamma_{j+1})) t_F(\overline{\gamma}_j, \gamma_{j+1}) \\ &+ I_3(\overline{\gamma}_j)(\max\{t_G(\overline{\gamma}_j, \gamma_{j+1}), r \cdot b(\overline{\gamma}_j, \gamma_{j+1})\} + t_L + t_{TO}) + I_2(\gamma_{j+1})(t_L + r \cdot b(\gamma_{j+1}, \gamma_{j+1}) + t_{TO}). \end{aligned} \tag{3.1}$$

Here, $b(\cdot)$ is a function which gives the amount by which the battery should be recharged between two sites.

Therefore, we can define the cost of the path π as:

$$T(\pi) = \sum_{j=1}^{n-1} T(\gamma_j, \gamma_{j+1}) + \min\{t_M(\gamma_n, \overline{\gamma}_n), t_F(\gamma_n, \overline{\gamma}_n)\} \tag{3.2}$$

At the end of π we take the minimum of t_M and t_F because at the last site the UAV will not need to conduct any type of charging and therefore will only need to cover the site. The problem is now ready to be defined.

Problem 4 (Multiple Polygon Coverage)**.** Given a set of boustrophedon cells to be covered, find a path π^*, which contains $\sigma(\cdot)$, $\gamma(\cdot)$, $I_2(\cdot)$, $I_3(\cdot)$, $I_1(\cdot)$ and $b(\cdot)$, for the UAV that visits and covers all of the boustrophedon cells, while minimizing the cost (Equation 3.2), and ensuring that the UAV does not run out of battery capacity. The path π^* must specify the order in which to visit the boustrophedon cells, $\sigma(\cdot)$, the entry site for each boustrophedon cell, $\gamma(\cdot)$, the recharging indicator functions, $I_2(\cdot)$, $I_3(\cdot)$ and $I_1(\cdot)$, the amount of recharging at a site, $b(\cdot)$, and when to change flight modes during coverage.

Note that finding a path for the UAV necessitates finding a path for the UGV that supports the UAV recharging schedule.

3.3 GTSP-based Algorithm

We solve the polygon coverage problem by reducing it to GTSP. In this section, we describe in detail the reduction to GTSP. The input to GTSP is a directed weighted graph where the vertices are partitioned into clusters. The objective is to find a minimum cost tour that visits exactly one vertex in each cluster. If all the clusters contain exactly one vertex, then GTSP trivially reduces to TSP. We show how to create the clusters, the edges between the clusters and then show how to convert the solution for GTSP into tours for the UAV and

the UGV. Algorithm 1 summarizes the process in pseudocode.

Algorithm 1 Overview of the approach.

1: Initialize graph G
2: **for** $i = 1, 2, \ldots, n$ **do**
3: **for** $k = 1, 2, \ldots, C$ **do**
4: Add $v_{a_i}^k$ and $v_{b_i}^k$ to G
5: **end for**
6: Create cluster i containing $\{v_{a_i}^1, v_{b_i}^1, \ldots v_{a_i}^C, v_{b_i}^C\}$
7: **end for**
8: $T \leftarrow \{$M-M, F-F, M-F, F-M, M-DTU, F-DTU, M-MDU, F-FDU, M-FDU, F-MDU, M-DUM, F-DUF, M-DUF, F-DUM$\}$
9: **for** each vertex $v \in G$ **do**
10: **for** each vertex $w \in G$ **do**
11: **if** $v \neq w$ **then**
12: $d \leftarrow \infty$
13: **for** each edge type $t \in T$ **do**
14: $c \leftarrow$ cost of edge type t from v to w
15: $d \leftarrow \min\{d, c\}$
16: **end for**
17: Create edge $v \rightarrow w$ with weight d in G
18: **end if**
19: **end for**
20: **end for**
21: Solve GTSP with graph G and clusters $1, \ldots, n$.
22: Extract UAV tour from GTSP solution.
23: Extract UGV tour from GTSP solution.

A Vertices and Clusters

We discretize the battery's charge states into C levels. We create C vertices, one corresponding to each discretized battery level, for each site a_i and b_i corresponding to boustrophedon cell i, shown in line 4 in Algorithm 1. The vertex represents coverage of a boustrophedon cell and is denoted by $v_{a_i}^k$ or $v_{b_i}^k$, where k corresponds to the discretized battery level and a_i or b_i corresponds to the entry site. Thus there are $2nC$ total vertices in the graph. We create

one cluster per boustrophedon cell, shown in line 6 in Algorithm 1. This cluster contains $2C$ vertices, C of them corresponding to a_i and C of them corresponding to b_i.

B Edges

We create an edge between every pair of vertices that do not belong to the same cluster (i.e., do not belong to the same boustrophedon cell). Recall that a vertex corresponds to the entry site for the corresponding boustrophedon cell. Therefore, an edge between two vertices represents the UAV starting at the entry site of the first boustrophedon cell and ending at the entry site of the next boustrophedon cell. This includes two travel legs: coverage of the first boustrophedon cell and then traveling from the exit site of the first boustrophedon cell towards the second boustrophedon cell. Recall that the UAV must always fly the first leg, either in multi-rotor or fixed-wing mode without switching between mode; however, the second leg can be a combination of recharging, flying, and/or recharging while traveling on the UGV. The UAV can only switch between flight modes when at a vertex, not along an edge.

When creating edges the UAV has to respect the Dubins constraints of a fixed-wing aircraft. The Dubins constraint of a fixed-wing UAV means that the UAV has a minimum turn radius when executing turns. The Dubins constraints will only affect the costs of the UAV when in the fixed-wing mode. With this in mind, the edges that utilize the UAV's fixed-wing mode have to take into account the cost of the Dubins constraints when executing turns. Since the starting orientation of the UAV is known at a site and the ending orientation of the UAV at the next site, the minimum cost path to travel for the UAV can be calculated [71].

Equation 3.1 gives the cost of traveling between two entry sites of different boustrophedon cells. The actual cost depends on the five binary indicator variables: $I_1(\gamma_j, \overline{\gamma}_j)$, $I_2(\gamma_j)$, $I_3(\overline{\gamma_j})$,

$I_1(\overline{\gamma}_j, \gamma_{j+1})$, and $I_2(\gamma_{j+1})$. This gives a total of 2^5 possible travel options. However, fourteen of these thirty-two options are redundant. Specifically, if the UAV chooses to recharge while traveling on the UGV, then also recharging on either end of this leg is redundant and, in fact, more time-consuming since it will have to take-off and land more than once. Formally, if $I_3(\overline{\gamma_j}) = 1$, then the optimal algorithm will never set $I_2(\gamma_j) = 1$ nor $I_2(\gamma_{j+1}) = 1$. Also if $I_3(\overline{\gamma_j}) = 1$ then what $I_1(\gamma_j, \overline{\gamma}_j)$ and $I_1(\overline{\gamma}_j, \gamma_{j+1})$ is equal to does not matter and those possibilities can be eliminated as well. We leave two states where $I_1(\gamma_j, \overline{\gamma}_j) = 1$ and $I_1(\gamma_j, \overline{\gamma}_j) = 0$ to handle the state in which the UAV will use the UGV as a transport. Therefore, of these 2^5 possibilities, fourteen can be eliminated, leaving a total of eighteen possibilities. These are shown in Figures 3.3, 3.4, 3.5, 3.6, 3.7. Note that since the UAV starts with 100% battery capacity, it will never recharge at the first entry site.

The eighteen edge combinations are donated using the notation: M = Multi-rotor, F = Fixed-wing, D = Down/Land, U = Up/Take-off, and T = Transit. The first leg is always the UAV flying to cover the boustrophedon cell. The actual edge costs are described in the remainder of this section.

In the following, it is shown how to compute the edge cost between vertices $v_{a_i}^{k_i}$ and $v_{a_j}^{k_j}$. As a reminder a_i, b_i, a_j, and b_j represent potential entry/exit sites for the i^{th} and j^{th} boustrophedon cell. This is line 14 in Algorithm 1. In the algorithm we use v and w to represent the possible $v_{a_i}^{k_i}$, $v_{a_j}^{k_j}$, $v_{b_i}^{k_i}$, and $v_{b_j}^{k_j}$. k_i' denotes the battery at $v_{b_i}^{k_i'}$ if going from $v_{a_i}^{k_i}$ and $v_{a_j}^{k_j}$. Note that there will also be edges between $v_{b_i}^{k_i}$ and $v_{a_j}^{k_j}$, $v_{a_i}^{k_i}$ and $v_{b_j}^{k_j}$, and $v_{b_i}^{k_i}$ and $v_{b_j}^{k_j}$. The costs for these edges can be obtained using the same formula just by swapping a with b and vice-versa.

The cost of the **M-M**, **F-F**, **M-F**, and **F-M** type of edges, Figure 3.3, between $v_{a_i}^{k_i}$ and $v_{a_j}^{k_j}$ is ∞ if the energy required to go from a_i to b_i and then to a_j is more than $k_j - k_i$. Else, the

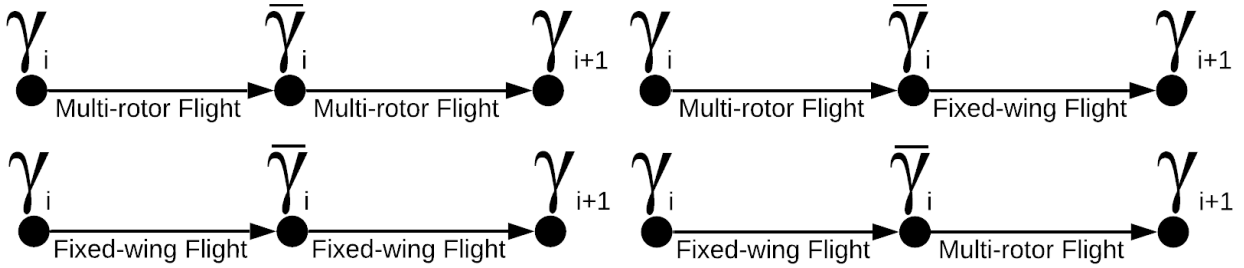


Figure 3.3: M-M, F-F, M-F, F-M.

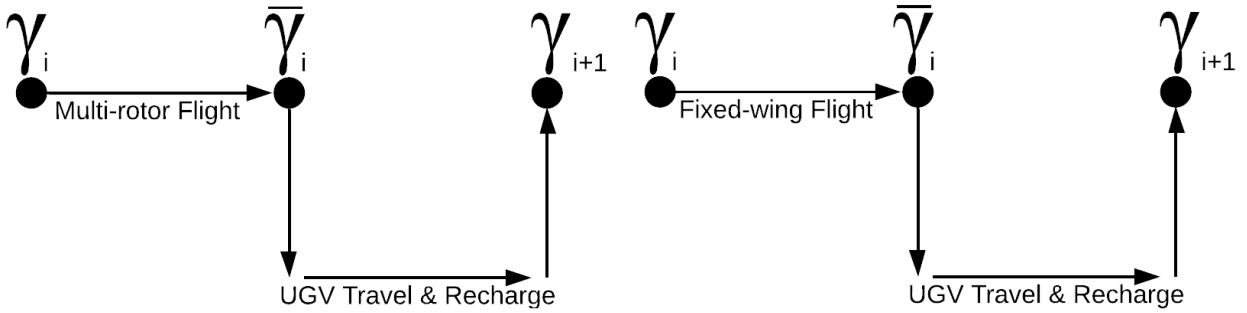


Figure 3.4: M-DTU, F-DTU.

edge cost is given by:

$$T_{\text{M-M}}(v_{a_i}^{k_i}, v_{a_j}^{k_j}) = t_M(v_{a_i}^{k_i}, v_{b_i}^{k'_i}) + t_M(v_{b_i}^{k'_i}, v_{a_j}^{k_j}), \tag{3.3}$$

$$T_{\text{F-F}}(v_{a_i}^{k_i}, v_{a_j}^{k_j}) = t_F(v_{a_i}^{k_i}, v_{b_i}^{k'_i}) + t_F(v_{b_i}^{k'_i}, v_{a_j}^{k_j}). \tag{3.4}$$

$$T_{\text{M-F}}(v_{a_i}^{k_i}, v_{a_j}^{k_j}) = t_M(v_{a_i}^{k_i}, v_{b_i}^{k'_i}) + t_F(v_{b_i}^{k'_i}, v_{a_j}^{k_j}), \tag{3.5}$$

$$T_{\text{F-M}}(v_{a_i}^{k_i}, v_{a_j}^{k_j}) = t_F(v_{a_i}^{k_i}, v_{b_i}^{k'_i}) + t_M(v_{b_i}^{k'_i}, v_{a_j}^{k_j}), \tag{3.6}$$

The cost of the **M-DTU** and **F-DTU** type of edges, Figure 3.4, between $v_{a_i}^{k_i}$ and $v_{a_j}^{k_j}$ is equal to ∞ if the energy required to go from a_i to b_i is more than $k'_i - k_i$. Else, the edge cost is

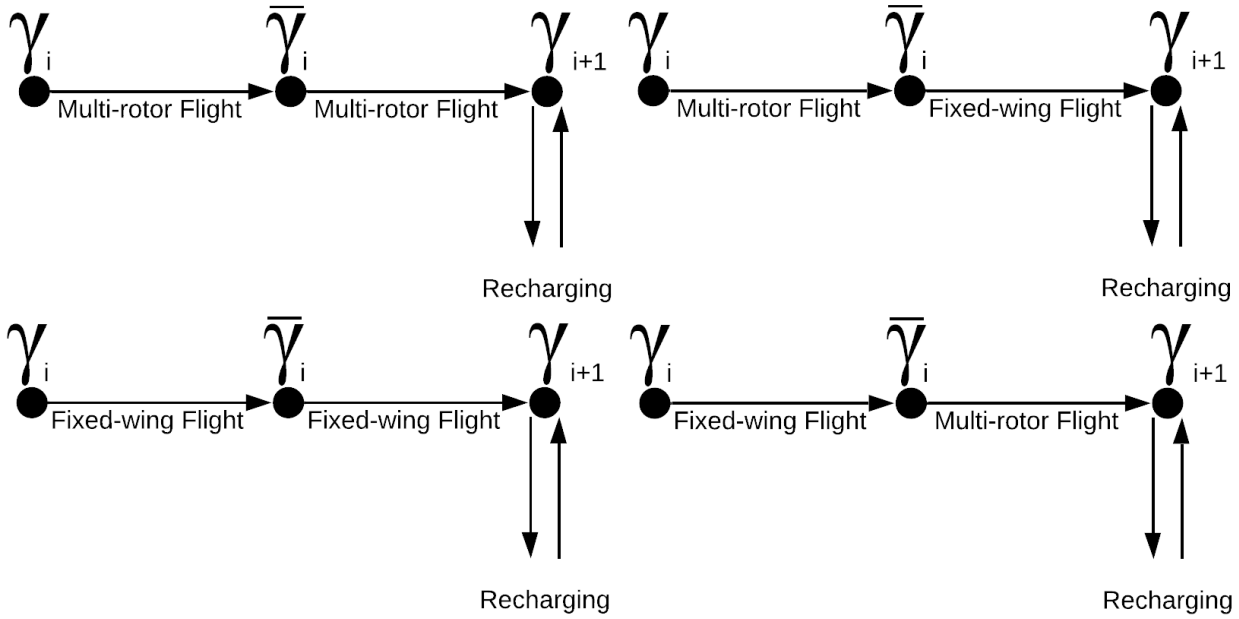


Figure 3.5: M-MDU, F-FDU, M-FDU, F-MDU.

given by:

$$T_{\text{M-DTU}}(v_{a_i}^{k_i}, v_{a_j}^{k_j}) = t_M(v_{a_i}^{k_i}, v_{b_i}^{k'_i}) + t_L + \max(t_G(v_{b_i}^{k'_i}, v_{a_j}^{k_j}), r \cdot e) + t_{TO}, \tag{3.7}$$

$$T_{\text{F-DTU}}(v_{a_i}^{k_i}, v_{a_j}^{k_j}) = t_F(v_{a_i}^{k_i}, v_{b_i}^{k'_i}) + t_L + \max(t_G(v_{b_i}^{k'_i}, v_{a_j}^{k_j}), r \cdot e) + t_{TO}, \tag{3.8}$$

where $e = \max\{0, k_j - (k'_i - \|b_i - a_j\|_2)\}$ gives the recharging amount.

The cost of the **M-MDU**, **F-FDU**, **M-FDU**, and **F-MDU** type of edges, Figure 3.5, between $v_{a_i}^{k_i}$ and $v_{a_j}^{k_j}$ is ∞ if the energy required to go from a_i to b_i and then to a_j is more than $k_j - k_i$. Else, the edge cost is:

$$T_{\text{M-MDU}}(v_{a_i}^{k_i}, v_{a_j}^{k_j}) = t_M(v_{a_i}^{k_i}, v_{b_i}^{k'_i}) + t_M(v_{b_i}^{k'_i}, v_{a_j}^{k_j}) + t_L + r \cdot e + t_{TO}, \tag{3.9}$$

$$T_{\text{F-FDU}}(v_{a_i}^{k_i}, v_{a_j}^{k_j}) = t_F(v_{a_i}^{k_i}, v_{b_i}^{k'_i}) + t_F(v_{b_i}^{k'_i}, v_{a_j}^{k_j}) + t_L + r \cdot e + t_{TO}, \tag{3.10}$$

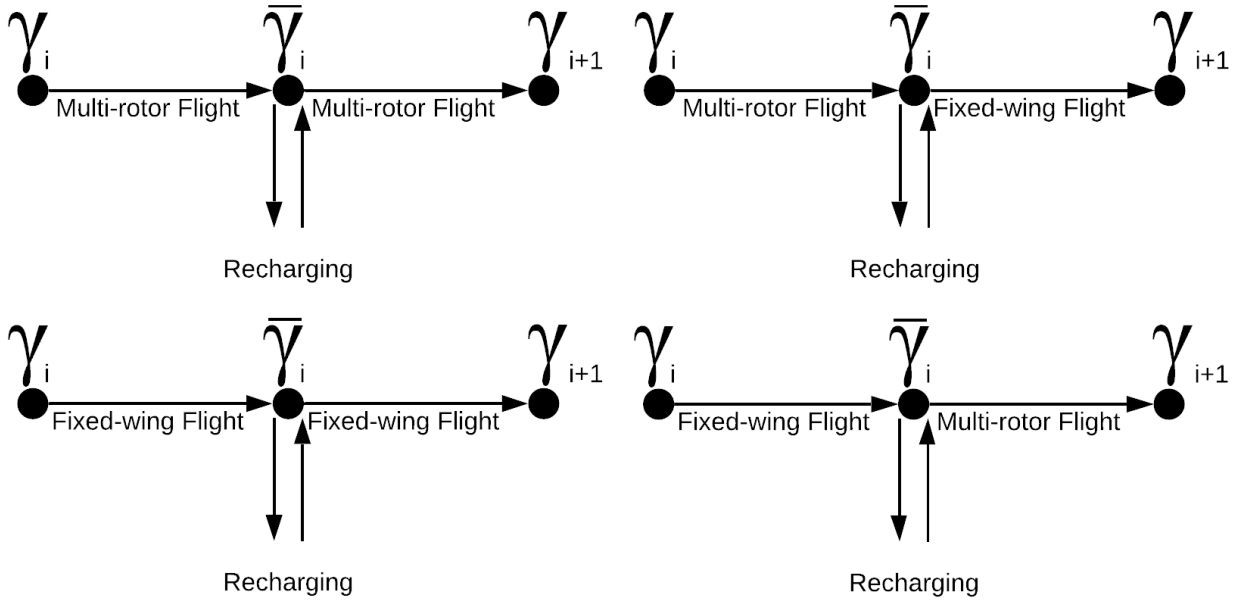


Figure 3.6: M-DUM, F-DUF, M-DUF, F-DUM.

$$T_{\text{M-FDU}}(v_{a_i}^{k_i}, v_{a_j}^{k_j}) = t_M(v_{a_i}^{k_i}, v_{b_i}^{k'_i}) + t_F(v_{b_i}^{k'_i}, v_{a_j}^{k_j}) + t_L + r \cdot e + t_{TO}, \tag{3.11}$$

$$T_{\text{F-MDU}}(v_{a_i}^{k_i}, v_{a_j}^{k_j}) = t_F(v_{a_i}^{k_i}, v_{b_i}^{k'_i}) + t_M(v_{b_i}^{k'_i}, v_{a_j}^{k_j}) + t_L + r \cdot e + t_{TO}, \tag{3.12}$$

where $e = \max\{0, k_j - (k_i - (\|a_i - b_i\|_2 + \|b_i - a_j\|_2))\}$ gives the recharging amount.

The cost of the **M-DUM**, **F-DUF**, **M-DUF**, and **F-DUM** type of edges, Figure 3.6, between $v_{a_i}^{k_i}$ and $v_{a_j}^{k_j}$ is equal to ∞ if the energy required to go from a_i to b_i is more than $k'_i - k_i$ and then b_i to a_j is more than $k_j - k'_i$. Else, the edge cost is given by:

$$T_{\text{M-DUM}}(v_{a_i}^{k_i}, v_{a_j}^{k_j}) = t_M(v_{a_i}^{k_i}, v_{b_i}^{k'_i}) + t_L + r \cdot e + t_{TO} + t_M(v_{b_i}^{k'_i}, v_{a_j}^{k_j}), \tag{3.13}$$

$$T_{\text{F-DUF}}(v_{a_i}^{k_i}, v_{a_j}^{k_j}) = t_F(v_{a_i}^{k_i}, v_{b_i}^{k'_i}) + t_L + r \cdot e + t_{TO} + t_F(v_{b_i}^{k'_i}, v_{a_j}^{k_j}), \tag{3.14}$$

$$T_{\text{M-DUF}}(v_{a_i}^{k_i}, v_{a_j}^{k_j}) = t_M(v_{a_i}^{k_i}, v_{b_i}^{k'_i}) + t_L + r \cdot e + t_{TO} + t_F(v_{b_i}^{k'_i}, v_{a_j}^{k_j}), \tag{3.15}$$

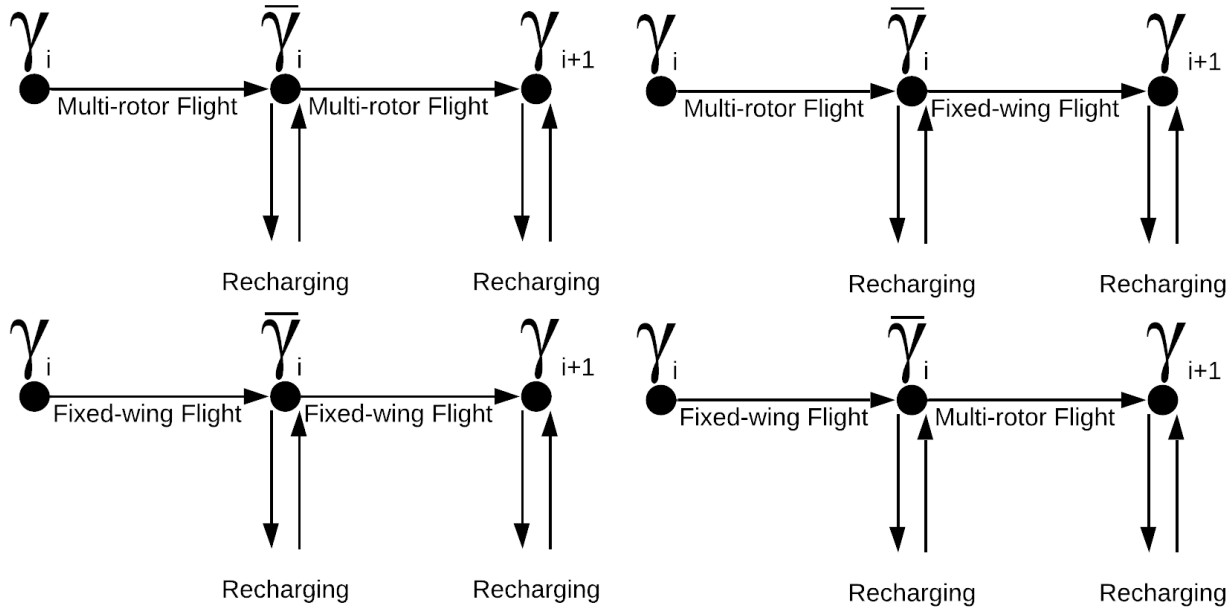


Figure 3.7: M-DUMDU, F-DUFDU, M-DUFDU, F-DUMDU.

$$T_{\text{F-DUM}}(v_{a_i}^{k_i}, v_{a_j}^{k_j}) = t_F(v_{a_i}^{k_i}, v_{b_i}^{k'_i}) + t_L + r \cdot e + t_{TO} + t_M(v_{b_i}^{k'_i}, v_{a_j}^{k_j}), \tag{3.16}$$

where $e = \max\{0, k'_i - (k_i - \|a_i - b_i\|_2)\}$ gives the recharging amount.

The cost of the M-DUMDU, F-DUFDU, M-DUFDU, and F-DUMDU type of edges, Figure 3.7, between $v_{a_i}^{k_i}$ and $v_{a_j}^{k_j}$ is equal to ∞ if the energy required to go from a_i to b_i is more than $k'_i - k_i$ and then b_i to a_j is more than $k_j - k'_i$. Else, the edge cost is given by:

$$T_{\text{M-DUMDU}}(v_{a_i}^{k_i}, v_{a_j}^{k_j}) = t_M(v_{a_i}^{k_i}, v_{b_i}^{k'_i}) + t_L + r \cdot e_1 + t_{TO} + t_M(v_{b_i}^{k'_i}, v_{a_j}^{k_j}) + t_L + r \cdot e_2 + t_{TO}, \tag{3.17}$$

$$T_{\text{F-DUFDU}}(v_{a_i}^{k_i}, v_{a_j}^{k_j}) = t_F(v_{a_i}^{k_i}, v_{b_i}^{k'_i}) + t_L + r \cdot e_1 + t_{TO} + t_F(v_{b_i}^{k'_i}, v_{a_j}^{k_j}) + t_L + r \cdot e_2 + t_{TO}, \tag{3.18}$$

$$T_{\text{M-DUFDU}}(v_{a_i}^{k_i}, v_{a_j}^{k_j}) = t_M(v_{a_i}^{k_i}, v_{b_i}^{k'_i}) + t_L + r \cdot e_1 + t_{TO} + t_F(v_{b_i}^{k'_i}, v_{a_j}^{k_j}) + t_L + r \cdot e_2 + t_{TO}, \tag{3.19}$$

$$T_{\text{F-DUMDU}}(v_{a_i}^{k_i}, v_{a_j}^{k_j}) = t_F(v_{a_i}^{k_i}, v_{b_i}^{k'_i}) + t_L + r \cdot e_1 + t_{TO} + t_M(v_{b_i}^{k'_i}, v_{a_j}^{k_j}) + t_L + r \cdot e_2 + t_{TO}, \tag{3.20}$$

where $e_1 = \max\{0, k_i' - (k_i - \|a_i - b_i\|_2)\}$ and $e_2 = \max\{0, k_j - (k_i' - \|b_i - a_j\|_2)\}$ gives the recharging amount for e_1 and e_2, respectively.

The actual edge cost between $v_{a_i}^{k_i}$ and $v_{a_j}^{k_j}$ is the minimum of all eighteen types. This is done in the for loop starting on line 13 in Algorithm 1. Specifically, the final edge cost is given by:

$$\begin{aligned} T(v_{a_i}^{k_i}, v_{a_j}^{k_j}) = \min\{&T_{\text{M-M}}, T_{\text{F-F}}, T_{\text{M-F}}, T_{\text{F-M}}, T_{\text{M-DTU}}, T_{\text{F-DTU}}, T_{\text{M-MDU}}, T_{\text{F-FDU}}, T_{\text{M-FDU}}, \\ &T_{\text{F-MDU}}, T_{\text{M-DUM}}, T_{\text{F-DUF}}, T_{\text{M-DUF}}, T_{\text{F-DUM}}, T_{\text{M-DUMDU}}, T_{\text{F-DUFDU}}, T_{\text{M-DUFDU}}, T_{\text{F-DUMDU}}\}. \end{aligned} \tag{3.21}$$

The algorithm also keeps track of which type of edge gives the final edge cost. This is used when converting the GTSP tour into a solution for the original problem.

C Solving GTSP

The GTSP instance is solved using the GLNS solver [102], in line 21 in Algorithm 1. GLNS uses a neighborhood search heuristic to find the optimal solution for the given GTSP instance. GLNS also allows for finding feasible solutions in lesser time, potentially at the expense of optimality.

Common alternatives for finding the optimal GTSP solution are Integer Programming or reducing GTSP to TSP [85] and then using a TSP solver such as Concorde [11]. In previous work [122], we showed that GLNS finds the optimal solution for a similar class of GTSP instances in times that are at least an order of magnitude faster than the other approaches. As a result, only GLNS is used for solving the GTSP instances in this chapter.

D Converting the GTSP solution to a Coverage Tour

The optimal tour obtained from the GTSP solver is a tour that visits exactly one vertex in each cluster. Recall that one cluster corresponds to one boustrophedon cell. The optimal tour will visit only one vertex within a cluster. The chosen vertex corresponds specifies the entry site for the boustrophedon cell as well as the corresponding battery level.

For example, if the edge between $v_{a_i}^{k_i}$ and $v_{b_j}^{k_j}$ is selected, then this implies the UAV will cover the i^{th} boustrophedon cell with a_i as the entry site and b_i as the exit site. Then, the UAV will travel from b_i to the entry site of the next boustrophedon cell which is chosen to be b_j. The actual mode of transportation between $v_{a_i}^{k_i}$ and $v_{b_j}^{k_j}$ depends on the type of the edge, denoted by the eighteen edges in Figures 3.3, 3.4, 3.5, 3.6, 3.7. Depending the type, the algorithm constructs the actual tour and recharging schedule for the UAV. See line 22 in Algorithm 1.

The UGV path can be determined based on the type of edges chosen by considering the edges in the order they appear in the optimal GTSP tour. For a **M-M**, **M-F**, **F-M**, **F-F** edge, the UGV is not required. For an **M-DUM**, **F-DUF**, **M-DUF**, **F-DUM** edge, the exit site of the first boustrophedon cell is added to the UGV tour. Similarly, for an **M-MDU**, **F-FDU**, **M-FDU**, **F-MDU**, the entry site of the second boustrophedon cell is added to the UGV tour. Finally, for **M-DUMDU**, **F-DUFDU**, **M-DUFDU**, **F-DUMDU**, **M-DTU**, and **F-DTU** edges, the exit site of the first boustrophedon cell and the entry site of the second one are both added to the UGV tour. This is done in line 23 in Algorithm 1.

E Performance Guarantees

If the GTSP solver finds the optimal tour, then the corresponding UAV tour is also the optimal solution to Problem 4 with the additional assumption that the UGV is as fast as

the UAV. If the UAV is faster than the UGV, then it is possible that the solution yields paths where the UAV reaches a landing site before the UGV. In such cases, one possibility is to have more than one UGV that can support the UAV tour. In the previous work [122], an Integer Programming solution is presented to minimizes the number of slower UGVs required to support the UAV tour. As a note, there is a chance that the GLNS solver finds a sub-optimal solution which depends on the settings used in GLNS that trade-off time and solution quality. The optimally guarantee is further explained in the paper by Smith and Imeson [102]. In Algorithm 1, all lines except line 21 run in $O(\max(n^2, nC))$ time. For line 21, a bound cannot be given on the time complexity since the number of iterations in GLNS [102] can be non-deterministic and unbounded. Nevertheless, we find that GLNS returns near optimal solutions in limited time. We present empirical results in the next section to discuss this further.

3.4 Simulations

In this section, we compare the results with an optimal solution as well as we present qualitative and quantitative results to evaluate the proposed algorithm. In particular, we analyze the effect of various parameters on the tour cost and the computational time of the algorithm. The experiments were run on an Ubuntu 16.04 computer with an Intel i7-8750H CPU running at 2.2GHz, with 6 physical cores, 32GB of RAM, and a GTX 1070 GPU.

A Mixed Integer Linear Programming (MILP) vs GLNS

For the rest of the paper we use GLNS to solve the GTSP instances. GLNS is the state-of-the-art GTSP solver [102], however it may not always yield the optimal solution. We compare

the solution obtained from GLNS with that obtained directly using a Mixed Integer Linear Programming (MILP) formulation.

We report results for all three modes of operation (slow, medium, and fast) for GLNS. We used Gurobi [46] to solve the MILP instance directly. We set the BarConvTol parameter to 0, which implies that Gurobi will terminate only after obtaining an optimal solution.

We create ten random instances of four boustrophedon cells with four battery levels. The results are shown in Table 3.1. We report the ratio between the final costs of GLNS/Gurobi for all three modes of GLNS, as well as the time taken of GLNS in all three modes and Gurobi. We observe that in the fast mode, GLNS is indeed faster than Gurobi but has a higher cost ratio. The medium mode takes comparable time to Gurobi and the slow mode is much slower. However, the medium mode and the slow mode has a similar cost ratio. Smith and Imeson [102] report that GLNS in the medium mode has an the average percentage gap from the best known solution of 6.97%. The medium mode is also the default mode, which we use for the rest of the simulations in this chapter. Gurobi was unable to solve instances larger than those with four boustrophedon cells and four battery levels with Gurobi. Specifically, Gurobi would crash before obtaining a solution running out of memory. We suspect this is due to the fact that the MILP formulation consists of a much larger number of decision variables than the GTSP formulation. However, as it will be shown shortly, GLNS is able to handle larger instances with the same computing hardware.

B Qualitative Examples

We use the boustrophedon cells given in Figure 3.1 as the input. There are a total of 66 boustrophedon cells. The solution is shown in Figure 3.8. The boustrophedon cells are marked with rectangles. The input parameters were set to: $t_{TO} = 5$, $t_L = 45$, $r = 2$,

	average	min	max	std
Cost Ratio Slow (GLNS/Gurobi)	1.23	1.14	1.39	0.07
Cost Ratio Medium (GLNS/Gurobi)	1.23	1.14	1.39	0.07
Cost Ratio Fast (GLNS/Gurobi)	1.36	1.14	1.67	0.14
GLNS Solve Time Slow (seconds)	0.10	0.09	0.12	0.01
GLNS Solve Time Medium (seconds)	0.03	0.02	0.04	0.01
GLNS Solve Time Fast (seconds)	0.01	0.01	0.01	0.00
Gurobi Solve Time (seconds)	0.02	0.02	0.02	0.00

Table 3.1: Summary of Gurobi solver vs GLNS solver in all three modes.

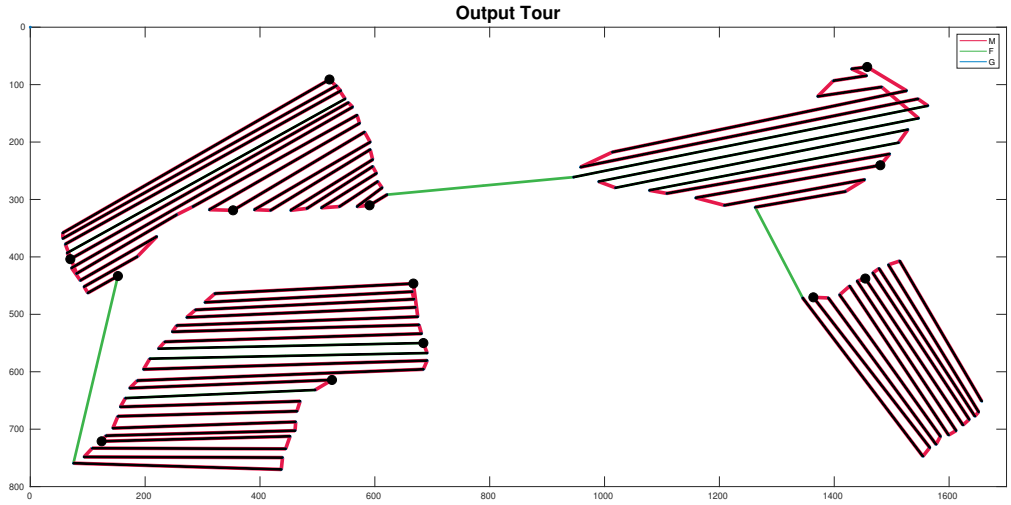


Figure 3.8: Solution obtained by the algorithm showing how the UAV will traverse the input instance given in Figure 3.1. The bolded black sites represent stationary recharging.

$D_{\max} = 1800$, $C = 20$, $fRatio = 3$, and $TR = 3$. The UGV speed was set to be 20% as that of the UAV.

We compare the results of the algorithm proposed with a naive baseline. The baseline approach visits each boustrophedon cell in multi-rotor mode in the same order in which they appear along the boundary of the polygon. The UAV lands to recharge on the UGV only when covering the next boustrophedon cell would deplete it of the remaining energy. Once on the UGV, the UAV recharges to maximum capacity. The baseline approach produces a

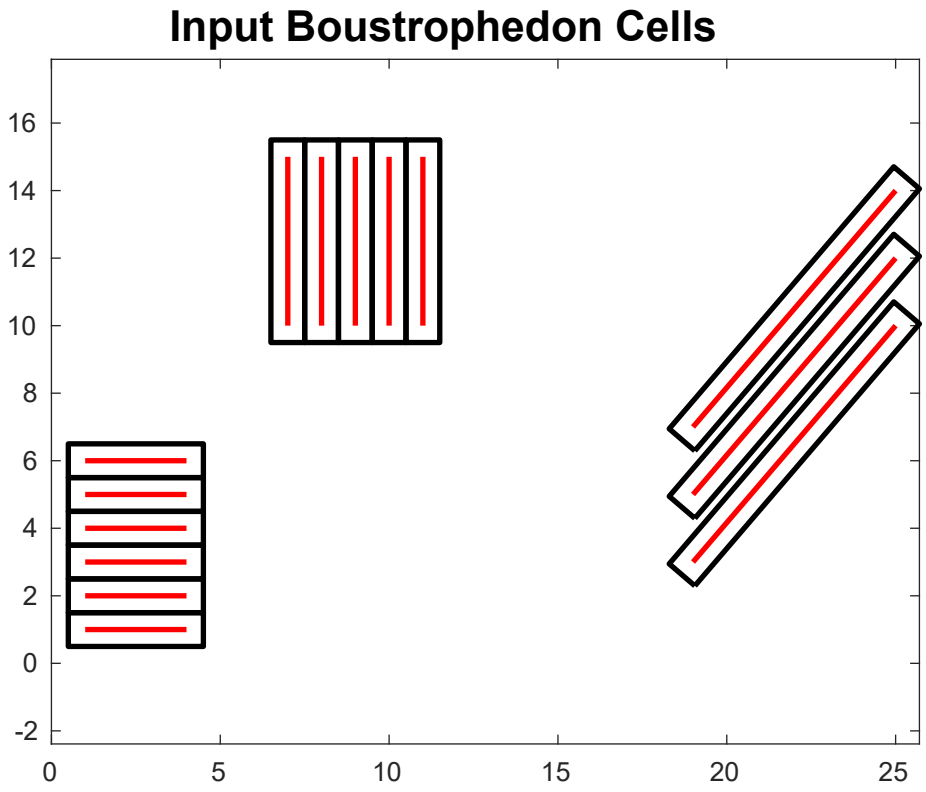


Figure 3.9: Input

Figure	Input Parameters Unique Quality	Output Change	Total Path Cost (meters)
Figure 3.10a	Enough battery capacity	No UGV necessary	99
Figure 3.10b	Enough battery capacity	No UGV necessary	207
Figure 3.10c	UGV is slower and $D_{\max}$ is small	UAV charges in place	253
Figure 3.10d	UGV is slower and $D_{\max}$ is small	UAV charges in place	251
Figure 3.10e	UGV is slower and $D_{\max}$ is small	UAV charges in place	419
Figure 3.10f	UGV is not slower than UAV	UAV uses UGV as mobile recharging station	242

Table 3.2: Summary of Figure 3.10.

tour which requires 29564 seconds for completion (as given by Equation 3.2) where as the proposed algorithm produces a tour which requires 25595 seconds.

Figures 3.10a– 3.10f presents additional qualitative examples that show the effect of changing multiple input parameters on the solution for the input given in Figure 3.9. These input parameters were chosen intuitively to simulate certain behaviors from the UAV/UGV system. These input parameters are intended to characterize different behaviors in the system and keep the findings general to understand the behaviors of the system. In addition, in Table 3.2 we summarize the effects of specific parameter inputs as well as highlight the total path cost. It is observed that if the UAV has enough energy capacity $D_{\max}$ then the final tour does not use the UGV (Figure 3.10a and Figure 3.10b). One effective way to obtain the value for $D_{\max}$ would be to conduct an energy calibration flight test where the UAV is flown from 100% to 0% energy. If the UGV is significantly slower than the UAV and $D_{\max}$ is small, then the UAV only recharges in-place (Figure 3.10c, Figure 3.10d, and Figure 3.10e) and does not use **M-DTU** or **F-DTU** edges. However, if the UGV is not as slow, then the tour will use **M-DTU** and **F-DTU** edges (Figure 3.10f). In Figure 3.8 and 3.10e the UAV uses both multi-rotor and fixed-wing flight modes as well as recharges on the UGV. Figure 3.11 shows Figure 3.10e with the proper Dubins paths plotted. For all cases, it is always possible to find a feasible solution since the graph is fully connected and obeys the assumptions given in Section 3.2. We present quantitative evaluation of these parameters next.

C Effect of Changing $D_{\max}$ on the Tour Cost

Next, we study the effect of changing the total battery capacity, i.e., changing $D_{\max}$, on the total tour time. We randomly generate 15 boustrophedon cells in a 100m $\times$ 100m environment such that no two boustrophedon cells intersect with each other and each boustrophedon cell is no more than 10 meters long. Figure 3.12a shows one example.

We vary $D_{\max}$ from 10 meters to 50 meters. We use the same set of 15 boustrophedon cells for each value of $D_{\max}$. Figure 3.12b shows the average, minimum, and maximum value of the optimal tour cost. It is observed that the tour cost decreases as $D_{\max}$ increases, as is expected. It is also observe a step decrease in the minimum and maximum tour costs as $D_{\max}$ increases. This can be attributed to the fact that as $D_{\max}$ increases the UAV can travel a larger distance without running out of energy. Therefore, it may need to land/take-off fewer number of times. Each landing and taking-off operation costs a fixed amount of time. Therefore, it is seen that a step decrease in the tour cost as $D_{\max}$ increases is observed.

D Effect on the Computational Time

Next, we empirically analyze the computational time as a function of some of the input parameters.

Figure 3.13a shows the effect of increasing the number of input boustrophedon cells on the computational time. The input number of boustrophedon cells is varied from 10 to 50 in steps of 1. The figure shows the average, minimum, and maximum computational times for 10 random instances. The input parameters for these experiments were kept the same: $t_{TO} = 100$, $t_L = 100$, $r = 2$, $C = 20$, $fRatio = 3$, and $TR = 1$.

Figure 3.13b shows the effect of increasing the battery level, i.e., C, on the computational

time. We vary C from 10 to 100 in steps of 10. The input parameters for these experiments were: $t_{TO} = 100$, $t_L = 100$, $r = 2$, $fRatio = 3$, and $TR = 1$. The figure shows the average, minimum, and maximum computational time for 10 random instances with 15 boustrophedon cells each.

It is observed that the computational time increases (perhaps, exponentially) with increasing the number of input boustrophedon cells and battery level. Nevertheless, the computational time is still small enough (less than 50 minutes) for moderately sized instances (50 boustrophedon cells).

3.5 Field Experiments

We conducted proof-of-concept field experiments using the UAV and UGV shown in Figure 3.14c. The UAV is a DJI 450 frame [9] with a Pixhawk 2.1 [48] flight controller running the APM firmware [12] and the UGV is a Clearpath Husky [22]. The UAV is equipped with dual GPSs, a downwards facing LiDAR (for relative altitude estimation) and the IR-Lock infrared camera [53]. The UGV is fitted with infrared LED beacons. The IR-Lock system [53] allows for precision landing on the UGV with up to 10cm accuracy in nominal wind conditions. More details on the system are reported in the prior work [122].

Figure 3.14a shows the input boustrophedon cells for the proof-of-concept experiment conducted at Kentland Farms at Virginia Tech. The motion of the UGV is restricted to only those sites that lie on the road. Specifically, edges that correspond with sites are on the road are allowed to conduct recharging. These sites are marked in red in Figure 3.14a. The output tour for the UAV is shown in Figure 3.14b. The following parameters were used as input to the outdoor field experiments: $t_{TO} = 100$, $t_L = 100$, $r = 2$, $D_{\max} = 1000$, 13 boustrophedon cells, $C = 100$, $fRatio = 0.5$, and $TR = 3$. Since the platform is a multi-

rotor, we set $fRatio < 1$. This forces the fixed-wing flights to be more expensive than the multi-rotor ones. With the addition of TR, the algorithm will never use any edges that use the fixed-wing mode.

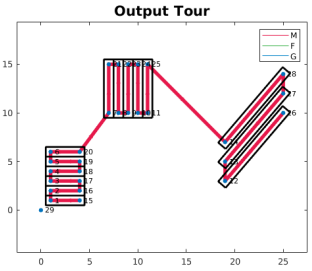
The GPS trace of the UAV and the UGV are shown in Figures 3.14d and 3.14e. Both robots were fully autonomous during the trial that lasted 12 minutes, including taking-off and landing from the ground robot. The trial provides a proof-of-concept demonstration of the algorithm.

3.6 Conclusion

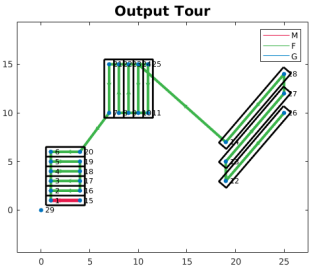
We present an algorithm for optimal coverage of boustrophedon cells with an energy-limited UAV and a UGV. The UGV acts as a mobile recharging station that can mule the UAV between sites, while the UAV can switch between multi-rotor and fixed-wing flight modes. We analyze the effects of various input parameters on the total tour cost as well as the computational time. We show six different sets of input parameters that exhibit different behaviors of the system such as no recharging, stationary recharging only, and mobile recharging as well as choosing between either multi-rotor and fixed-wing modes. Additionally, we show that the algorithm outperforms a baseline approach by finding an optimal tour in significantly less time than the baseline. We evaluate the algorithm through field experiments.

If the UGV is slower, it is possible that the UAV may reach a site before the UGV. In [123], we showed how find the minimum number of UGVs required to ensure that the UAV can execute its tour without having to wait for the UGV. A possible extension is to find a tour for a fixed number of slower UGVs that still ensures that the UAV does not need to wait for the UGV. We restrict the UAV to land and take-off only from the entry/exit sites of a boustrophedon cell. A possible extension would be to relax this assumption which can result

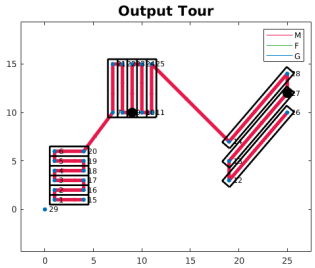
in even shorter tours. To reduce the total computational time we could investigate methods similar to those of Lewis et al. [66] that study how to optimize the number of boustrophedon cells used for coverage. Another avenue for future work would be to extend this chapters work to account for multiple UAVs. With multiple UAVs, there would be more complications with the scheduling of the UAVs and coordination with UGVs. This would require optimizing the wait time of the UAVs, optimizing the wait time of the UGV, and creating paths for the UAV to minimize the driving distance of the UGV. Due to the increased complexity of multiple UAVs, we focus on the unsolved problem of a single hybrid UAV and UGV team for persistent missions.



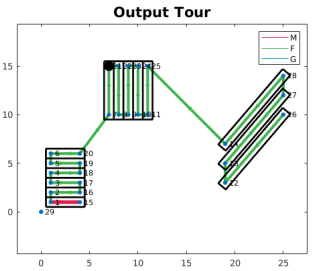
(a) Multi-rotor only with input $t_{TO} = 5$, $t_L = 45$, $r = 2$, $D_{\max} = 200$, UGV speed is one-fifth of the UAV, $C = 20$, $fRatio = 1$, and $TR = 5$



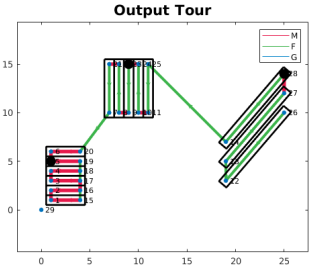
(b) Fixed-wing only with input $t_{TO} = 5$, $t_L = 45$, $r = 2$, $D_{\max} = 10$, UGV speed is one-fifth of the UAV, $C = 20$, $fRatio = 50$, and $TR = 1$



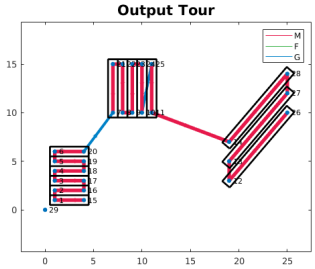
(c) Multi-rotor with stationary recharging only with input $t_{TO} = 5$, $t_L = 45$, $r = 2$, $D_{\max} = 50$, UGV speed is one-fiftieth of the UAV, $C = 20$, $fRatio = 1$, and $TR = 5$



(d) Fixed-wing with stationary recharging only with input $t_{TO} = 5$, $t_L = 45$, $r = 2$, $D_{\max} = 20$, UGV speed is one-hundredth of the UAV, $C = 50$, $fRatio = 6$, and $TR = 0.5$



(e) Mixed flight with stationary recharging only with input $t_{TO} = 5$, $t_L = 45$, $r = 2$, $D_{\max} = 20$, UGV speed is one-fiftieth of the UAV, $C = 20$, $fRatio = 3$, and $TR = 1$



(f) UGV recharging only with input $t_{TO} = 5$, $t_L = 45$, $r = 2$, $D_{\max} = 50$, UGV speed is one-fifth of the UAV, $C = 20$, $fRatio = 1$, and $TR = 5$

Figure 3.10: 3.9 Input for qualitative examples to help explain input parameters and the effects. 3.10a results in a tour that uses only the UAV in the multi-rotor configuration. 3.10b results in a tour that uses only the UAV in the fixed-wing configuration. 3.10c Minimum number of landings/take-offs in place for the given input parameters while staying in the multi-rotor configuration. 3.10d Minimum number of landings/take-offs in place for the given input parameters while staying in the fixed-wing configuration. 3.10e Minimum number of landings/take-offs in place for the given input parameters. 3.10f Minimum number of landings/take-offs using the UGV to recharge for the given input parameters. In each figure bolded black sites represent stationary recharging and bolded blue sites represent the start end end sites of the tour. Note that the flight along the last boustrophedon cell can be either multi-rotor or fixed-wing because there is no cost for switching between flight modes and the UAV will not have to charge at the last location.

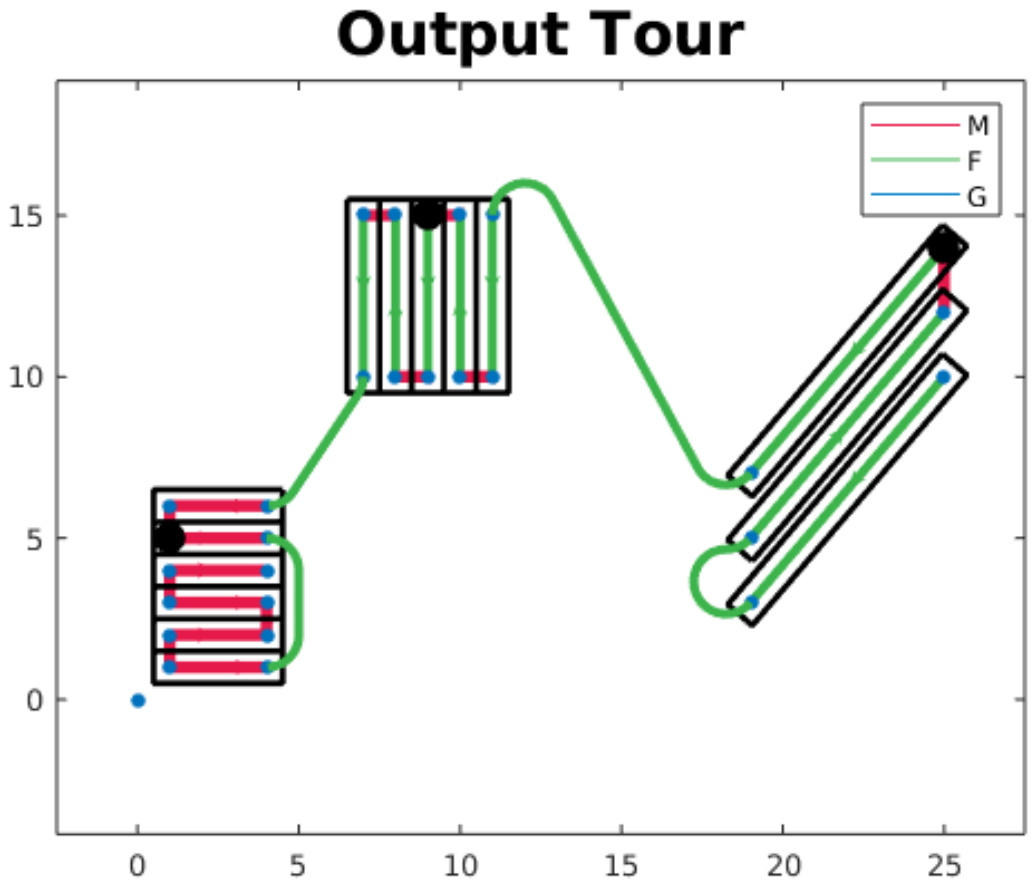
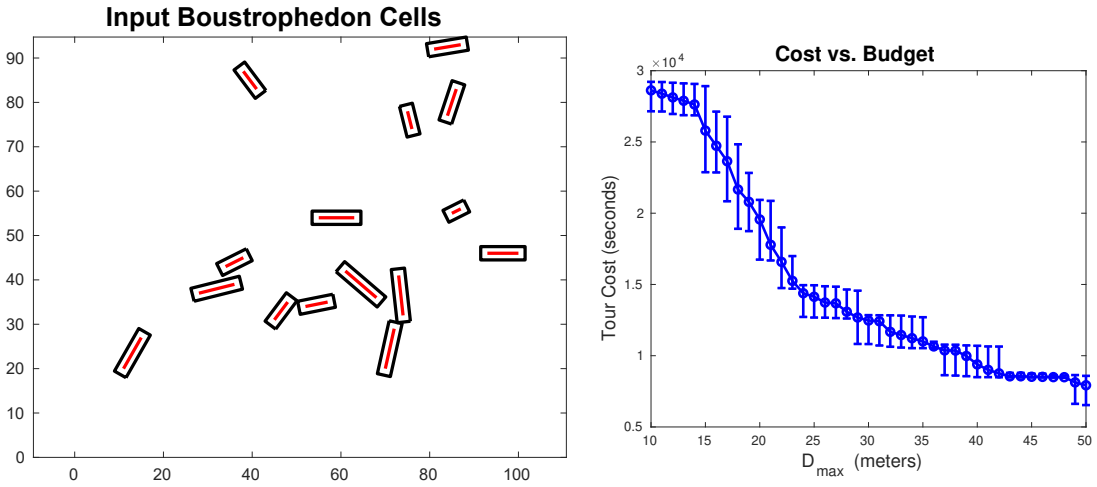
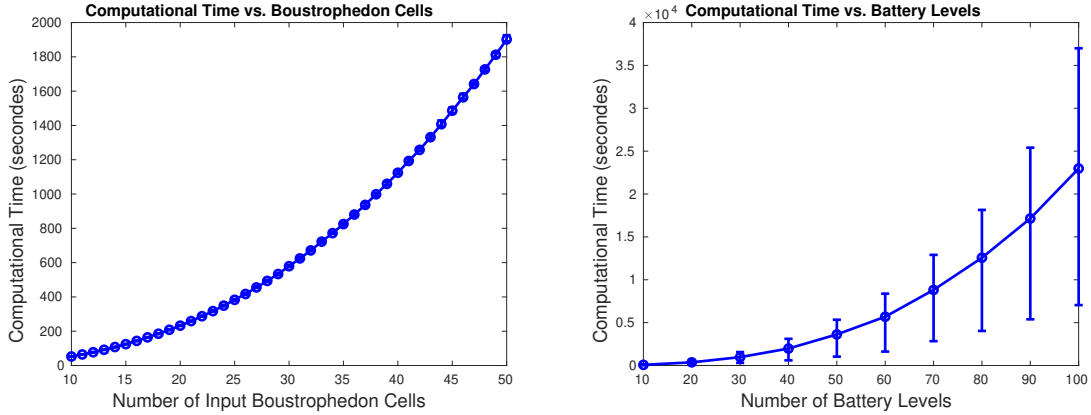


Figure 3.11: Figure 3.10e with Dubins paths for all transitions between boustrophedon cells in the fixed-wing mode.



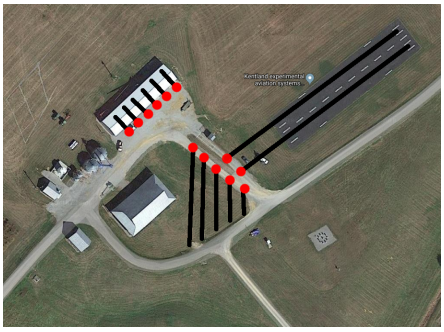
(a) Input (b) Cost vs. budget

Figure 3.12: 3.12a Example input boustrophedon cells for the 10 trials used for generating 3.12b. We randomly create 15 non-overlapping boustrophedon cells, each no more than 10m. 3.12b Tour cost vs. flight budget, $D_{\max}$. We vary the total budget as well as the distance per battery level. The input parameters were: $t_{TO} = 1000$, $t_L = 1000$, $r = 2$, $C = 20$, $fRatio = 3$, and $TR = 3$. The UGV is 5 times slower than the UAV.

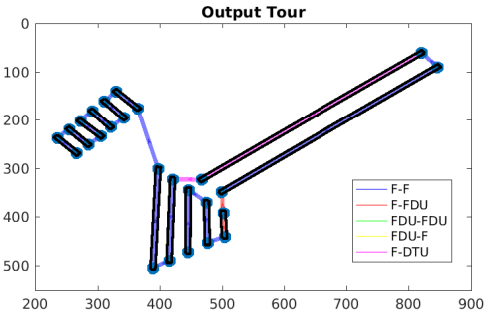


(a) (b)

Figure 3.13: Input parameters: $t_{TO} = 100$, $t_L = 100$, $r = 2$, UGV speed is one-fifth that of the UAV, $fRatio = 3$, $TR = 1$ for both plots. 10 random sets of input boustrophedon cells were randomly generated in a $100m \times 100m$ environment. We set $C = 20$ with varying boustrophedon cells for 3.13a and vary C with 15 boustrophedon cells for 3.13b.

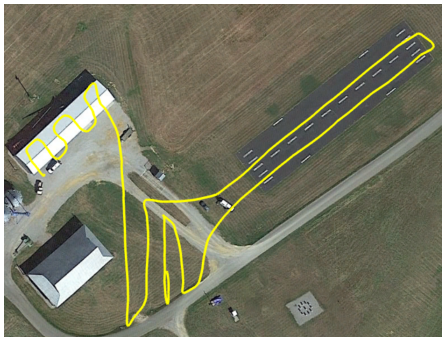


(a) Input



(b) Output

(c) Robots



(d) UAV Path

(e) UGV Path

Figure 3.14: Proof-of-concept Experiment with 13 boustrophedon cells. The input parameters were: $D_{\max} = 1000$, $C = 100$, $t_{TO} = 100$, $t_L = 100$, $r = 2$, UGV speed is one-fifth that of the UAV, $fRatio = 0.5$, and $TR = 3$. The UGV is also restricted to the road network (red sites).

Chapter 4

3D Area Coverage Applications to Infrastructure Inspection

In this chapter, we apply the 2D area coverage algorithm for inspecting planar surfaces of a 3D structure. Specifically, we focus on the application of inspecting infrastructures, such as a bridge, using the 2D algorithm. Here the 2D areas that need to be explored are not all on the same plane. Instead, these are 2D planar surfaces on the bridge (Figure 4.2a) that are provided as input. The GTSP algorithm from the previous chapter can be applied here to find the order in which to inspect the surfaces as well as the entry and exit points.

The challenge we focus on here is safe execution of the planned path. In the 2D case, the UAV flew on a fixed altitude plane without any obstacles in the environment. During bridge inspection, the UAV may not always have access to GPS information. The UAV may not have an accurate model of the environment. Therefore we propose a combination of a high-level planner and a low-level controller to address these challenges. The low-level controller uses LiDAR information to conduct coverage of a 2D region. The high-level planner finds the sequence of visiting the 2D regions.

4.1 System Description

In this section, we describe the hardware and the software architecture of the system.

A Hardware Description

We use the DJI S900 [35] as the UAV platform (Fig. 4.1). This UAV has a hexarotor design. Six motors provide a maximum thrust of 2.5Kg. The UAV is powered by a 6S 10000mAh Lithium-polymer battery and weighs 3.5kg without any payload.

We use the Pixhawk autopilot as the main flight controller [93]. The Pixhawk autopilot has an in-built IMU, compass, accelerometers, and gyroscopes. The Pixhawk runs the PX4 firmware version 1.7.3v [75] for low-level flight control. The UAV is also equipped with an Nvidia Jetson TX2 for high-level control. The software on the Jetson runs on Ubuntu 16.04 and uses **ROS** (Robot Operating System) Kinetic 1.12.13. The Nvidia Jetson TX2 is used for processing sensor data and publishing the desired velocity to the Pixhawk flight controller. The Nvidia Jetson TX2 has a 256-core Pascal GPU, quad-core ARMv8 processor rev3, 8GB of DDR4 memory, 32GB of memory, and uses a maximum of 15W of power.

The UAV is equipped with two Scanse Sweep 2D LiDAR sensors [105] and a GoPro Hero7 Black Camera. The two LiDARs are placed in a horizontal and vertical arrangement. Both LiDARs are used for estimating the distance to the bridge structure and enable autonomous navigation. These sensors have a maximum range of 40m, maximum resolution of 1cm, sample rate of up to 10Hz, 360° field of view, use 5V@650mA, and weigh 120g each. GoPro Hero7 Black Camera is used to collect images of the bridge structure for defect identification. The GoPro Hero7 Black shoots 4K 60FPS video with an aspect ratio of 16:9. This camera can be replaced by a higher resolution inspection-grade camera.

We use the Pixhawk to run PX4 firmware along with the mavros package [73]. Mavros is a ROS package that allows for communication between the PX4 firmware and the onboard Nvidia Jetson TX2. The Jetson TX2 runs software associated with the local navigation routines and the higher-level supervisor. The software is modular with separate ROS nodes

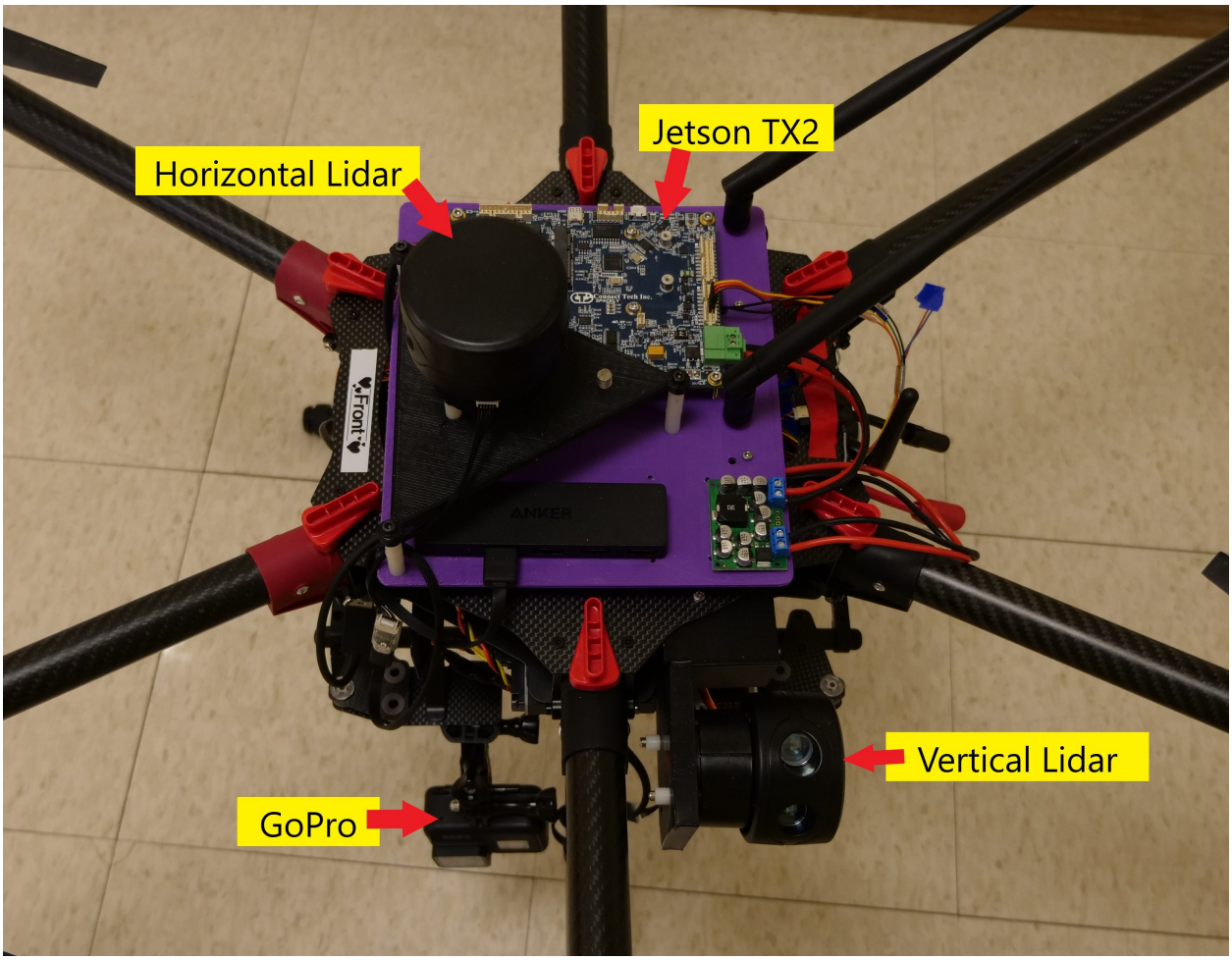


Figure 4.1: UAV for bridge inspection.

running in **ROS** Kinetic. The individual modules are explained in Section B. ROS nodes allow for the publishing and subscribing of data. The supervisor and local navigation routines communicate with each other through publishing and subscribing to figure out what velocity commands need to be sent to the flight controller. Once the correct velocity is figured out the give velocity is broadcast to the UAV flight controller through mavros.

B Algorithm Description

Our current approach for complete inspection coverage of all bridge surfaces is to autonomously execute a series of maneuvers from a library of navigation routines (Fig. 4.4a). While the UAV autonomously navigates the bridge, an onboard camera records images of the bridge that can be examined for defects in real time and/or used for post-processing.

The algorithm consists of three modules:

1. global planner to find the sequence of local navigation routines needed for complete visual coverage of the bridge;
2. local navigation routines for real-time, low-level, LiDAR based navigation; and
3. supervisor to determine if the UAV has completed a local navigation routine and can progress to the next one using LiDAR data.

We describe each of the three modules in details next.

Global Planner

The first step in the proposed algorithm is to find a global tour that the UAV must follow for full visual coverage of the bridge. In this work, we focus on inspecting box girder bridges (shown in Figure 4.2a), in particular all the external surfaces of the bridge (girder, column, deck, etc.).

We first partition the bridge surfaces into a set of planar surfaces, as shown in Fig. 4.2a. Each planar surface can be approximated by a polygon. The UAV can fully inspect the bridge by finding a route that inspects each of the surfaces. This decomposition makes it easier to find a global plan as well as make it easier for the operator to understand the plan.

To inspect each surface, the UAV needs to visually cover the corresponding polygon. The local navigation controller ensures that the robot moves in such a way so as to visually cover each polygon. Therefore, the goal of the planner is to find the sequence in which to visit the polygons as well as the entry and exit points for each polygon. Here, we take advantage of the specific structure of the box girder bridge. We associate with each polygon two points, which will represent coverage of the polygon from both directions. This means that one of the nodes will be chosen as the entry node to the planar surface and the other node is chosen as the exit node. The local navigation routine controller will ensure navigation between the entry and exit nodes and ensure visual coverage of the bridge surface. Since there are no physical markings on the bridge surface, the supervisor will identify when the exit node of the current polygon/entry node of the next polygon has been reached and pick the appropriate local navigation routine to continue rest of the visual inspection tour.

The visual coverage problem is that of finding the sequence in which the bridge surfaces must be traversed, as well as determining the entry node for each of the polygons representing the surfaces. We formulate visual coverage as a GTSP [85]. The input to GTSP is a graph where the nodes are partitioned into disjoint clusters. The goal is to find the minimum cost tour that visits at least one node in each cluster. GTSP generalizes the NP-Hard Traveling Salesperson Problem, and therefore is NP-Hard as well. Nevertheless, there are good numerical solvers present [102] that can be used to find the optimal solution for reasonably-sized instances.

We convert the visual coverage problem into a GTSP instance as follows. We create one cluster for each of the polygons. Each cluster contains the two nodes associated with that polygon, as shown in Fig. 4.2b. One of the two nodes will be chosen as the entry node and the other will be chosen as an exit node by the algorithm. An edge is created between every pair of nodes that belong to separate clusters. These edges are the combination of coverage of the cluster and then traveling to another cluster. In Fig. 4.2b if the UAV were to go from

node 1 to node 7, the UAV would have to cover cluster A and then go to node 7 by going from node 1 to node 6 then to node 7. Once that is done the algorithm filters out edges that cannot be traversed by one of the local navigation routines.

For any cluster X, let X_1 and X_2 represent its entry and exit nodes respectively. Then, the cost on an edge from node A_1 in cluster A to node B_1 in cluster B is given by,

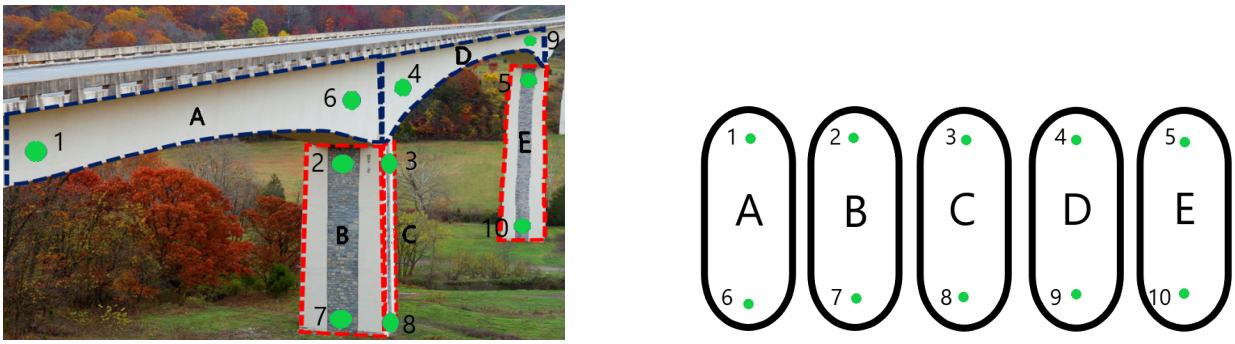
$$C(A_1, B_1) = D(A_1, A_2) + D(A_2, B_1)$$

where $D(A_2, B_1)$ represents the distance between nodes A_2 and B_1 in 3D space, i.e., the cost is the sum of the distance to cover the current polygon (surface) and the distance to reach the entry node of the next polygon from the exit node of the current polygon. When conducting experiments, a picture of the bridge is used to obtain an estimate of distances. Using rough estimates of the distances between entrance and exit nodes for each planar surface will still allow me to obtain an accurate solution as long as it is to scale.

Instead of using the actual 3D distances (which will not be known because there is no 3D model of the bridge), distances can be estimated. In this work, we use a side-view image of the bridge to find the scaled distances (in pixel units) between the nodes.

This is the input to the GTSP solver. We use the Generalized Large Neighborhood Search (GLNS) solver [102] which is the state-of-the-art for GTSP instances. Once a solution from the GLNS solver is obtained, the full tour to be visited by the UAV can be constructed. The GLNS solver will give me a solution in the GTSP instance of the problem. To convert that back to the original problem we examine the order in which the clusters are visited and the entry points, which both are given by the solver. In Fig. 4.2b if the output from the GLNS solver was 1, 2, 8, 4, 5 then we know that the clusters are visited in the order of A, B, C, D, E. However, we need to transform the output from the GLNS solver into something that

the UAV can use for navigation along the bridge surface. To do this we take each point in the output from GLNS and add the corresponding exit point. For the algorithms output we would change the output to **1**, 6, **2**, 7, **8**, 3, **4**, 9, **5**, 10, with the bolded values as the original GLNS output. Now that we have the output the UAV can execute full traversal of the bridge. By using a GTSP formulation, we can guarantee that the UAV visits every cluster exactly once, which implies the UAV visit each polygon surface that needs to be inspected exactly once. The local navigation routine ensures that each surface is inspected as required.



(a) (b)

Figure 4.2: (a) Bridge partitioned into surfaces with entry/exit nodes (green dots). (b) GTSP clusters with entry/exit nodes (green dots) corresponding to Fig. 4.2a.

Local Navigation Routines

For safe and accurate autonomous traversal of bridge surfaces, we employ an algorithm that must be fast enough to run in real-time as well as robust to factors such as wind disturbances, unreliable GPS, and inaccurate compass measurements. When examining girder style bridges there are four main parts that need to be traversed the girder, column, bottom, and top, shown in Fig. 4.3. To traverse all of these parts of the bridge there is a total of eight local navigation routines. To traverse the girder of a bridge there are two methods that need to be employed and for traversal of the column there are two. Also when inspecting the top

and bottom of the bridge surface there are two methods for each of them. For girder flight, Fig. 4.3, a right and left local navigation routine and for column flight, Fig. 4.3, up and down local navigation need to be executed. Similar to the girder local navigation routines the top flight and bottom flight, Fig. 4.3, of the bridge needs right and left local navigation routines. In total that leaves me with girder right(GR), girder left(GL), column up(CU), column down(CD), bottom right(BR), bottom left(BL), top right(TR), and top left(TL). For this chapter we focus on coverage of one side face of the bridge. Through the rest of the chapter we will be using the above notation. Due to this we only use four local navigation routines (GR, GL, CU, and CD).

We employ 2D LiDAR (a rotating 1D LiDAR) based local control for this purpose. By using two 2D LiDARs, one placed horizontally and one placed vertically, we can obtain real-time data as well as have robust navigation. We implement all routines for traversing along the girder of the bridge and along the columns of the bridge. The four local navigation routines only use the two 2D LiDARs, allowing the UAV to navigate in environments that contain no GPS signal. We exploit the geometry inherent to bridges and treat the bridge surfaces as planes. A 2D LiDAR scan of a plane shows up as a line as shown in Fig. 4.4b.

A Hough Transform based approach is used to find the best fit line that approximates a bridge surface. We implement a PID control to fly the UAV parallel to the bridge surface, maintaining a fixed distance as well as a specific distance from the top and side of the girder and column respectively. The software architecture is illustrated in Fig. 4.5. A more detailed description of Estimation and Control is provided below.

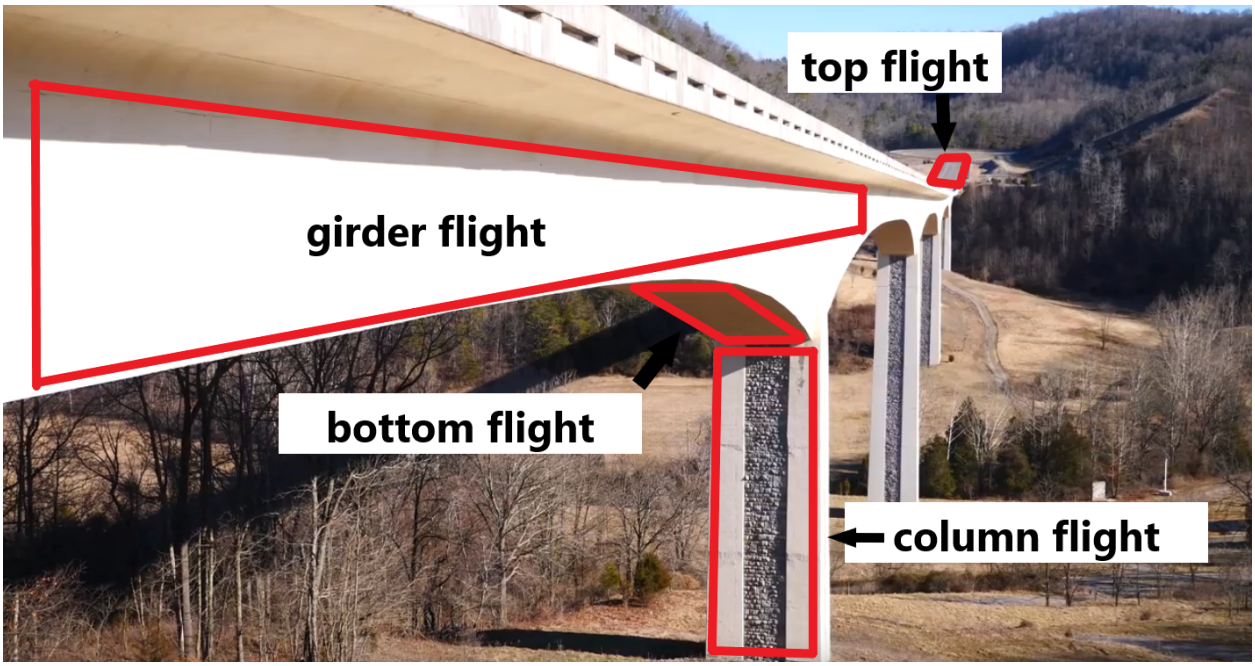


Figure 4.3: Bridge showing the different possible local navigation routines. For girder flight it extends along the full bridge and is replicated for the opposite side. Bottom flight is along all bottom portions of the bridge as well as top flight is along the full top of the bridge. Column flight is also duplicated along all column surfaces.

Estimation

To understand how bridge surfaces are estimated, consider, the problem of estimating the vertical span of the bridge girder surface (highlighted in red in Fig. 4.6a). We start by obtaining the 2D point cloud data from the vertical LiDAR. Once we get the data the vertical span of the bridge girder surface is estimated. The following are the steps involved (explained with reference to Figs. 4.6a, 4.6b & 4.6c). Data points that are very close to the body of the UAV and very far from the body of the UAV (ground plane, distant trees, etc.) are initially filtered out (circled in blue in Fig. 4.6c). Then a Hough Transform to extract lines from the filtered data is executed. Lets make the assumption that the three lines as shown in Figs. 4.6a & 4.6c are found with enough confidence. The confidence is directly related to the number of raw data points that fall on the extracted lines. The UAV

(a) (b)

Figure 4.4: (a) Illustration of some useful navigation routines. (b) White dots: LiDAR data; Red line: best fit using Hough Transform.

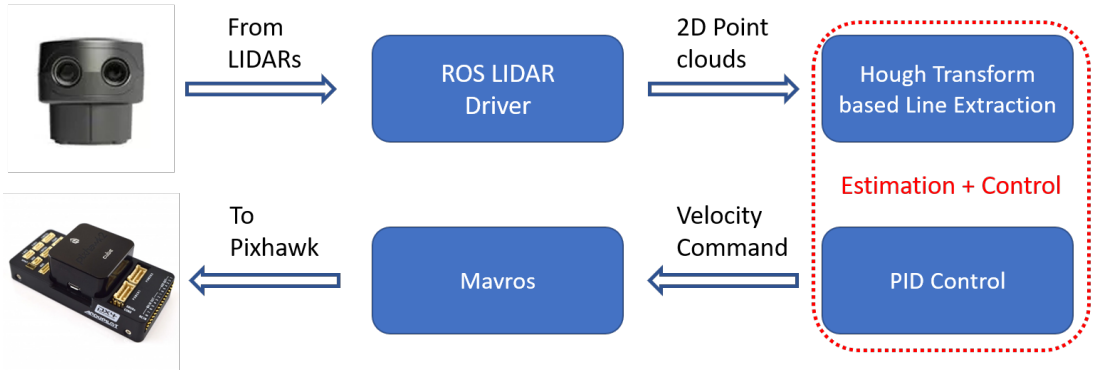
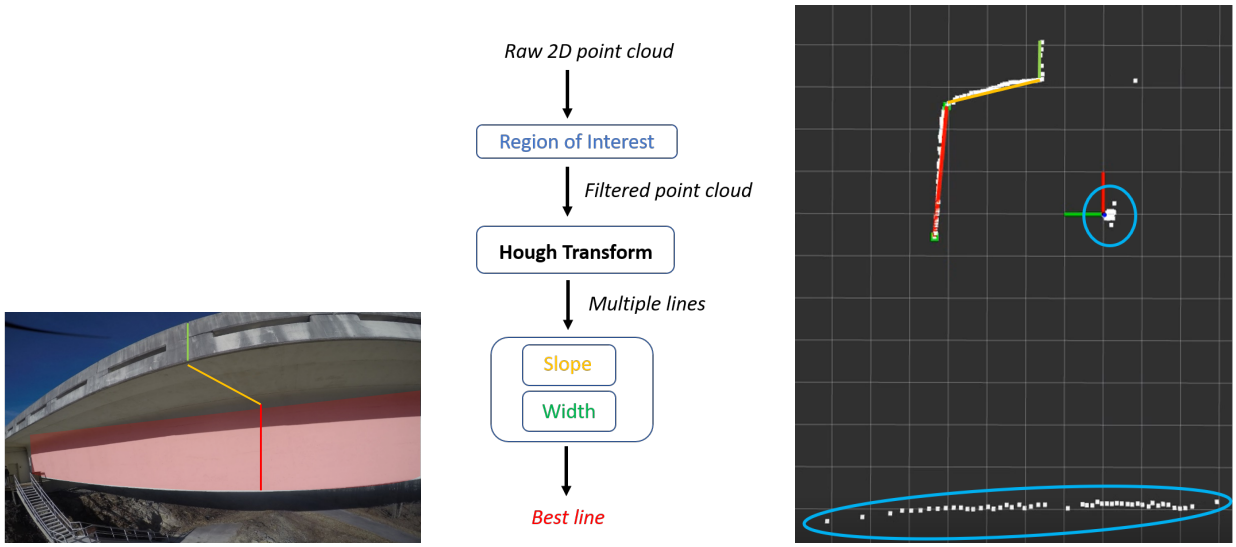


Figure 4.5: Software architecture for autonomous navigation.

is interested in extracting only the red line which approximates the vertical span of the bridge girder surface. The UAV expects this line to have the characteristics of $\approx 90°$ and width $3-5$m. Due to this, the yellow line with slope $\approx 10°$ and the green line with width ≈ 1m will be filtered out because the characteristics do not match up with the expected vertical span of the bridge (Figs. 4.6a & 4.6c). These expectations are specific based on what bridge is being traversed and will need to be changed based on the current bridge. However, the UAV does not need to know these accurately and only needs an estimate to allow for the UAV to navigate the bridge.



(a) (b) (c)

Figure 4.6: (a) The bridge girder surface highlighted in red. (b) Steps involved in the Hough Transform based Line Extraction process. (c) Visualization (rviz) of filtered out points and extracted lines.

Control

The goal of the algorithm is for the UAV to fly parallel to the (estimated) bridge surfaces. This involves implementing two independent PID loops along two perpendicular axes to maintain position with respect to the bridge surfaces. For *GR* and *GL* flight, the UAV maintains altitude and distance relative to the bridge. For *CU* and *CD*, flight the UAV maintains distance from the column edge and distance relative to the bridge. Dependent on the desired local navigation routine a constant nominal velocity along the axis of movement maneuvers the UAV parallel to the bridge surface, right and left for girder flight and up and down for column flight.

As an example, for the flight *GR* (Fig. 4.7):

- One PID loop maintains the desired distance along the axis perpendicular to the girder surface (green axis).

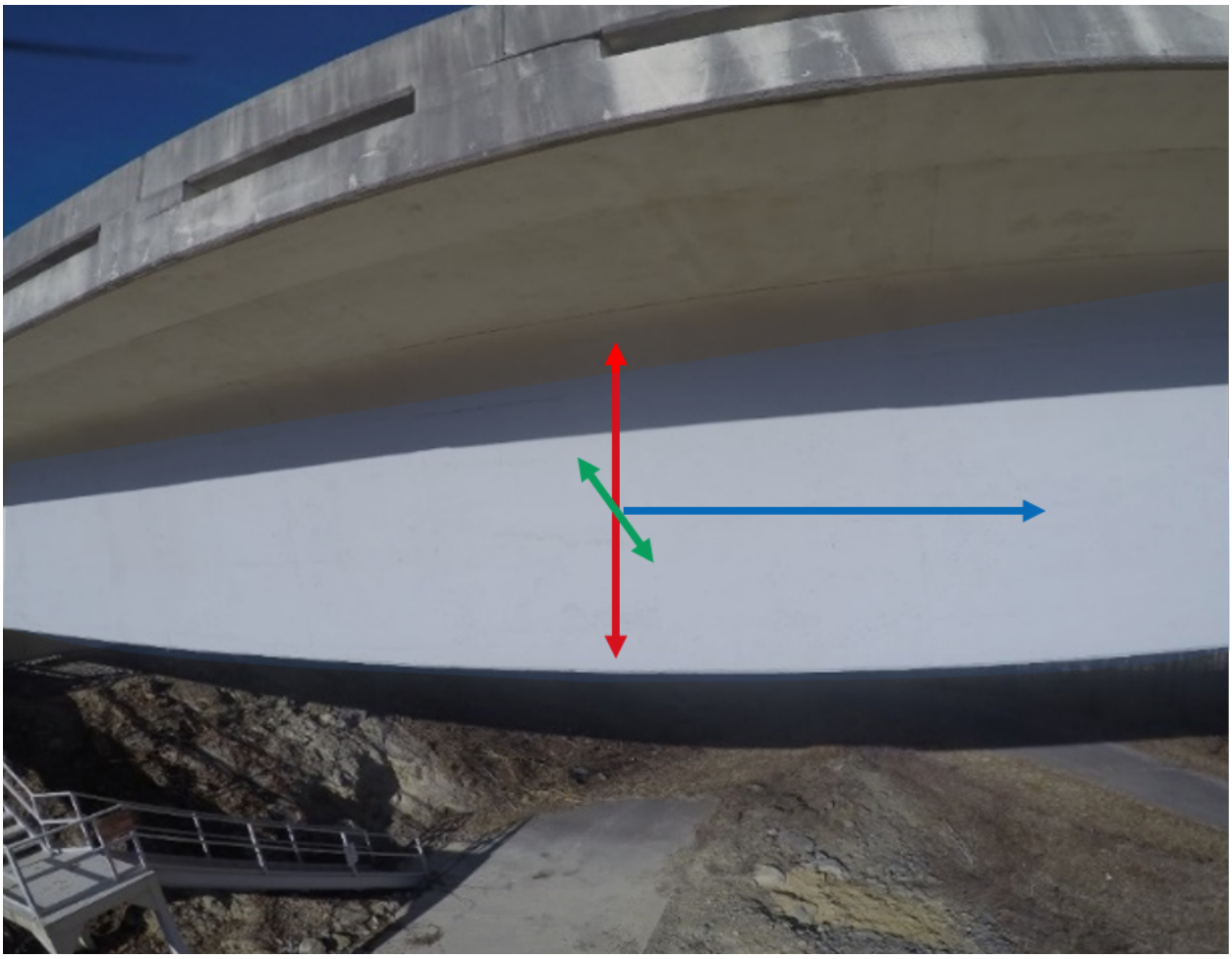


Figure 4.7: Independent PID loops maintain position along the red and green axes, while a constant velocity is given along the blue axis.

- The second PID loop maintains altitude w.r.t the top of the bridge girder (red axis).
- A nominal velocity drives the UAV along the axis parallel to the girder surface (blue axis).

Supervisor

The role of the supervisor is to make the decision of when to switch between local navigation routines. The supervisor executes autonomous switching between local navigation routines in the order that is given by the planner, described in Section B.

When making the decision to switch between local navigation routines the structure for special characteristics in the data from the two 2D LiDARs is examined. This allows the UAV to identify when to switch between local navigation routines. For instance, consider the scenario where the UAV is traversing up a bridge column using the local navigation routine *CU*, as shown in Figs. 4.8(a)(b). When analyzing this behavior, the horizontal LiDAR data is used to determine if the UAV can switch routines. When executing *CU* along the column of the bridge the horizontal LiDAR data is constant. However, when encountering the bridge girder, the number of data points from the horizontal LiDAR increases dramatically as shown in Figs. 4.8(c)(d). Due to this behavior, the supervisor knows that the UAV can autonomously switch from *CU* routine to the either *GR* or *GL* routines.

A coarse estimation of the global position can be used to better inform the supervisor. If the global position of the UAV was known then the UAV could use that information to decide when to switch. However, due to the inaccuracy of GPS around bridge structures, all GPS data was removed for the simulations and experiments.

It is to be noted that the supervisor need not necessarily be fully automated even though a fully autonomous method is implemented. Currently the supervisor executes fully au-

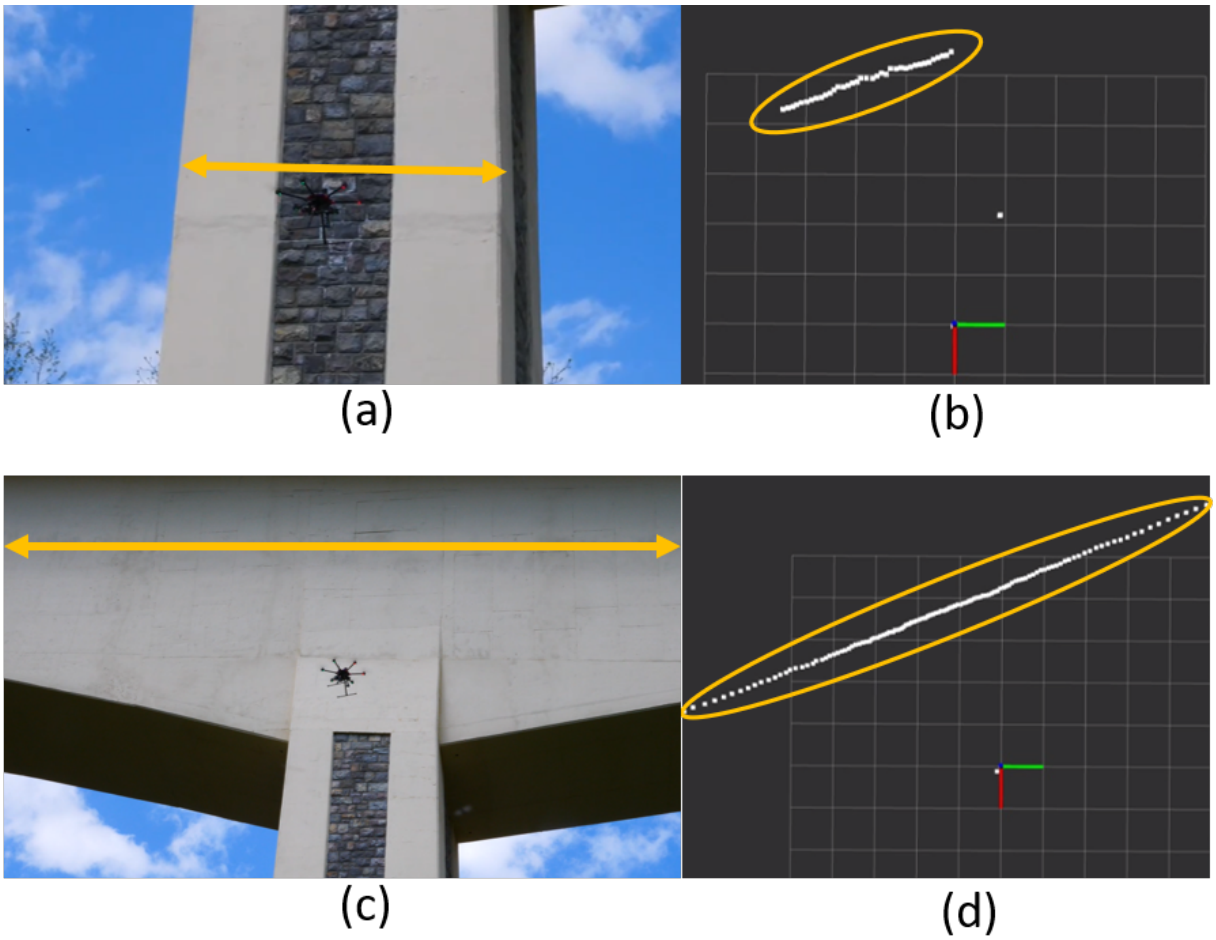


Figure 4.8: Switching from column follow to girder follow.

tonomous flight for 2D surfaces at a time, for instance one surface of the bridge containing multiple sections of girders and columns. Once the UAV has completed coverage of one bridge surface using the local navigation routines, a human operator would then switch the UAV to another bridge surface. This can be done if needed and in tricky scenarios such as switching from flight under the bridge to flight beside the bridge. This allows execution of 3D coverage of the bridge autonomously with a minimum human operator intervention.

4.2 Experiments and Results

A Individual Routines

We conducted experimental flights at the Virginia Tech Transportation Institute (VTTI) Smart Road Bridge, which is 53m tall, 609m long, 5 span, variable height concrete box girder bridge (Fig. 4.9). We tested LiDAR based autonomous flight (no GPS) with a GoPro camera onboard collecting images of the bridge.

Figure 4.9: Experiment at the VTTI Smart Road Bridge.

Flight beside bridge girder

The objective of the experiment was to have the UAV fly alongside the bridge (i.e., *GR*) with a nominal velocity of 0.5m/s maintaining a horizontal separation of 4.5m to the girder. The LiDAR scan (rotation) rate was 2Hz, and hence the control signal (velocity in the direction

perpendicular to the bridge) was also issued at 2Hz. The LiDAR can scan at up to 10Hz, and a higher control rate can be employed, for instance, in windy conditions. The algorithmic computations only take approximately 1ms. Fig. 4.10a shows the control signal driving the UAV to the desired separation of 4.5m from the structure. Hence, we could experimentally verify that accurate flight alongside the bridge at a fixed horizontal distance is feasible, which is helpful for consistent data (image) collection during flight.

Maintaining the correct altitude with respect to the bridge structure proved to be more challenging. Since the terrain around the bridge is not even, using a downward facing LiDAR is not an option for maintaining altitude. The UAV could not rely on the barometer alone, not only because of barometric drifts and inaccuracies, but also because the bridge may slope up or down. Hence, the UAV needs some method to hold position w.r.t. the bridge structure itself. For this purpose, the UAV uses the vertical cut from the 2D LiDAR rotating in the vertical plane that gives me a cross-section of the bridge. The vertical cut was used to maintain an altitude of 2.5m below the top of the girder as shown in Fig. 4.10b.

Flight along bridge column

The objective of this experiment was to have the UAV execute *CU* and *CD* along the bridge column with a vertical nominal velocity of 0.5m/s or -0.5m/s, while compensating in the horizontal direction. The UAV maintains a separation of 4.5m w.r.t. the column. The experimental results in Figs. 4.10c & 4.10d show the control algorithm compensating to maintain the center and the desired distance to the column.

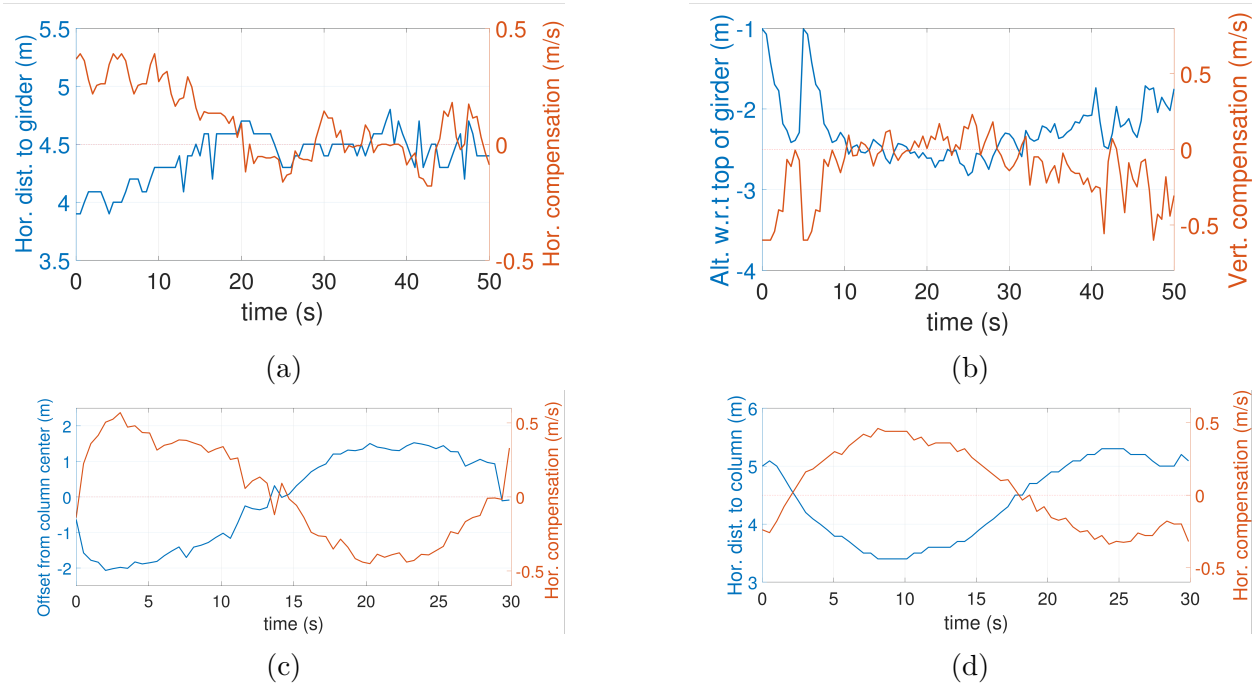


Figure 4.10: (a) Correction velocity in the direction perpendicular to the bridge structure maintaining the desired distance of 4.5m from it. (b) Correction velocity in the vertical direction maintaining the desired altitude of 2.5m below the top of the girder. (c) Correction velocity in the direction tangential to the bridge column centering the UAV w.r.t the column. (d) Correction velocity in the direction perpendicular to the bridge structure maintaining the desired distance of 4.5m from it.

B Full Scale Sims

In addition to conducting experiments for the individual routines, we conduct simulation experiments on full scale bridges. These bridges are modeled in Gazebo 7. We start by creating a bridge in SketchUp and exporting a .dae file to create a model in Gazebo. In Fig. 4.11 we show the full image of the bridge with annotations to represent individual sections. In Fig. 4.11 the green nodes represent possible locations of switching and the red outlines represent the polygons created. We obtain the optimal tour output from the GTSP planner. As a reminder the planner is initially fed a set of planar surfaces, shown in Fig. 4.11. We use Fig. 4.11 to obtain a rough estimate of the distances between entrance and exit nodes

for each planner surface. This means that the edge costs are not the actual bridge distances, but a scaled version. The reason for a non-exact measurement is to replicate what a human inspector/operator might have on hand at a work site. Since inspectors do not always have access to exact specifications of a bridge they might have to take an overview picture and then make rough estimates on the relationships between entrance and exit nodes in the picture.

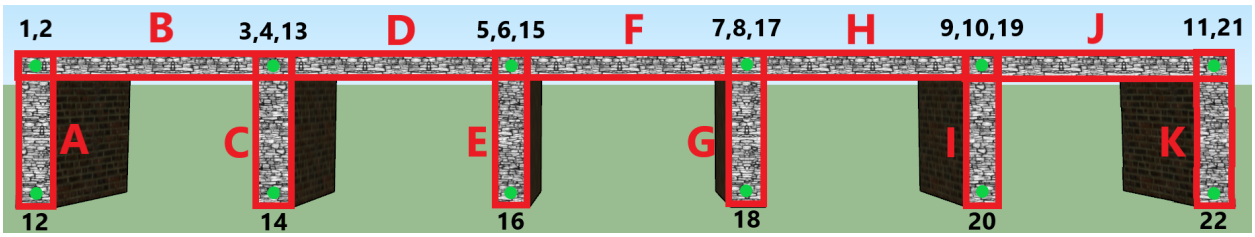


Figure 4.11: Bridge used for full scale simulations split into planner surfaces. The red letters represent the polygon surfaces and black numbers are the ID of the green nodes representing the coverage of each planner surfaces. *GL* and *GR* local navigation routines can be executed on polygon surfaces B, D, F, H, and J. *CU* and *CD* local navigation routines can be executed on polygon surfaces A, C, E, G, I, and K.

We start by converting the instance shown in Fig. 4.11 into a GTSP instance, shown in Fig. 4.12. In Fig. 4.12 we show the clusters created as well as a subset of edges created and the costs. When creating the GTSP instance a fully connected graph is created and then edges are pruned out. The edges that are removed are inter-cluster edges and edges that are deemed impossible because the UAV would lose reference to the bridge structure, not being able to navigate correctly. Once this GTSP instance is created the input can be given to a GTSP solver, GLNS, and an output is obtained.

Once the output of the order in which the UAV should visit from the planner is obtained, shown in Fig. 4.13, it is given it to the supervisor. Fig. 4.13 shows the output from the planner. Each cluster is shown with the ID letter from Fig. 4.13 matching the ones in Fig. 4.11 as well as the node ID numbers. Once the supervisor has the output from the

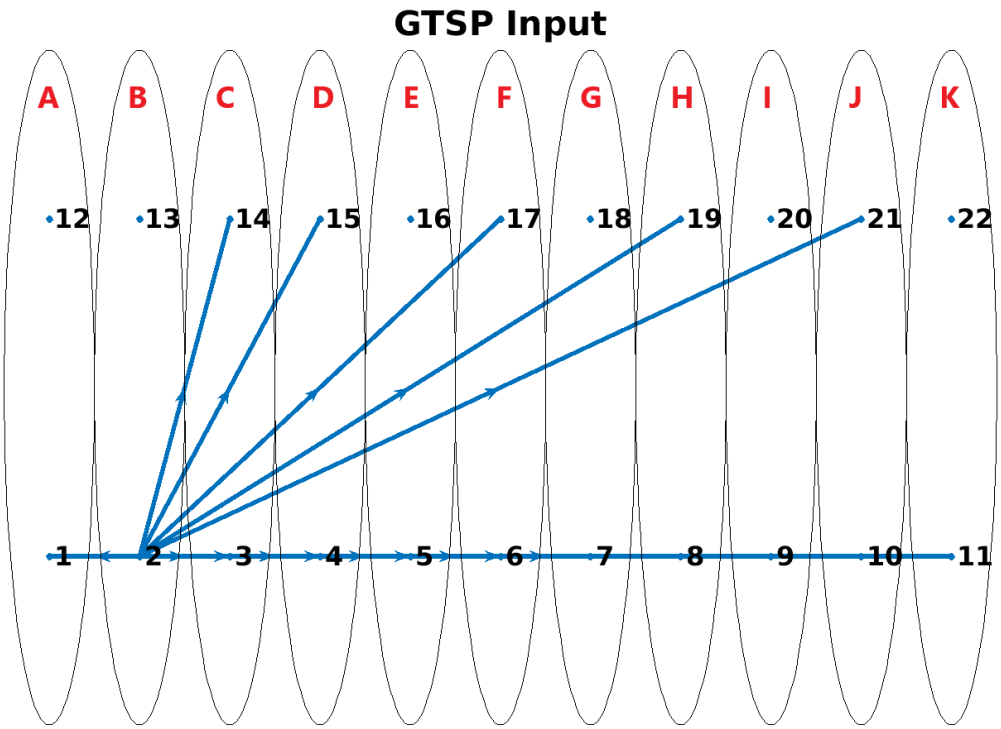


Figure 4.12: Input to GTSP solver. The red letters correspond to the polygon identification numbers in Fig. 4.11 and the black numbers correspond to the green nodes created for each polygon to signify the direction of coverage in Fig. 4.11. Here is a subset of the edges that are created as input to the GTSP solver. We do not display all edges for readability.

planner it is able to execute the full coverage of the bridge's planner surface. For the bridge in Fig. 4.11 the planner took 1.92s and gave me the solution of *CU* on column K, *GL* on girder J, *CD* on column I, and then repeating this until column A is reached. This order of local navigation routines is given to the supervisor and executed in real time.

It can be seen in Fig. 4.14 the path taken by the UAV. A plot of the GPS location of the UAV from the simulation in Matlab is shown and annotated in the figure. In the figure the green nodes are where the UAV Scheduler switched between local navigation routines. The black and red nodes represent the start and end of the UAVs path for coverage of the bridge. This proof-of-concept implementation demonstrates the feasibility of the entire system.

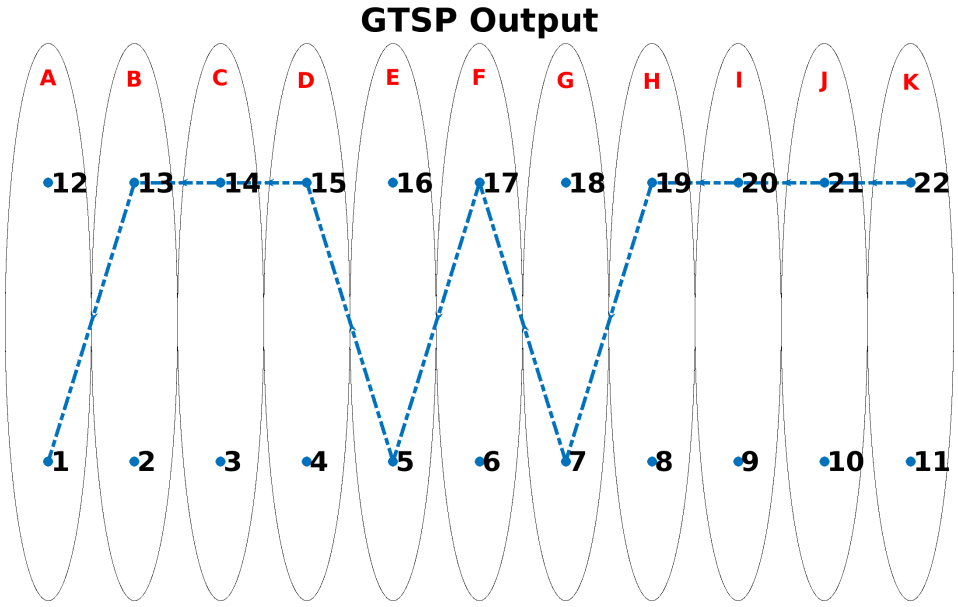


Figure 4.13: Output from GTSP solver. The red letters correspond to the polygon identification numbers in Fig. 4.11 and the black numbers correspond to the green nodes created for each polygon to signify the direction of coverage in Fig. 4.11. The solver starts with polygon K and moves from right to left. With this the output of the GTSP solver gives *CU*, *GL*, *CD* and repeats until the last polygon A ending on *CD*.

4.3 Conclusion

We presented a fast LiDAR based approach for autonomous navigation using two 2D LiDARs scanning in horizontal and vertical planes. We described strategies to switch between navigation routines to cover various bridge surfaces. We envision full coverage of the bridge using a total of eight routines for UAV flight. *GR*, *GL*, *CU*, and *CD* have already been tested and the rest, characterized in Section B, are conceptually similar to the ones already tested. The LiDAR based approach can also be used to provide safety guarantees in assisted manual flights. Operator (pilot) input can be rejected if it will result in the UAV getting too close to the bridge structure. Adapting ideas from the present approach to different types of bridges (which could require working with other sensors such as Stereo camera for navigation) is a

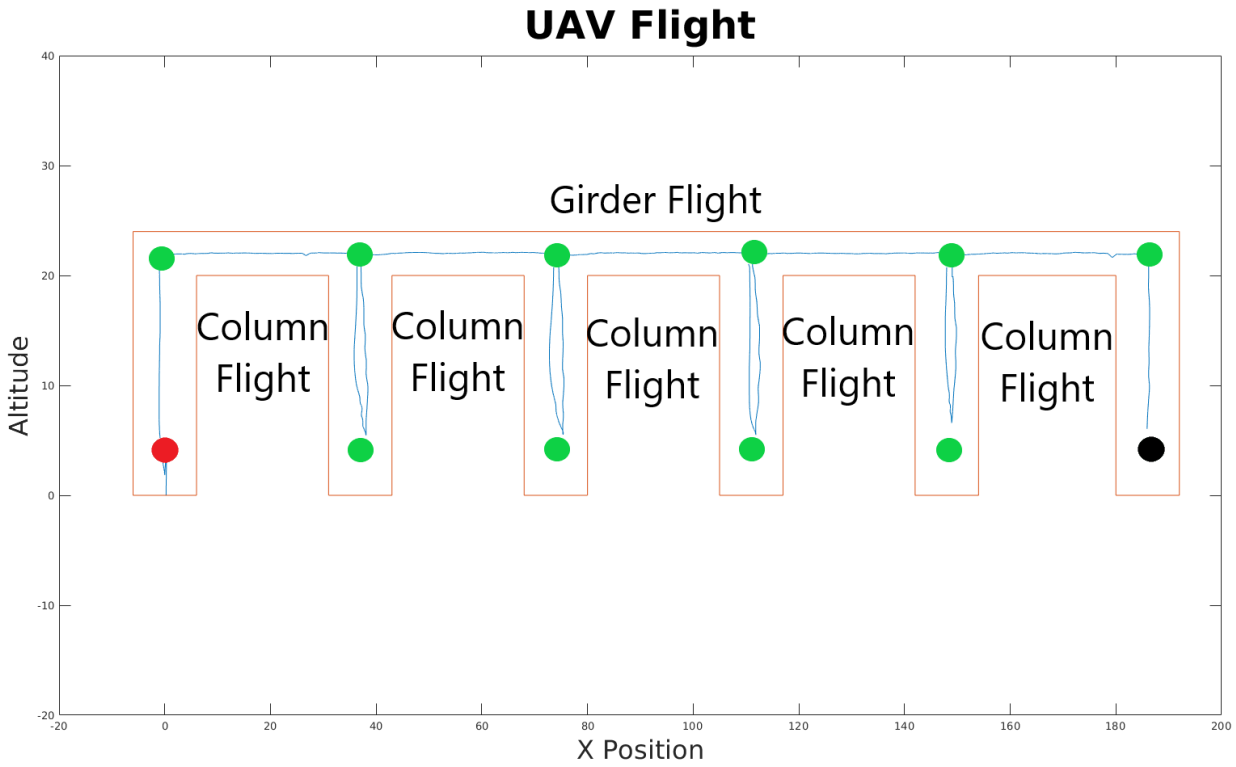


Figure 4.14: Simulation data obtained from mavros of UAV flight on the bridge located in Fig. 4.11. The blue line indicates the actual flight of the UAV and the orange lines represent the bridge structure. The green nodes are locations when the UAV switched modes from girder to column flight or *CD* to *CU* flight. The black and red nodes represent the starting and end location of the UAV path, respectively.

future direction.

Chapter 5

3D Online Surface Inspection

In this chapter, we expand on the work on inspecting planar surfaces of a 3D structure to executing online path planning for full 3D coverage of infrastructures. We focus on inspecting infrastructures, such as a bridge, using an online algorithm that will replan a path for a UAV. Here we do not require a previous map of the infrastructure and instead we will build a voxel map (Figure 5.1) using onboard LIDAR and RGB data. The proposed algorithm, will use a GTSP algorithm to cluster viewpoints that can see the same POI on the bridge to execute faster and more cost effective 3D infrastructure coverage.

In the previous chapter, it is assumed a prior map of the 3D environment was given. Therefore, the planner was an offline one. Here, this assumption is removed. Because a prior map is not given, the UAV needs to segment out the structure of interest in real-time as well as provide a flight path which can execute efficient 3D coverage. Here, the UAV can use the RGB imagery to perform semantic segmentation. Figure 5.2 shows an example where only the voxels corresponding to the bridge have been segmented out. We propose an online algorithm that will replan paths for the UAV to fully cover the segmented 3D structure while taking into account a safety buffer around all obstacles and plan only along the structure of interest. By using a GTSP approach, we can improve the efficiency of the coverage paths compared to baseline frontier-based exploration algorithms.

In this chapter, we will focus on the task of using UAVs for inspecting the surface of a bridge for defects such as corrosion, rust, fatigue, cracks, etc. that can be inspected visually by

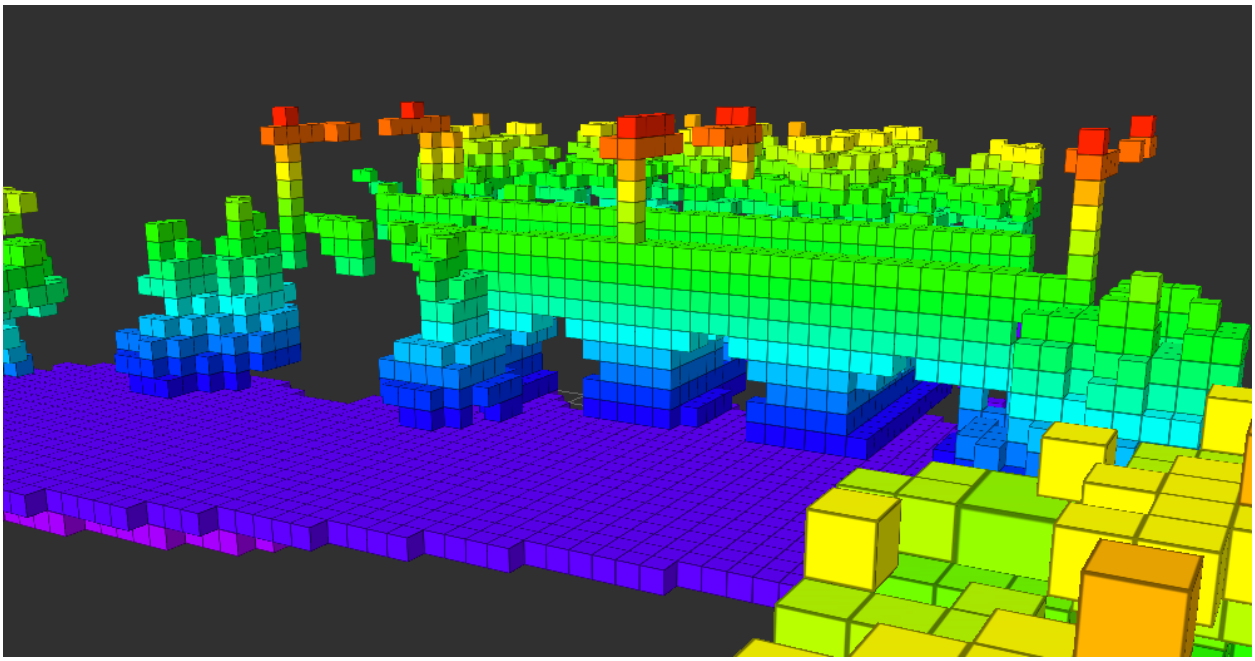


Figure 5.1: Voxel map of a bridge.

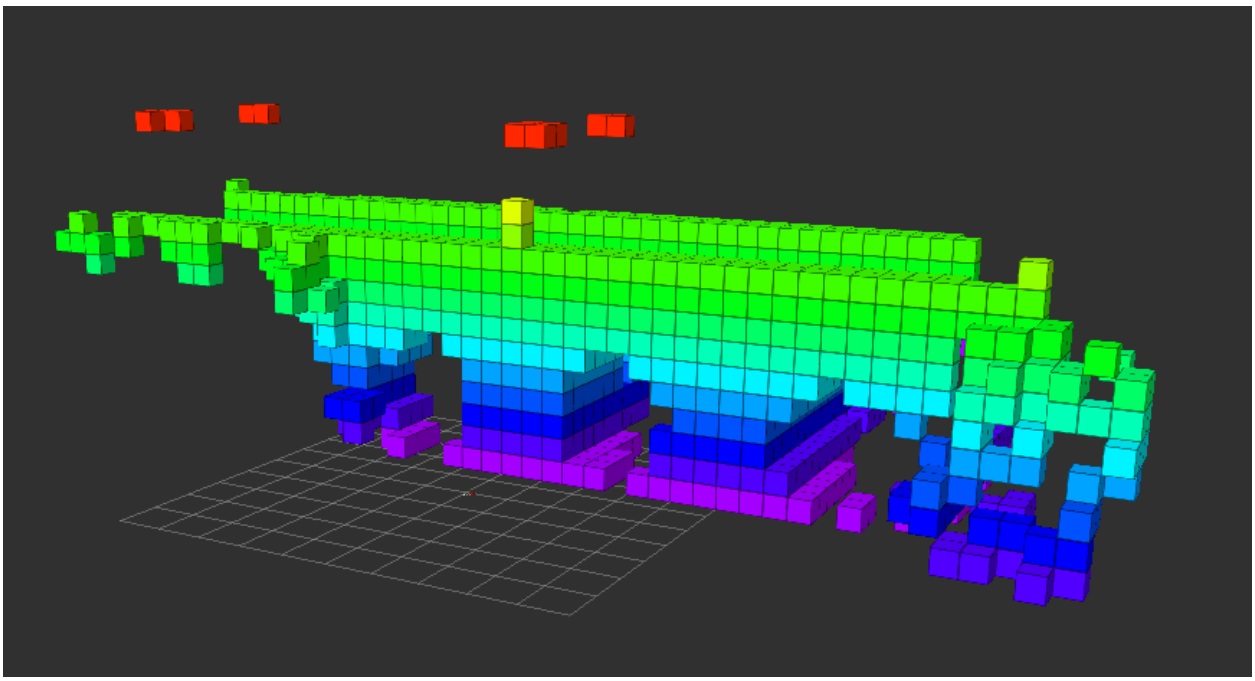


Figure 5.2: Trimmed voxel map of a bridge in Figure 5.1.

a human inspector. While UAVs are being used for bridge inspection, most operations are manual [36]. This requires skillful piloting in high wind conditions. Besides navigational challenges, visual inspection also requires careful planning of the viewpoints to obtain high-resolution images covering all points on the surface of the bridge. This is challenging to do manually for a large bridge where one may not always have visual-line-of-sight. To overcome these issues, solutions for assisting manual flights for targeted bridge inspection are developed [98]. In this chapter, we focus in particular on the issue of careful planning of inspection paths for the UAV to ensure high-resolution images for all points on the surface of the bridge.

A point is said to be inspected if at some point along the path of the UAV that point falls within the *viewing cone* and *viewing distance* of the camera attached to the UAV (defined in Section 5.2). The objective is to find a path of minimum length that guarantees inspection of all points on the surface of the bridge.

Inspection is closely related to coverage and exploration, problems that have been well-studied in the literature but, as we will show, are not necessarily the best approaches for inspection. If the UAV is given a 3D model of the environment, (including the bridge), then it can find a coverage path that covers all points on the bridge using an offline planner [54]. In practice, the UAV often does not have any prior model of the layout of the bridges. In many cases, even if a prior 3D model is available, it may not be accurate due to changes in the environment surrounding the bridge as well as structural changes that may have been made to the bridge. As a result, we focus on the more general problem of inspecting a bridge without any prior information about its geometry.

Frontier-based strategies [119] are typically used for exploring an initially unknown environment. A frontier is defined as the boundary between explored and unexplored regions. The strategies choose which amongst the set of frontiers to visit (and which path to follow to get

Figure 5.3: An example of $viewP$. The green box represents the face of a bridge voxel, the blue cone represents the viewing cone, and the red band on the cone represents the viewing distance.

to the chosen frontier) to help speed up exploration. The algorithm terminates when there are no more accessible unexplored regions. A bounding box can be placed around the bridge to restrict exploration when operating in an open environment. Exploring the bridge does not necessarily mean that the UAV will get inspection quality images. Instead, an inspection planner that can take into account the viewing cone and distance constraints may be more efficient. We present such a planner, termed *GTSP-Based Algorithm for Targeted Surface Bridge Inspection (GATSBI)*, and show that it outperforms the frontier-based exploration strategies in terms of efficiently inspecting the bridge (Section 5.4).

GATSBI consists of several modules: 3D occupancy grid mapping using LiDAR data, a semantic segmentation algorithm to find pixels that correspond to the surface of the bridge, a GTSP solver for finding inspection paths for the UAV, and a navigation algorithm for executing the planned path. We use off-the-shelf modules for occupancy grid mapping (Octomap [50]), solving GTSP (GLNS [102]), and point-to-point navigation (MoveIt [24]). The key algorithmic contribution is to show *how* to reduce the inspection problem to a GTSP instance and the full pipeline that outperforms baseline strategies. The technique takes into account overlapping viewpoints that can view the same parts of the bridge and simultaneously selects *where* to take images as well as *what order* to visit those locations. We also show when and how to replan as more information about the environment is obtained.

In summary, this chapter provides the following contributions:

- present an online algorithm, GATSBI, that plans paths to efficiently inspect bridges when no prior information about them is present;
- provide a completeness guarantee for a variant of the GATSBI algorithm;
- evaluate GATSBI (and the effects of various parameters within the algorithm) through numerous simulations in Gazebo using 3D models of bridges;

- compare GATSBI to a baseline frontier-based exploration algorithm using metrics that capture targeted inspection of bridges.

The rest of this chapter is organized as follows. In Section 5.1, we review previous work on inspection, coverage, and exploration. In Section 5.2, we present the problem formulation and in Section 5.3, we describe the solution. In Section 5.4, we present Gazebo simulation results. Finally, in Section 5.5 we conclude with a discussion of future work.

5.1 Related Work

As described earlier, frontier-based exploration is a widely-used method for 3D exploration of unknown environments [27, 84, 127]. Several variants of frontier exploration have been proposed, in particular, focusing on which amongst the frontiers to choose to visit next [28, 99]. Another popular approach is to model the exploration problem as that of information gathering and choosing a path (or a frontier) that maximizes the information gain [26, 92]. Additionally, there are Next-Best-View approaches [91] that, as the name suggests, plan the next-best location to take an image from in order to explore the environment. We refer the reader to a recent, comprehensive survey on multi-robot exploration by Quattrini-Li [67] that covers a variety of exploration strategies.

As we show in the simulations, generic exploration strategies are inefficient when performing targeted inspection. There has been work on designing inspection algorithms that plan paths that take into account the viewpoint considerations [49, 90, 95, 104]. When prior information is available, one can plan inspection paths carefully by taking into account the geometric model of the environment. Typically, the prior information is a low resolution version of the environment and the inspection path can be used to obtain high-resolution measurements of the environment [90, 95]. Unlike these works, we consider a scenario where no prior

information is given and the robot must plan on incrementally revealed information.

Bircher et al. [15] presented a receding horizon planner for both the exploration problem as well as the inspection problem. The two algorithms are similar and use a Rapidly-Exploring Random Tree to generate a set of candidate paths in the known, free space of the environment. Then, one amongst these paths is selected based on a criterion that values how much information a path gains about the environment. The planner is used in a receding horizon fashion (repeatedly invoked as new information is gained). We follow a similar approach; however, for the work by Bircher et al., the inspection surface is known a priori, where in the work in this chapter the only requirement is a portion of the bridge be seen at the start of inspection. The environment is not necessarily known; they test for both known and unknown. GATSBI has no prior knowledge about the environment and minimal knowledge of the target inspection surface.

Song et al. [104] recently proposed an online algorithm which consists of a high-level coverage planner and a low-level inspection planner. The low-level planner takes into account the viewpoint constraints and chooses a local path that gains additional information about the structure under inspection. The work proposed differentiates by guaranteeing a quality of inspection, not requiring a bounding box around the target infrastructure, and segmenting the infrastructure of interest from the environment, guaranteeing inspection of only the target infrastructure.

5.2 Problem Formulation

The overarching problem that we solve in this chapter can be described as follows.

Problem 1 Given a UAV with a 3D LiDAR and RGB camera, find a path to inspect every point on the bridge surface so as to minimize the total flight distance.

We consider the scenario where the geometric model of the bridge is unknown a priori. We assume that the UAV is initialized at a location where at least some part of the bridge is visible. In addition, we do not take into account the battery level of the UAV. Some of the previous work studies how to account for the UAV's battery level and supplement missions with an UGV for mobile recharging [122, 123]. An inspection path is planned for the part of the bridge that is visible. As more of the bridge is seen, the UAV replans. Problem 2 (defined shortly) focuses on finding a path for the UAV based on the partial information. We solve the larger Problem 1 by repeatedly solving Problem 2 as new information is gain.

We use a 3D semantic, occupancy grid to represent the model of the bridge that is built online. Each voxel in the occupancy grid will be assigned a semantic label. The label indicates whether the voxel is free $v_F \in V_F$; is occupied, part of the bridge, and previously inspected $v_{BI} \in V_{BI}$; is occupied, part of the bridge, but not yet inspected $v_{BN} \in V_{BN}$; and occupied but not a bridge voxel (i.e., obstacles) $v_O \in V_O$. The free voxels, V_F, represent free space where the UAV can fly. The inspected bridge voxels, V_{BI}, are bridge voxels that have already been inspected by the UAV. The non-inspected bridge voxels, V_{BN}, represent bridge voxels that have been added to the 3D occupancy map, but not yet inspected. Lastly, the obstacle voxels, V_O, represent occupied voxels that are not the bridge in the 3D occupancy map. The goal is to inspect all the voxels that correspond to the bridge surface, i.e., to ensure that $V_{BN} = \emptyset$.

A voxel $v_{BN} \in V_{BN}$ is said to be inspected if the UAV inspects at least one of its six faces. A face is said to be inspected if the center of that face falls within a cone given apex angle centered at the UAV camera and within a minimum and maximum range of the UAV camera. The apex angle represents the field-of-view of the camera that is rigidly attached to the UAV.

We first define a vector $\vec{co}$ from the center of the face of the voxel to be inspected to the center of the optical axis. Then we define a vector $\vec{oc}$ from the center of the optical axis to the center of the face of the voxel to be inspected. The angle between $\vec{co}$ and the face of the voxel has to be less than the viewing angle constraint. The angle created from $\vec{oc}$ and lens has to be less than the viewing angle constraint as well. Both of these angles have to be less than the viewing angle constraint to consider the face to be inspected. The viewing distance is comprised of a minimum and maximum distance that a bridge voxel should be inspected from to ensure quality images for inspection. An example of viewing constraints can be seen in Figure 5.3, where the green box represents a face of a bridge voxel, the blue cone represents the viewing cone, and the red band on the cone represents the viewing distance. For the rest of this chapter, we refer to viewing cone and distance as, $viewP$.

Problem 2 Given a 3D occupancy map consisting of four sets of voxels $(V_F, V_{BI}, V_{BN}, V_O)$, find a minimum length path that inspects every voxel in V_{BN}.

In the following section we show how to model this problem as a GTSP-instance.

5.3 The GATSBI Planner

In this section, we give an overview of the GATSBI algorithm. The full pipeline is given in Figure 5.4 and broadly consists of three modules: perception, planning, and execution. We describe each in details next.

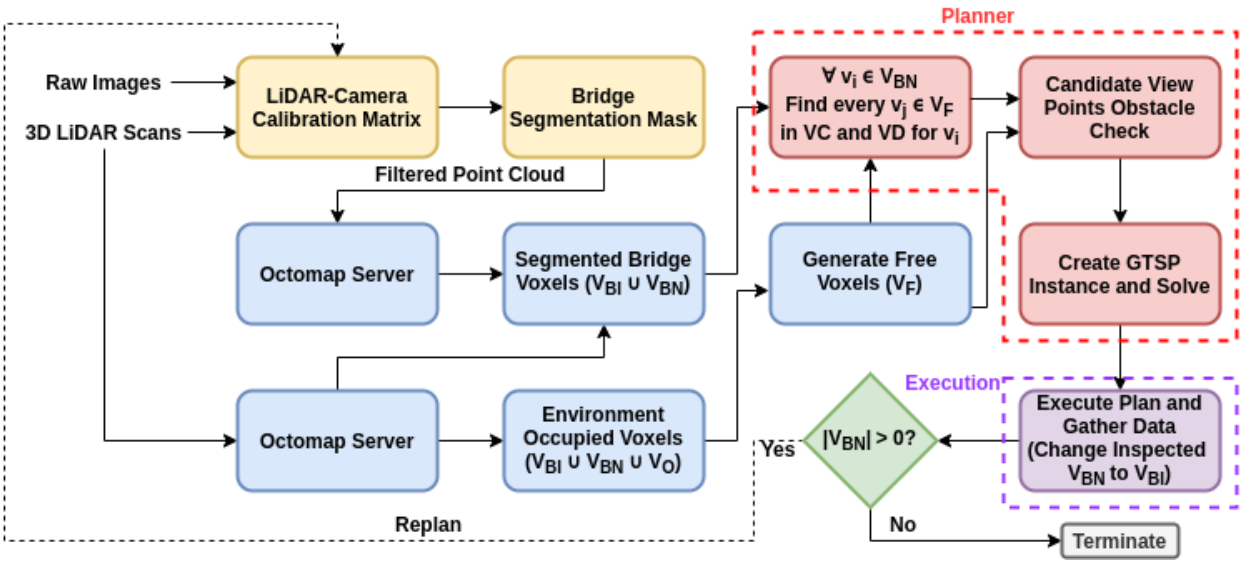


Figure 5.4: Flow diagram of GATSBI. The yellow blocks represent segmentation of bridge from environment, blue blocks represent voxel map generation, red blocks represent the planner, and the purple block represents execution of path.

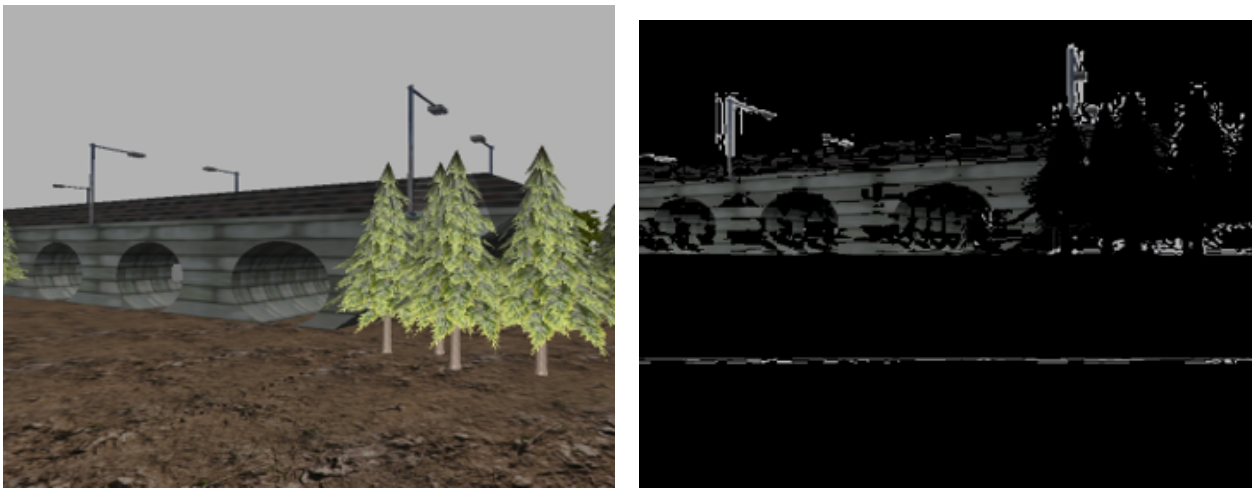


Figure 5.5: Left: Single image from the UAV RGB camera. Right: Binary mask created of the image in left.

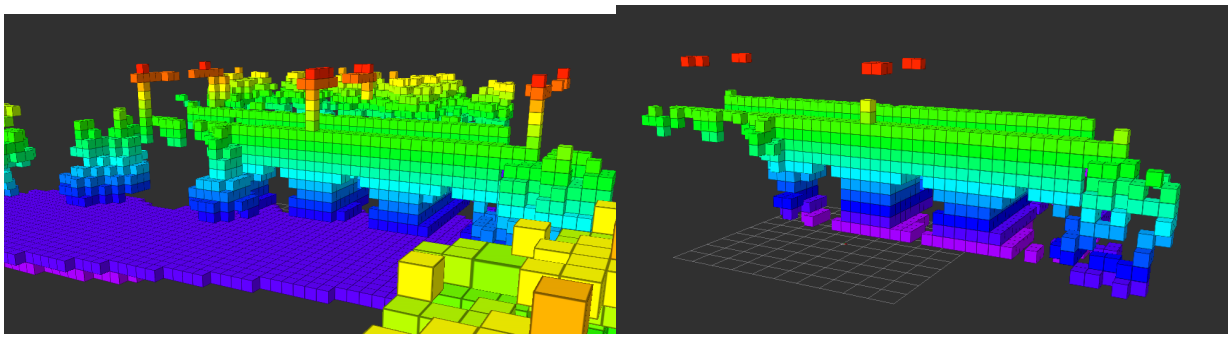


Figure 5.6: Left: Full voxel map containing $V_{BI} \cup V_{BN} \cup V_O$. Right: Segmented bridge voxel map containing $V_{BI} \cup V_{BN}$.

A Perception

GATSBI starts by segmenting the bridge from the environment using the LiDAR and RGB sensor information. Each incoming image (Figure 5.5-left) undergoes semantic segmentation to find pixels corresponding to the bridge.

Using the transformation matrix obtained by extrinsic LiDAR to RGB camera calibration, the LiDAR data is projected on to the image with a mask. Figure 5.5-right shows a mask from the color segmentation and Figure 5.7-row 2 shows masks from the neural network. The 3D points that lie on the segmented bridge are classified as bridge points. The point clouds obtained through segmentation using either the colored or neural net segmentors are then fed to OctoMap [50] to create a 3D occupancy grid. The output of this module is a set of voxels: free V_F, bridge V_{BI}, V_{BN}, and obstacle V_O. Only the segmented voxels (V_{BI}, V_{BN}) shown in Figure 5.6-right are used to plan inspection paths. The other voxels V_O and V_F are used only to plan collision-free paths and take into account viewing constraints. Either color-based segmentation or neural network-based segmentation of the bridge can be used for segmenting the bridge from the surrounding environment. Both methods are used in the simulation (Section 5.4). The neural network-based segmentation method has been previously shown to work robustly with real-world bridge images [14].

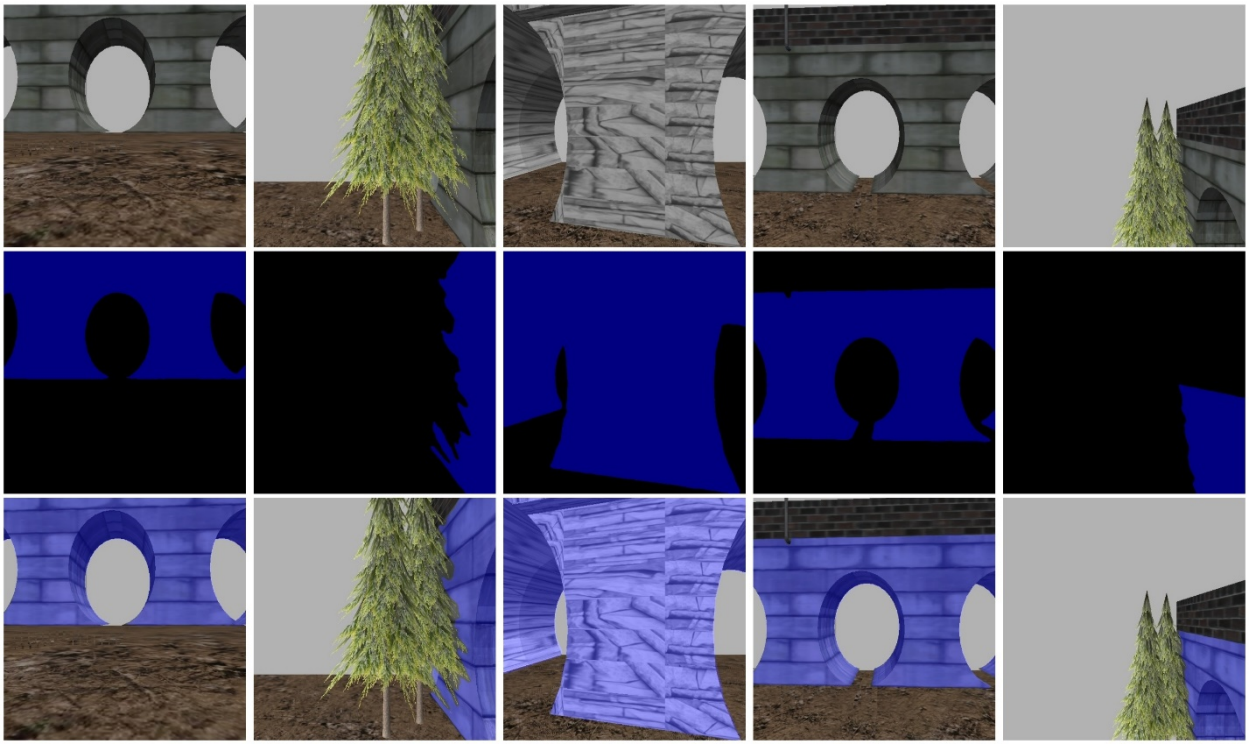


Figure 5.7: Example images from neural net. The first row is example images, second row the output mask, and thir row is the combined mask and image overlay.

B Planner

In order to inspect a bridge, all voxels in V_{BN} need to be inspected (as described in Section 5.2). The planner is repeatedly called over time. The V_{BI} set keeps track of voxels that have been inspected by the UAV previously. This avoids unnecessarily inspecting the same voxel more than once (some are unavoidable). Specifically, each voxel in V_{BN} must be viewed from some point on the path of the UAV within $viewP$. We formulate this problem as a GTSP instance.

The input to GTSP consists of a weighted graph where the vertices are partitioned into clusters. The objective is to find a minimum weight tour that visits at least one vertex in each cluster once. GTSP generalizes the Traveling Salesperson Problem and is NP-Hard [102]. In the following, we describe how we generate the input graph, G, and clusters for GTSP.

Each vertex in G corresponds to a candidate viewpoint. We check all pairs of $v_F \in V_F$ and $v_{BN} \in V_{BN}$ to see if v_F lies within $viewP$ of one of the faces of v_{BN}. If so, we add a vertex in the graph G corresponding to the pair v_F and v_{BN}. This vertex is also added to the cluster corresponding to each v_{BN}.

There can be multiple vertices in G corresponding to the same v_F, but each such voxel will belong to distinct clusters of G. If a v_F does not lie within $viewP$ of any $v_{BN} \in V_{BN}$, then it does not get added to the graph G.

All free voxels that can inspect the same v_{BN} will contribute one vertex each to the cluster corresponding to v_{BN}. The GTSP tour will ensure at least one viewpoint in this cluster is visited.

Next, we create an edge between every pair of vertices in G. The cost for each of these edges is the Euclidean distance between the two vertices. For vertices in the graph G that

correspond to the same v_F, the Euclidean distance will equal to zero. With the vertices, edges, and clusters, we create a GTSP instance and use the GTSP solver, GLNS [102], to find a path for the UAV. Note that GTSP will obtain a tour that returns back to the starting vertex (i.e., current position of the UAV). But, the UAV does not need to execute the last edge returning back to the start position of this tour.

C Execution

We use the navigation planner software MoveIt [24] to execute the GTSP path. Prior to moving from one point to the next, the UAV checks the distance from the current location of the UAV to the next vertex in the GTSP path. We use a point-to-point planner implemented within the MoveIt software based on the work done by Köse [62]. The point-to-point implementation uses an RRT connect solution for finding collision free paths for point-to-point navigation. For the rest of this chapter we use the terminology MoveIt to describe using the point-to-point planner. This distance is found using MoveIt [24] which takes into account the obstacles in the occupancy grid. If the difference between this MoveIt distance and the Euclidean distance is greater than DD (discrepancy distance), the UAV replans the GTSP path. The UAV replans as many times as needed to ensure that the first edge in the returned tour is within DD of the MoveIt distance. This is called a lazy evaluation of edge costs. Computing the MoveIt distance (which is a more accurate approximation of the actual travel distances) between every pair of vertices will be time consuming. By checking the discrepancy lazily, we find a tour quickly while also not executing any edge where the actual distance is significantly larger than the expected distance captured by the edge cost.

We keep track of each newly visited cluster during the path flight. Each of these newly visited clusters may correspond to a non-inspected bridge voxel. They are moved from set V_{BN} to

V_{BI} since they have now been inspected. We execute the plan until one of two conditions are met: either a time limit (RPT) elapses, or the UAV completes the path, whichever occurs first. We also record the raw sensor data during the flight. The MoveIt planner uses this new data for navigation. Once we complete navigation, we use the stored data to update the bridge inspected and non-inspected voxels and replan. Once V_{BN} is empty, the bridge is considered inspected and GATSBI will terminate.

D Completeness Guarantee

In this section, we will show that GATSBI is a complete algorithm, i.e., it will ensure that every voxel that can be inspected will eventually be inspected. This completeness guarantee holds when the environment satisfies some additional requirements that we state next. Nevertheless, the algorithm itself can be applied for environments that do not satisfy this requirement. The simulation results demonstrate this.

For completeness to hold, we require that all voxels of the bridge have an adjacent bridge voxel. That is, there are no isolated bridge voxels. We show the completeness by making some minor changes to how GATSBI works. These changes make it easier to prove the completeness; however, the actual implementation that we use in the experiments does not need to accommodate these changes. The changes are as follows. Each GTSP tour is fully executed before replanning. Inspecting a voxel requires the viewing cone's apex, angle to be 0°, and the center of the face of the voxel, to be inspected, to create a line which is perpendicular to the face of the voxel, being inspected. This requirement can easily be relaxed albeit at the cost of making the proof more tedious.

In general, GATSBI can inspect a voxel from any point that satisfies the viewing requirements. For the sake of easier analysis, we restrict the movement of the UAV to what will be

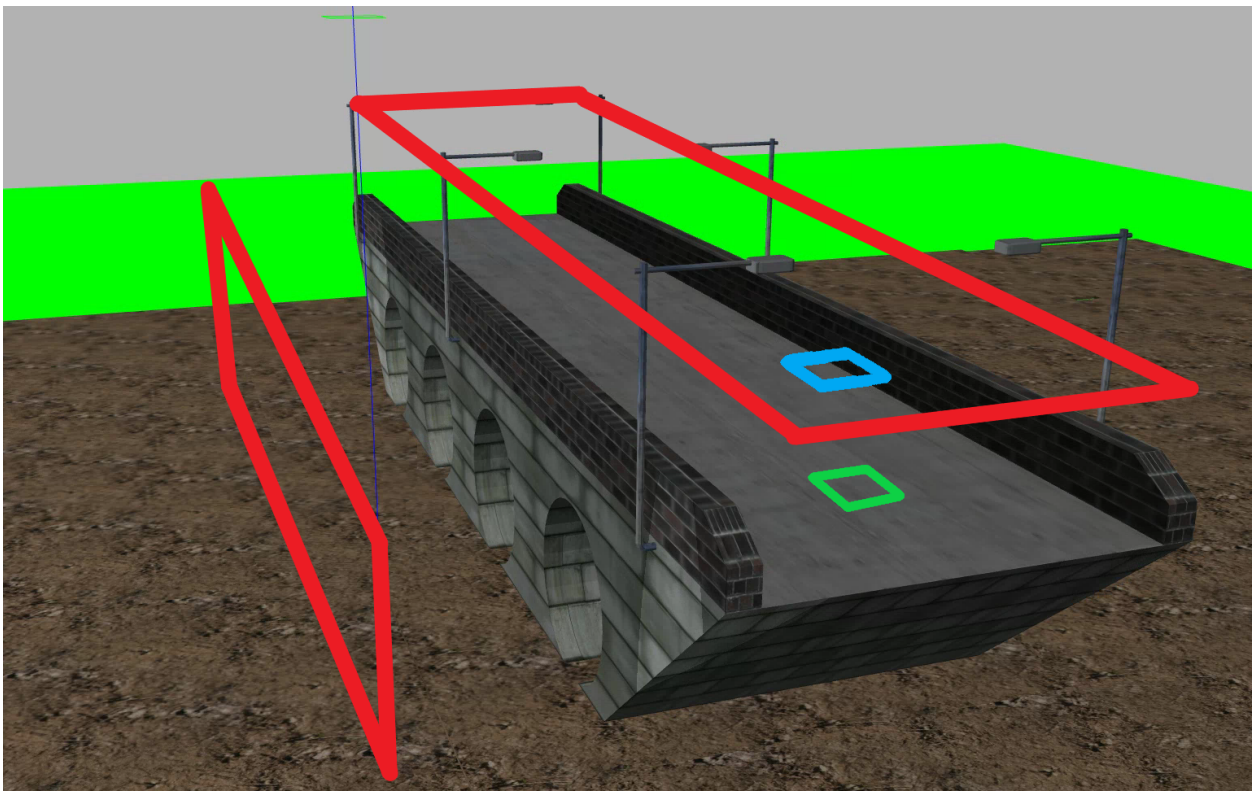


Figure 5.8: Two example VR shown as rectangles (red) on the top and side of the bridge with an example cell (blue) representing the bridge voxel (green).

called as Viewing Rectangles (VR). An example of a VR can be seen in Figure 5.8 as the red rectangles. These VRs allow me to inspect the bridge surface from a specified distance parallel to one of the three cardinal directions (i.e., parallel to the X, Y and Z axis of the voxel grid).

Each VR is a rectangle formed by a group of 2D cells projected from the faces of the bridge voxels at a specified distance. More specifically the set of VRs is, $P = P_1, P_2, ..., P_n$, where n represents the number of unique VRs needed to fully inspect the bridge. These VRs are created based on the viewing distance and number of voxels that represent the bridge.

Each VR consists of cells, c^i_j, where j is the identifier of the cell on the VR i. j ranges from $0, ..., m$, where m is the number of cells in the corresponding VR. We further require that the bridge is such that there can always be a set of VRs such that visiting some subset of cells in the VR is sufficient to ensure the full bridge surface is inspected. The cells in the VRs have the same side length as the voxels. Therefore, there is a one-to-one mapping between voxel faces that can be inspected and cells in the VRs. Figure 5.8 shows a cell, in blue, representing the bridge voxel to be inspected, in green. Visiting these cells in some VR is equivalent to inspecting the corresponding voxels.

With these requirements, whenever a UAV visits a cell that corresponds to a bridge voxel, then the UAV will also determine which of the four neighbors of this cell also correspond to bridge voxels. Therefore, *inspecting* a voxel is equivalent to visiting the corresponding cell, and *seeing* but not inspecting a voxel is equivalent to visiting a cell that corresponds to one of the neighbors of that voxel on the same VR. This holds trivially when the voxel resolution is less than the radius of the base of the viewing cone and the straight line between a cell and its corresponding voxel face is obstacle free. With the addition of VRs, the UAV has to know when and where to transition between different VRs. When a UAV should transition between VRs is when it has finished fully inspecting the current VR. To know where on the

new VR the UAV should transition, the cells on the current VR are projected onto the VRs that are adjacent to the current VR. Once these cells have been projected, the UAV finds the closest cell that represents a bridge voxel, is not on the current VR and then transitions to it. After the transition the UAV is now on a new un-inspected VR and since the VRs are known prior to inspection, the UAV is able to transition between VRs if and when necessary.

Theorem 5.1. *The modified version of GATSBI inspects all bridge voxels that are represented by VRs.*

Proof. We prove the theorem by contradiction. Suppose the modified version of GATSBI will not inspect all bridge voxels that are represented by VRs. Suppose there is a VR, P_i, with a cell c_j^i, that is not inspected on P_i. This means that c_j^i is not seen. If c_j^i was seen then it would be inspected since the modified version of GATSBI will find a tour that visits all the seen cells in a previous planning round. Since c_j^i is not seen then that means *all* neighbors of c_j^i have not been inspected. This previous step can be repeated until all possible values of j have been not seen on P_i. Since all values of j have not been seen on P_i then that means the UAV has not transitioned from any other VR, due to the capability that the UAV can transition between VRs. □

5.4 Simulations

In this section, we present simulation results to evaluate the performance of GATSBI. We first present a qualitative example of the inspection paths produced by GATSBI. Then, we study the effect of varying the parameters on the performance of the algorithm. We also evaluate the computational time for various subroutines within GATSBI. Finally, we compare GATSBI with the baseline frontier-based exploration.

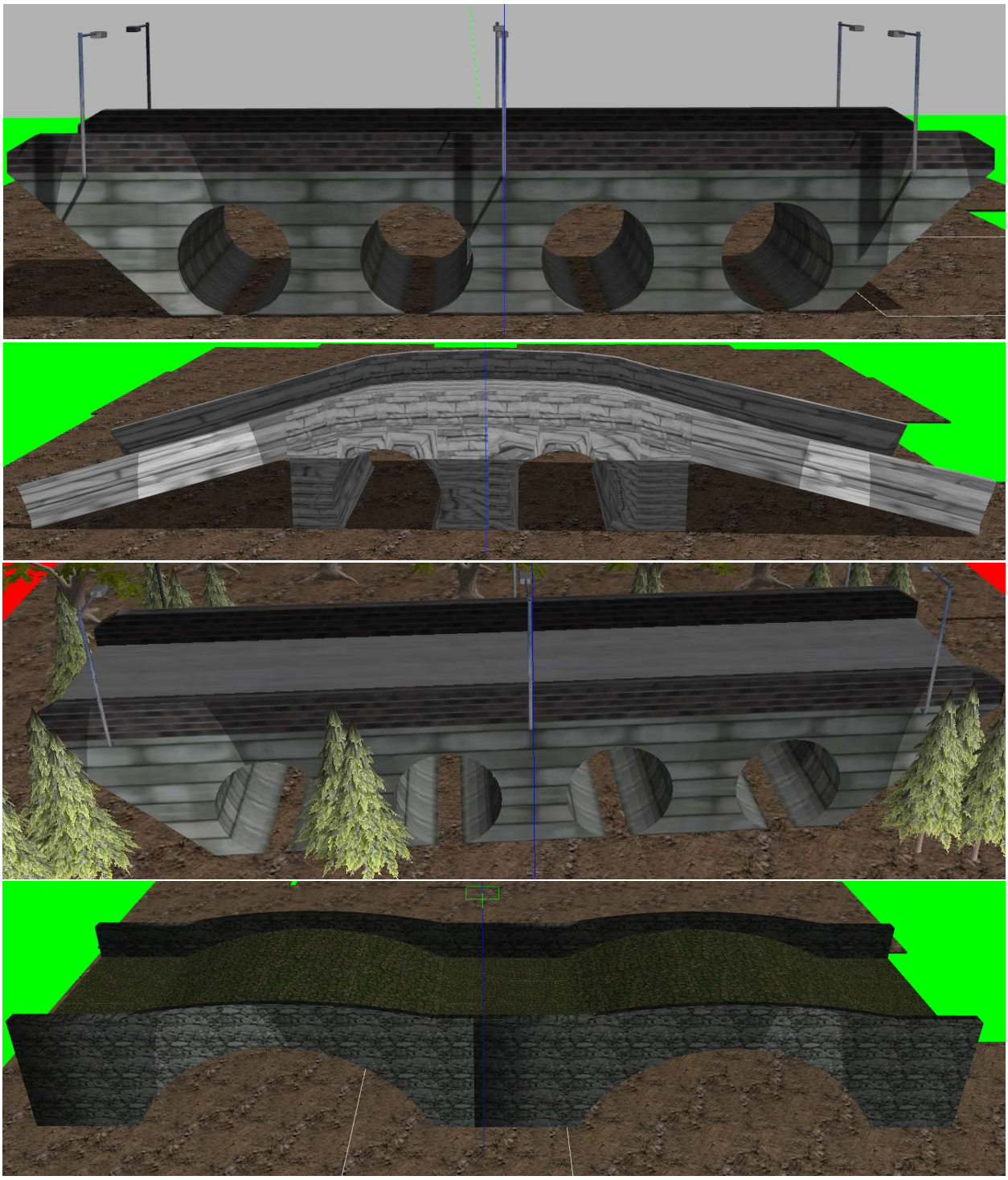


Figure 5.9: From top to bottom, bridges 1, 2, 3, and 4.

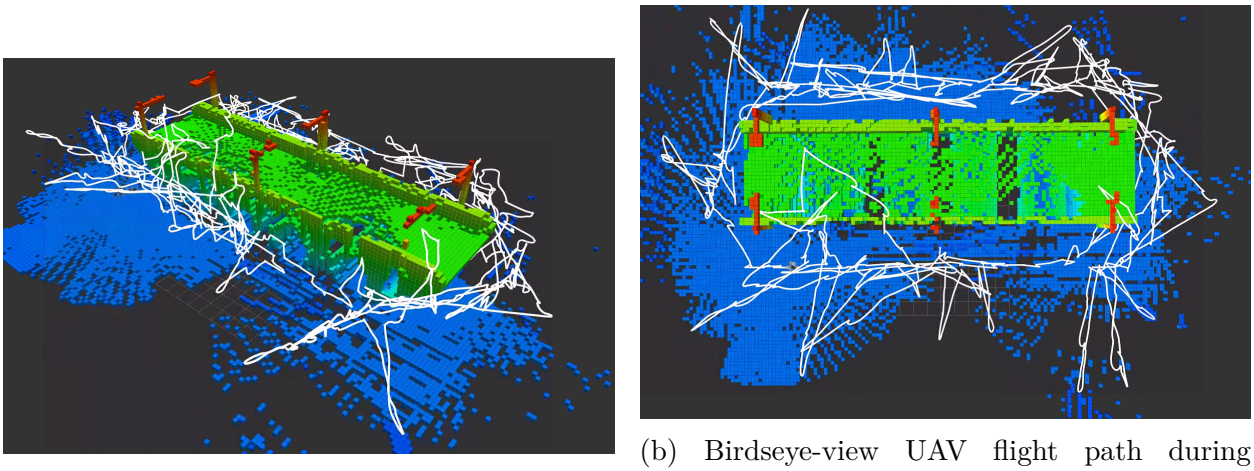
Setup We use Robot Operating System (ROS) Melodic on Ubuntu 18.04 and Gazebo to carry out the simulations. The simulated UAV is equipped with a Velodyne VLP-16 3D LiDAR, an ASUS Xtion Pro RGB camera, and GPS. The 3D LiDAR generates around 300,000 points/sec. It also has a 360° horizontal field of view with ±15° vertical field of view. The VLP-16 has a range of 100m [1]. The ASUS Xtion Pro has a resolution of 720x480. Its field of view is 58° horizontal, 45° vertical, and 70° diagonal, and the distance that it can be used for is from 0.8m to 3.5m [2]. The localization in simulations is perfect. In the real world, off-the-shelf Visual Inertial Odometry along with GPS for localization along the bridge surface can be used [55, 103].

For all experiments, we use a viewing cone with apex angle of 0° and a viewing distance between 2 to 5 meters. We set the viewing cone to the strictest possibility — a straight line from the camera. This is to ensure the highest quality of images captured for inspection. We set the viewing distance based on [36], where they suggest a minimum flight distance of 2 meters and a maximum of 5 meters to allow for the safe flight of the UAV as well as accurate detection of faults on a bridge.

A Qualitative Example

We evaluate GATSBI on four bridges shown in in Figure 5.9. All bridges, except bridge three, do not contain any other object in the environment except for the ground. The third bridge, on the other hand, has other distractor objects such as the trees. The first and the third bridge are the same model. The second and fourth bridges are qualitatively different than the first. Nevertheless, they are all representative of box girder bridges, as in the previous work [98].

First, we validate the neural network based segmentation algorithm. A neural network



(a) Angled UAV flight path during GATSBI.

(b) Birdseye-view UAV flight path during GATSBI.

Figure 5.10: UAV flight paths.

model from DeeplabV3+ with a ResNet101 backbone is implemented [18]. There were a total of 439 images that were obtained from simulation bridges and were annotated using labelme2016 [113]. The data was split into 10% validation and 90% training. The model that was used was fine-tuned from a pre-trained ResNet101 model from PyTorch's official pre-trained model paths. Training took place using a Tesla V100 GPU. The model was trained for 20 epochs with a validation f1-score of 97.2% and the jaccard-index of 94.7%. This model has also been previously shown to work on real world images. For the rest of the simulation, we use a simpler color based segmentation algorithm. However, the choice of the segmentation algorithm does not affect the main contribution of this chapter, which is the GATSBI planner.

Figure 5.10 shows the path followed by the UAV as given by GATSBI. Figure 5.11-left shows the final voxel map containing only the V_{BI} voxels. A sample RGB image obtained by the UAV along the path is also shown in Figure 5.11-right. These results are for bridge 1; the results for the other bridges are comparable. Next we present quantitative results on GATSBI.

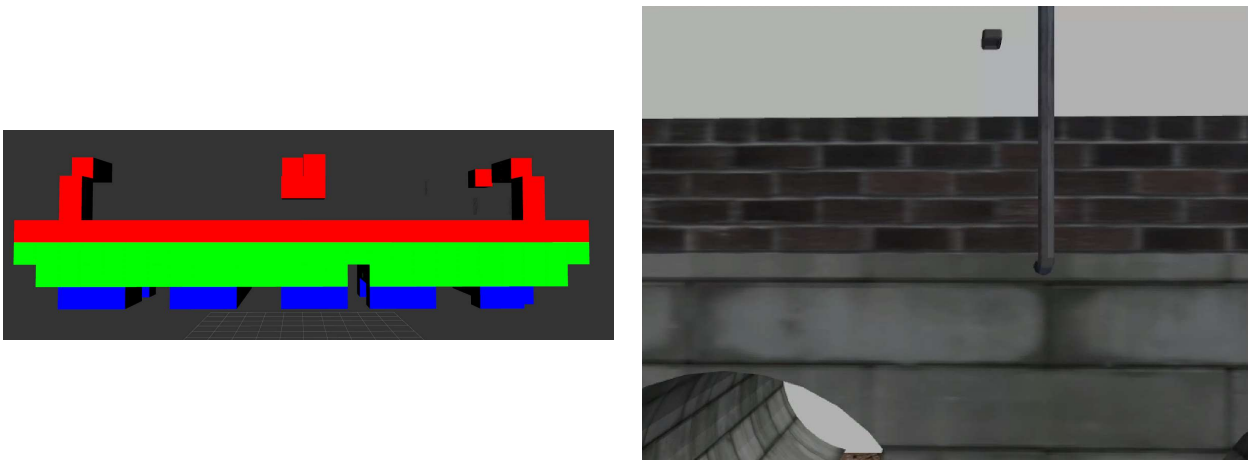


Figure 5.11: Outputs from GATSBI. Left: Example output of the voxel map for bridge 1. Right: Example inspection image of bridge 1.

B Effects of Varying the Parameters

Recall that the planner continues as long as $|V_{BN}| \neq 0$. Ideally, this would coincide when all voxels that *can* be inspected, *are* inspected. However, it is possible that the algorithm may terminate if the UAV finishes inspecting all bridge voxels it has seen so far, without having seen the entirety of the bridge. There are two parameters in GATSBI: *replanning trigger time*, RPT, and *distance discrepancy*, DD. In this section, we study the effect of varying these parameters on the performance of the algorithm.

Figure 5.12 shows the effect of varying RPT on the number of total bridge voxels inspected ($|V_{BI}|$ after the termination of the algorithm) and the total distance traveled by the UAV.

It is observed that the number of voxels inspected is not affected by RPT. The distance traveled seems to grow linearly with RPT except when $RPT = 150$ seconds. This is expected since larger RPT will let the UAV travel longer before taking into account new information and triggering a replan. A shorter RPT, on the other hand, will force the UAV to replan sooner, therefore leading to better optimized paths. We use $RPT = 90$ for the rest of the simulations.

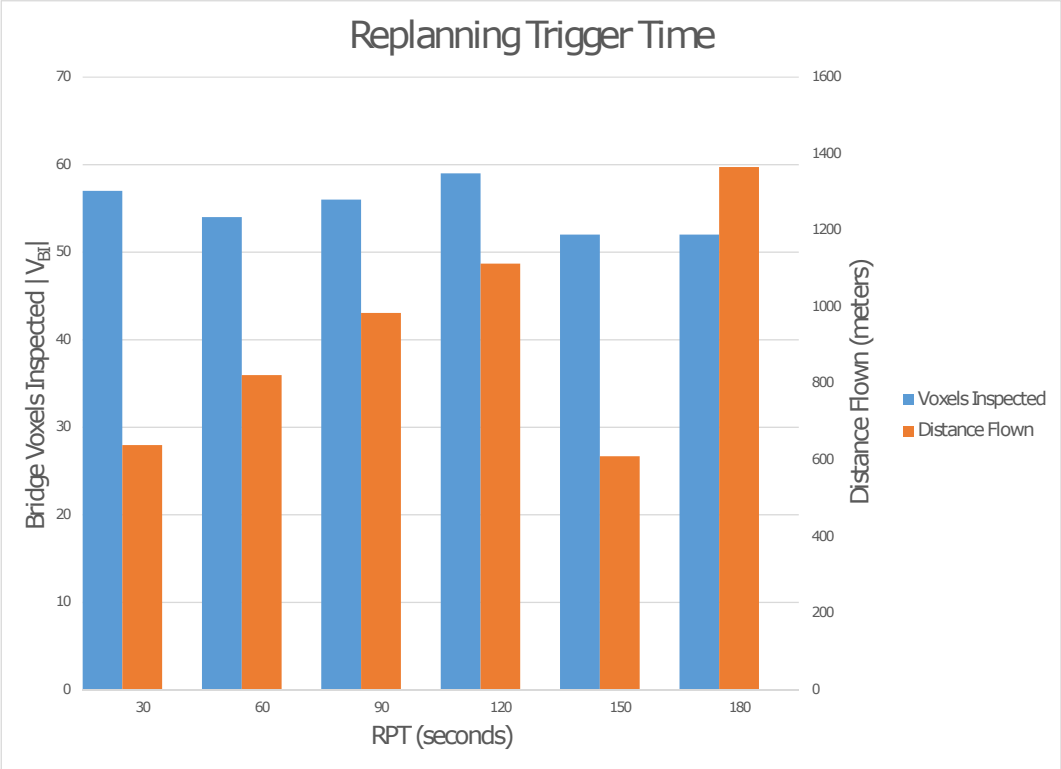


Figure 5.12: Evaluation of RPT vs voxels inspected and RPT vs flight distance. We use $RPT = 90$ for all other simulations.

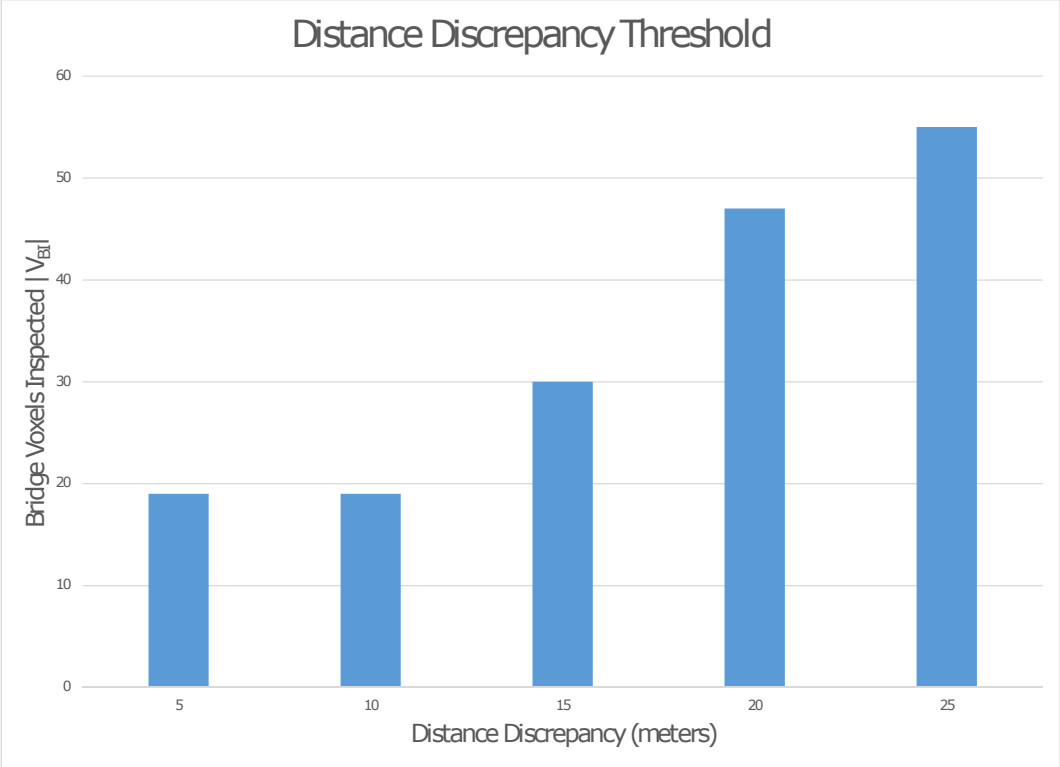


Figure 5.13: Evaluation of DD. We use $DD = 25$ for all other simulations.

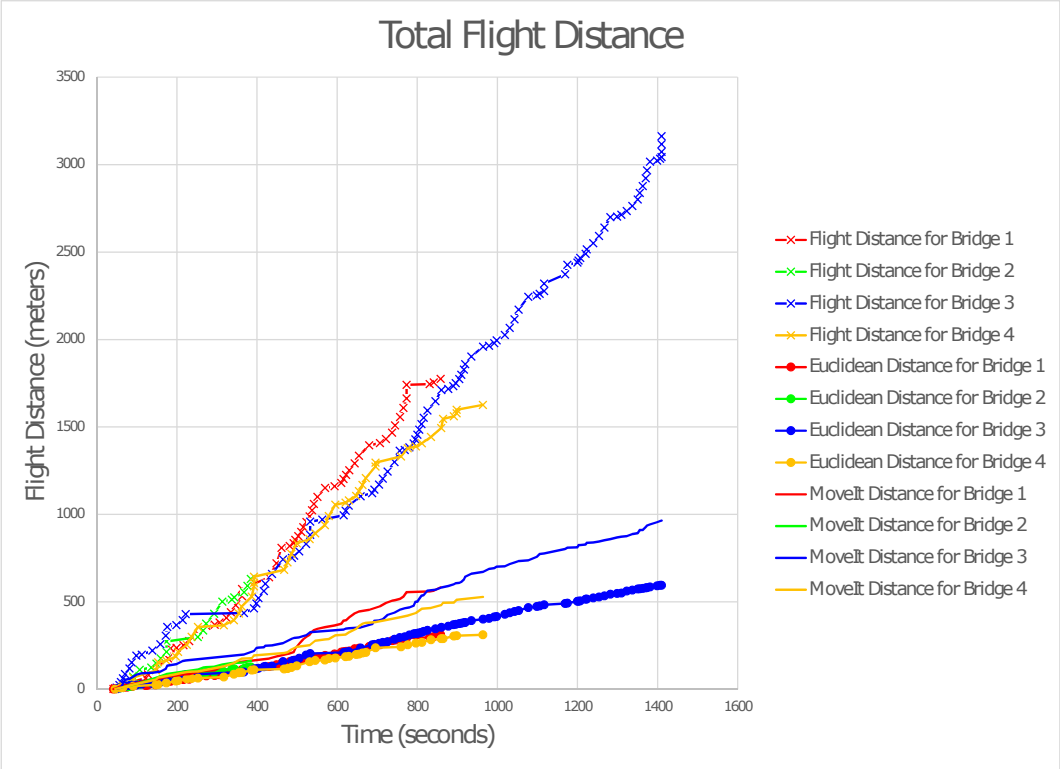


Figure 5.14: Travel distances for GATSBI.

Figure 5.13 shows the effect of the total number of bridge voxels inspected as we increase DD. A consistent increasing trend can be seen. Therefore, we choose $DD = 25$ when comparing the MoveIt distance and the Euclidean distance.

C Travel Distance

We also evaluate the total distance traveled by the UAV using GATSBI (Figure 5.14). We report three distances: flight distance, Euclidean distance, and MoveIt distance. The flight distance is computed by recording the actual position of the UAV during the inspection. The Euclidean distance is the straight-line distance between the current position of the UAV and the next point on the path returned by the GTSP solver. The MoveIt distance is the expected travel distance given by MoveIt to the next point.

It can be observed that the Euclidean and MoveIt distances are comparable to each other (and are bounded by the DD parameter). However, the actual flight distance is larger than the other two. This is likely because during the actual flight previously unseen obstacles may be seen and need to be avoided; thereby increasing the gap in the actual flight distance and the expected distance.

D Computational Time

We examine the time it takes for executing GATSBI. In Table 5.1, we report the average, minimum, maximum, and standard deviation (std) of the time for all bridges. We report three times: the time spent in the planner to create the GTSP instance (non-GTSP), the time taken to solve the GTSP instance (GTSP), and the flight time before the planner is called again (flight). It is seen that the time it takes GATSBI to perform segmentation and create a GTSP instance takes a maximum of 0.26 seconds. The GTSP solver takes a maximum of 6.15 seconds. Compared to the flight time (average of approximately 50s), the time taken by the planner is not significant. This suggests that GATSBI is not a bottleneck and is capable of running in real-time on UAVs that are executing 3D bridge inspection in unknown environments.

E Comparison with Baseline

We compare the performance of GATSBI with a baseline frontier-based exploration algorithm. Before comparison, we first found the best set of parameters that are used for the baseline method. The results are reported in the appendix A. We tested 3 main parameters. The first was the number of frontiers to visit before planning again. The second was whether to go to the closest frontier or a random frontier. Lastly, we tested the use of a grid

Bridge	Component	Average	Min.	Max.	STD
Bridge 1	Non-GTSP	0.06s	0.04s	0.090s	0.02s
Bridge 1	GTSP	5.84s	5.50s	6.00s	0.19s
Bridge 1	Flight	50.49s	7.45s	104.25s	26.50s
Bridge 2	Non-GTSP	0.08s	0.05s	0.10s	0.02s
Bridge 2	GTSP	6.05s	5.95s	6.15s	0.08s
Bridge 2	Flight	50.23s	13.22s	101.90s	23.75s
Bridge 3	Non-GTSP	0.13s	0.05s	0.26s	0.07s
Bridge 3	GTSP	5.14s	4.79s	5.45s	0.23s
Bridge 3	Flight	50.98s	6.82s	117.23s	27.39s
Bridge 4	Non-GTSP	0.08s	0.03s	0.11s	0.03s
Bridge 4	GTSP	5.99s	5.75s	6.24s	0.17s
Bridge 4	Flight	53.42s	4.87s	100.79s	26.49s

Table 5.1: Table showing the time taken for each part of GATSBI over multiple replans.

parameter. This grid parameter checked whether the UAV had previously visited a frontier in the same grid as the current target frontier. The parameter allowed any number of visits per grid cell from 1 to ∞ (standard frontier exploration). From the evaluations, visiting 10 frontiers before replanning, choosing random frontiers to visit, and having no grid cell visit restrictions is the best combination of parameters.

We compare GATSBI with the baseline method with these chosen parameters. Since the baseline method does not directly count the number of inspected voxels, we implement a package on top of the baseline to count the inspected voxels. This way we compare only the inspected voxels not the covered voxels. It is seen in Figure 5.15 that the proposed method does better than the baseline method when comparing the percentage of total bridge voxels inspected. We obtain this value by dividing the inspected bridge voxels ($|V_{BI}|$) upon termination of the algorithm by the total bridge voxels. The total number of bridge voxels are obtained by flying the UAV around the bridge and recording the number of bridge voxels. Note that in the third case, this number is greater than 100% because of some false positive bridge voxels reported in $|V_{BI}|$.

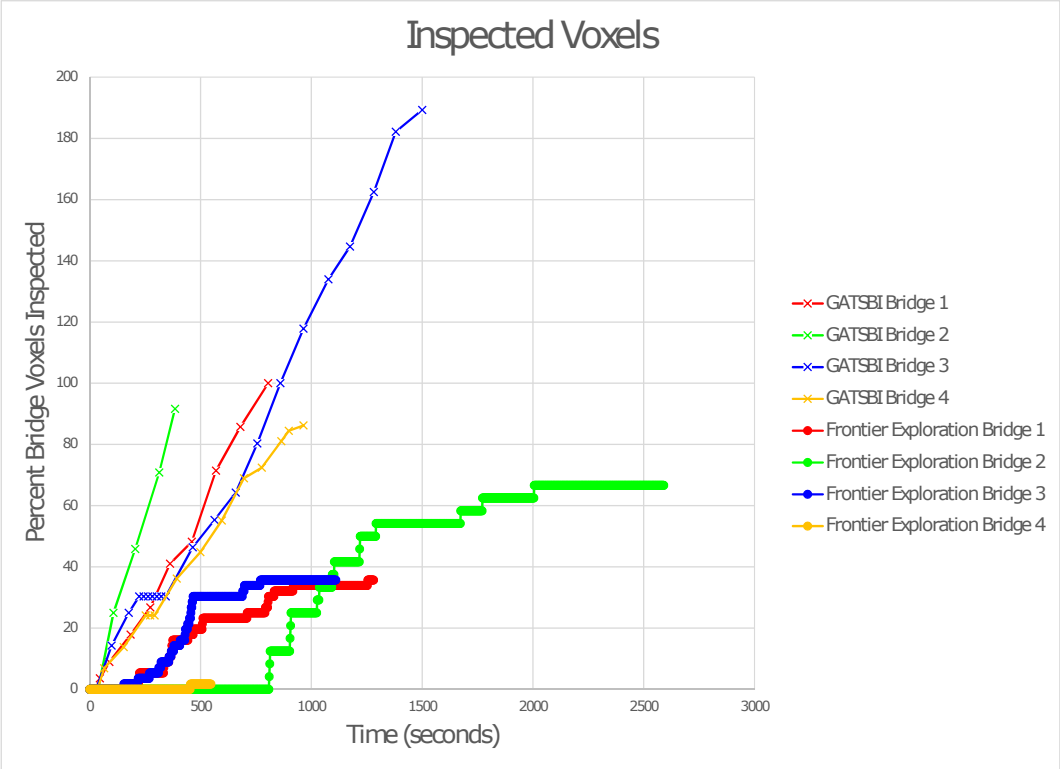


Figure 5.15: Percentage of voxels inspected over time for all bridges.

Nevertheless, it is observed that GATSBI achieves inspection of almost 100% voxels while the baseline only achieves a maximum of 35.7%. We also evaluate the efficiency in inspection by computing $|V_{BI}|/(|V_{BI}|+|V_{BN}|+|V_O|)$. This ratio captures the fraction of all bridge inspected voxels over occupied voxels. From Figure 5.16, it can be observed that GATSBI always has a higher fraction than the baseline. This validates the claim that GATSBI targets inspection of bridge surfaces instead of just covering the environment. We also see that GATSBI executes inspection faster than the baseline. Therefore, we justify the claim that GATSBI is more efficient in inspection compared to a frontier exploration algorithm.

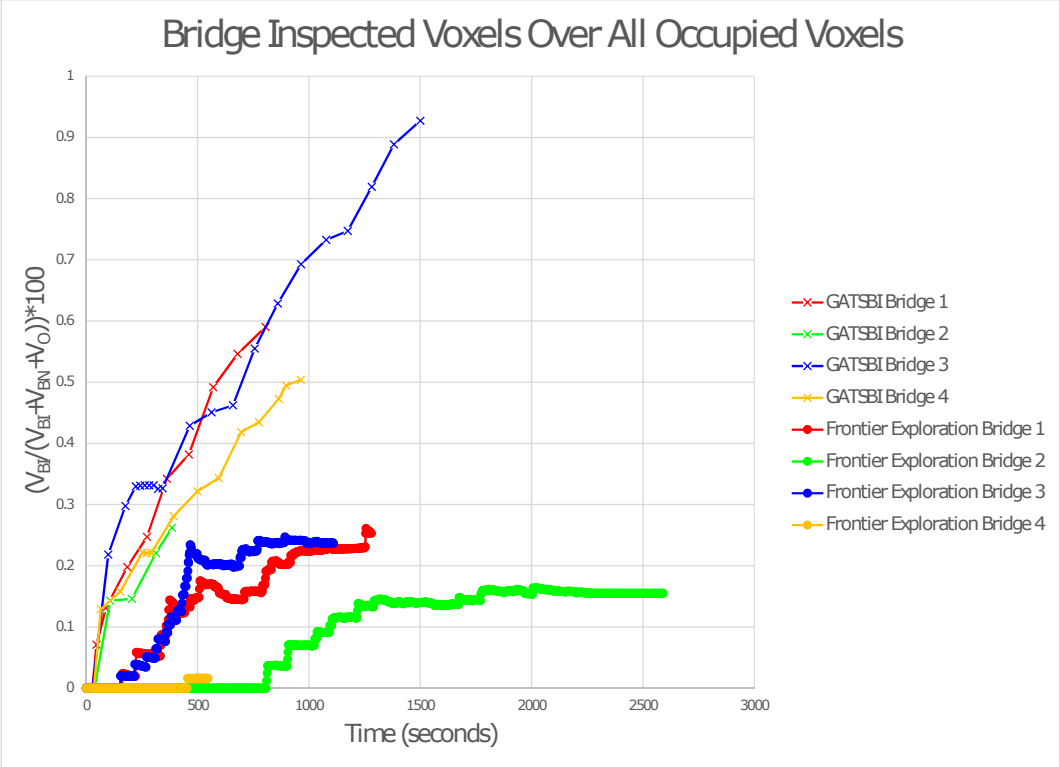


Figure 5.16: $|V_{BI}|/(|V_{BI}| + |V_{BN}| + |V_O|)$ over time for all bridges.

5.5 Conclusion

In this chapter, we present GATSBI, a method for 3D bridge inspection in an unknown environment. We evaluate the performance of the algorithm through Gazebo simulations with a UAV equipped with a 3D LiDAR and an RGB camera. The simulations show that GATSBI outperforms frontier-based exploration algorithms. In particular, we show that the algorithm is efficient in the sense that it targets voxels that are to be inspected, rather than simply exploring a volume. The simulations also demonstrate that the algorithm can run in real-time.

Chapter 6

Conclusion

The work presented in this dissertation solve the problems of a hybrid battery-limited UAV being used for persistent missions of point coverage and area coverage; as well as using a UAV for infrastructure inspection of bridges in an unknown and cluttered 3D environment. In Chapters 2 and 3 the focus was on overcoming the energy limitations of UAVs while covering sites of interest or 2D areas of interest on the ground plane. In Chapters 4 and 5 the focus was on overcoming the challenges associated with 3D inspection without any prior knowledge of the environment and with safety constraints when performing inspection in cluttered environments.

In Chapter 2 we started with the problem of point coverage on the ground plane. The UAV had a limited battery capacity which may prevent it from covering all points of interest in the environment. We presented a system which leveraged a UGV as a mobile charging stations and allowed the UAV to either charge in place or charge while being muled to the next take-off site. The work presented in this chapter minimized the total time it took a UAV to cover all points, including the flight, take-off, landing, and recharge times. Multiple simulations and proof-of-concept experiments were executed using this work and were presented at ICRA '18 [121] and later published in the Journal of Field Robotics [122].

Following Chapter 2, Chapter 3 extended the work to solve the problem of area coverage on the ground plane. In this chapter the hybrid UAV had to cover a set of boustrophedon cells while taking into consideration it's own battery life as well as what flight mode should be

used for coverage. An algorithm was presented that found the optimal recharging decisions: how much, where, and when to recharge the battery. In addition to these recharging decisions the algorithm found the optimal flight mode to use for coverage of the boustrophedon cells. The algorithm formulated models for multi-rotor UAVs as well as hybrid UAVs that could fly in either fixed-wing or VTOL modes. This was also evaluated through simulations and proof-of-concept experiments, and was presented at ICRA '19 [123] and is in submission at a journal.

In Chapter 4, we applied the 2D area coverage work to 3D surface area coverage. We solved the problem of 3D surface area coverage by reducing it to the 2D area coverage problem. Instead of having boustrophedon cells on the ground plane, they became planar surfaces on the bridge. Since not all cells were on the same plane the planner had to account for the 3D flight cost as well as how to transition between boustrophedon cells in the 3D environment. Low-level and high-level controllers were implemented that use LiDAR data to localized along the bridge surface to cover boustrophedon cells instead of GPS and compass data and decided on the order to cover boustrophedon cells and when to switch between low-level flight modes. This work was presented at ISER '19 [98] and a journal version is in revision.

Lastly in Chapter 5, a more general approach to 3D inspection was presented. In this chapter the work differentiated between coverage and inspection and focused on executing 3D inspection of bridges in an unknown and cluttered environment. An online planner was developed for the UAV to handle inspecting regions of interest. Our planner termed GATSBI plans online paths that are targeted towards inspecting all points on the surface of a bridge. The input to GATSBI is a RGB images and 3D occupancy grid map. We use semantic segmentation to segment the bridge voxels from the surrounding environment and then cluster candidate viewpoints for inspecting the bridge into a GTSP instance. This work is under review at IROS '21 [125].

In this dissertation the works presented focus on coverage in the 2D environment and then coverage and inspection in the 3D environment. Each chapter presents an increasingly harder problems that start with leveraging a UGV as a mobile charging stations, then working with hybrid UAVs to extend coverage of points to areas, and finally using a UAV in a complex and unknown environment to only inspect the structure of interest. Even with everything that is introduced in this dissertation there are still areas of future work. The main avenues for future work is improvements on the GATSBI algorithm presented in Chapter 5. We want to execute hardware experiments using a UAV with GATSBI running onboard. To do this we would need to implement the neural network segmentation, create a more robust method for exploring the bridge, and improve on the point to point flight planner. For future work we would also like to have GATSBI run in an assisted autonomy mode where an operator could take manual control of the UAV to fly unplanned routes and then return control to GATSBI to continue inspection of the structure of interest.

Bibliography

[1] Velodyne puck vlp-16 sensor | lidar | autonomoustuff. https://autonomoustuff.com/product/velodyne-puck-vlp-16/. (Accessed on 09/21/2020).

[2] Xtion pro | 3d sensor | asus global. https://www.asus.com/3D-Sensor/Xtion_PRO/. (Accessed on 09/22/2020).

[3] tahsinkose/hector-moveit: Hector quadrotor with moveit! motion planning framework. https://github.com/tahsinkose/hector-moveit. (Accessed on 10/16/2020).

[4] Moose ugv - clearpath robotics. https://clearpathrobotics.com/moose-ugv/, 10 2018. (Accessed on 11/17/2020).

[5] Federal Highway Administration. National Bridge Inspection Standards. *Federal Register*, 53:32611, 1988.

[6] AeroVironment. Av drone analytics - drones & data analytics with quantix & dss. https://www.avdroneanalytics.com/, 2020. (Accessed on 05/27/2020).

[7] AeroVironment. Quantix™ recon. https://www.avinc.com/tuas/quantix-recon, 2020. (Accessed on 05/27/2020).

[8] Shaimaa Ahmed, Amr Mohamed, Khaled Harras, Mohamed Kholief, and Saleh Mesbah. Energy efficient path planning techniques for uav-based systems with space discretization. In *Wireless Communications and Networking Conference (WCNC), 2016 IEEE*, pages 1–6. IEEE, 2016.

[9] Amazon. Amazon.com: Dji flame wheel f450 arf kit: Camera & photo. https://

www.amazon.com/DJI-Flame-Wheel-F450-ARF/dp/B00G4A2RBU, 2020. (Accessed on 09/19/2018).

[10] Anonymous. Skysense charging pad. (Web), December 2016. http://www.skysense.co/charging-pad/.

[11] David Applegate, ROBERT Bixby, Vasek Chvatal, and William Cook. Concorde tsp solver. *URL http://www. tsp. gatech. edu/concorde*, 2006.

[12] Ardupilot. Ardupilot firmware : /copter. http://firmware.ardupilot.org/Copter/, 2020. (Accessed on 09/19/2018).

[13] Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, 1998.

[14] Eric Loran Bianchi, Lynn Abbott, Pratap Tokekar, and Matt Hebdon. Coco-bridge: Common objects in context. dataset for structural detail detection of bridges. *ASCE Journal of Computing in Civil Engineering*, 2020. http://coco-bridge.com.

[15] Andreas Bircher, Mina Kamel, Kostas Alexis, Helen Oleynikova, and Roland Siegwart. Receding horizon path planning for 3d exploration and surface inspection. *Autonomous Robots*, 42(2):291–306, 2018.

[16] Avrim Blum, Shuchi Chawla, David R Karger, Terran Lane, Adam Meyerson, and Maria Minkoff. Approximation algorithms for orienteering and discounted-reward tsp. *SIAM Journal on Computing*, 37(2):653–670, 2007.

[17] Stanislav Bochkarev and Stephen L Smith. On minimizing turns in robot coverage path planning. In *Automation Science and Engineering (CASE), 2016 IEEE International Conference on*, pages 1237–1242. IEEE, 2016.

[18] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

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

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

[21] Howie Choset and Philippe Pignon. Coverage path planning: The boustrophedon decomposition. In *Proceedings of the International Conference on Field and Service Robotics*, pages 3–91, 1997.

[22] Clearpath. Husky ugv - outdoor field research robot by clearpath. https://www.clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/, 2020. (Accessed on 09/14/2018).

[23] Francesco Cocchioni, Adriano Mancini, and Sauro Longhi. Autonomous navigation, landing and recharge of a quadrotor using artificial vision. In *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, pages 418–429. IEEE, 2014.

[24] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. Reducing the barrier to entry of complex robotic software: a moveit! case study. *arXiv preprint arXiv:1404.3785*, 2014.

[25] Matthew Coombes, Wen-Hua Chen, and Cunjia Liu. Fixed wing uav survey coverage

path planning in wind for improving existing ground control station software. In *2018 37th Chinese Control Conference (CCC)*, pages 9820–9825. IEEE, 2018.

[26] Micah Corah, Cormac O'Meadhra, Kshitij Goel, and Nathan Michael. Communication-efficient planning and mapping for multi-robot exploration in large environments. *IEEE Robotics and Automation Letters*, 4(2):1715–1721, 2019.

[27] Daniel Louback da Silva Lubanco, Markus Pichler-Scheder, and Thomas Schlechter. A novel frontier-based exploration algorithm for mobile robots. In *2020 6th International Conference on Mechatronics and Robotics Engineering (ICMRE)*, pages 1–5. IEEE, 2020.

[28] Anna Dai, Sotiris Papatheodorou, Nils Funk, Dimos Tzoumanikas, and Stefan Leutenegger. Fast frontier-based information-driven autonomous exploration with an mav. *arXiv preprint arXiv:2002.04440*, 2020.

[29] Arun Das and Craig A Woolsey. Workspace modeling and path planning for truss structure inspection by unmanned aircraft. *Journal of Aerospace Information Systems*, 16(1):37–51, 2019.

[30] Jnaneshwar Das, Gareth Cross, Chao Qu, Anurag Makineni, Pratap Tokekar, Yash Mulgaonkar, and Vijay Kumar. Devices, systems, and methods for automated monitoring enabling precision agriculture. In *Proceedings of IEEE Conference on Automation Science and Engineering*, pages 462–469. IEEE, 2015. doi: 10.1109/CoASE.2015.7294123.

[31] Luca De Filippis, Giorgio Guglieri, and Fulvia Quagliotti. Path planning strategies for uavs in 3d environments. *Journal of Intelligent & Robotic Systems*, 65(1-4):247–264, 2012.

[32] Jason Derenick, Nathan Michael, and Vijay Kumar. Energy-aware coverage control with docking for robot teams. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3667–3672. IEEE, 2011.

[33] H. Dhami, K. Yu, T. Xu, Q. Zhu, K. Dhakal, J. Friel, S. Li, and P. Tokekar. Crop height and plot estimation for phenotyping from unmanned aerial vehicles using 3d lidar. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2643–2649, 2020. doi: 10.1109/IROS45743.2020.9341343.

[34] Carmelo Di Franco and Giorgio Buttazzo. Energy-aware coverage path planning of uavs. In *Autonomous Robot Systems and Competitions (ICARSC), 2015 IEEE International Conference on*, pages 111–117. IEEE, 2015.

[35] DJI. Spreading wings s900 - highly portable, powerful aerial system for the demanding filmmaker. https://www.dji.com/spreading-wings-s900, 2020. (Accessed on 06/10/2019).

[36] Sattar Dorafshan, Marc Maguire, Nathan V Hoffer, and Calvin Coopmans. Fatigue crack detection using unmanned aerial systems in under-bridge inspection. 2017.

[37] John A Dougherty. *Laser-Guided Autonomous Landing of a Quadrotor UAV on an Inclined Surface*. PhD thesis, The George Washington University, 2014.

[38] M. Dunbabin and L. Marques. Robots for environmental monitoring: Significant advancements and applications. *IEEE Robotics and Automation Magazine*, 19(1):24 –39, Mar 2012. ISSN 1070-9932. doi: 10.1109/MRA.2011.2181683.

[39] Gibrán Félix, Mario Siller, and Ernesto Navarro Alvarez. A fingerprinting indoor localization algorithm based deep learning. In *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 1006–1011. IEEE, 2016.

[40] Joseph Flynt. 10 best long flight time drones: Fantastic battery life - 3d insider. https://3dinsider.com/long-flight-time-drones/, November 2019. (Accessed on 09/21/2018).

[41] Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258–1276, 2013.

[42] Matthew N Gillins, Daniel T Gillins, and Christopher Parrish. Cost-effective bridge safety inspections using unmanned aircraft systems (uas). In *Geotechnical and Structural Engineering Congress 2016*, pages 1931–1940, 2016.

[43] Jeremy C Goldin. *Perching using a quadrotor with onboard sensing*. Utah State University, 2011.

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

[45] German Gramajo and Praveen Shankar. An efficient energy constraint based uav path planning for search and coverage. *International Journal of Aerospace Engineering*, 2017, 2017.

[46] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2015. URL http://www.gurobi.com.

[47] Bruno Herissé, Tarek Hamel, Robert Mahony, and François-Xavier Russotto. Landing a vtol unmanned aerial vehicle on a moving platform using optical flow. *IEEE Transactions on robotics*, 28(1):77–89, 2012.

[48] HEX. Pixhawk2.1 standard set - pixhawk2. http://www.proficnc.com/system-kits/31-pixhawk2-suite.html, 2020. (Accessed on 09/19/2018).

[49] Geoffrey A Hollinger, Brendan Englot, Franz S Hover, Urbashi Mitra, and Gaurav S Sukhatme. Active planning for underwater inspection and the benefit of adaptivity. *The International Journal of Robotics Research*, 32(1):3–18, 2013.

[50] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. doi: 10.1007/s10514-012-9321-0. URL http://octomap.github.com. Software available at http://octomap.github.com.

[51] Saghar Hosseini, Ran Dai, and Mehran Mesbahi. Optimal path planning and power allocation for a long endurance solar-powered uav. In *2013 American Control Conference*, pages 2588–2593. IEEE, 2013.

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

[53] IR-LOCK. Ir-lock | infrared tracking systems for drones and robot automation. https://irlock.com/, 2020. (Accessed on 09/14/2018).

[54] B. Kakillioglu, J. Wang, S. Velipasalar, A. Janani, and E. Koch. 3d sensor-based uav localization for bridge inspection. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 1926–1930, 2019.

[55] Burak Kakillioglu, Jiyang Wang, Senem Velipasalar, Alireza Janani, and Edward Koch. 3d sensor-based uav localization for bridge inspection. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 1926–1930. IEEE, 2019.

[56] Nare Karapetyan, Kelly Benson, Chris McKinney, Perouz Taslakian, and Ioannis Rekleitis. Efficient multi-robot coverage of a known environment. In *Proc. IEEE/RSJ*

International Conference on Intelligent Robots and Systems, Vancouver, BC, Canada, Sept. 2017.

[57] Nare Karapetyan, Jason Moulton, Jeremy S. Lewis, Alberto Quattrini Li, Jason M. O'Kane, and Ioannis Rekleitis. Multi-robot dubins coverage with autonomous surface vehicles. In *Proc. IEEE International Conference on Robotics and Automation*, 2018. To appear.

[58] F Paulo Kemper, Koji AO Suzuki, and James R Morrison. Uav consumable replenishment: design concepts for automated service stations. *Journal of Intelligent & Robotic Systems*, 61(1):369–397, 2011.

[59] Volodymyr Kharchenko and Dmitry Prusov. Analysis of unmanned aircraft systems application in the civil field. *Transport*, 27(3):335–343, 2012.

[60] Jonghoe Kim, Byung Duk Song, and James R Morrison. On the scheduling of systems of uavs and fuel service stations for long-term mission fulfillment. *Journal of Intelligent & Robotic Systems*, pages 1–13, 2013.

[61] Weiwei Kong, Dianle Zhou, Daibing Zhang, and Jianwei Zhang. Vision-based autonomous landing system for unmanned aerial vehicle: A survey. In *Multisensor Fusion and Information Integration for Intelligent Systems (MFI), 2014 International Conference on*, pages 1–8. IEEE, 2014.

[62] Tahsincan Köse. tahsinkose/hector-moveit: Hector quadrotor with moveit! motion planning framework. https://github.com/tahsinkose/hector-moveit. (Accessed on 03/03/2021).

[63] Gilbert Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European journal of operational research*, 59(3):345–358, 1992.

[64] Gilbert Laporte, Ardavan Asef-Vaziri, and Chelliah Sriskandarajah. Some applications of the generalized travelling salesman problem. *Journal of the Operational Research Society*, 47(12):1461–1467, 1996.

[65] Daewon Lee, Tyler Ryan, and H Jin Kim. Autonomous landing of a vtol uav on a moving platform using image-based visual servoing. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 971–976. IEEE, 2012.

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

[67] Alberto Quattrini Li. Exploration and mapping with groups of robots: Recent trends. *Current Robotics Reports*, pages 1–11, 2020.

[68] Teng Li, Chaoqun Wang, Max Q-H Meng, and Clarence W de Silva. Coverage sampling planner for uav-enabled environmental exploration and field mapping. *arXiv preprint arXiv:1907.05910*, 2019.

[69] Lantao Liu and Nathan Michael. Energy-aware aerial vehicle deployment via bipartite graph matching. In *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, pages 189–194. IEEE, 2014.

[70] Peter Liu, Albert Y Chen, Yin-Nan Huang, J Han, J Lai, S Kang, T Wu, M Wen, and M Tsai. A review of rotorcraft unmanned aerial vehicle (uav) developments and applications in civil engineering. *Smart Struct. Syst*, 13(6):1065–1094, 2014.

[71] Israel Lugo-Cárdenas, Gerardo Flores, Sergio Salazar, and Rogelio Lozano. Dubins path generation for a fixed wing uav. In *2014 International conference on unmanned aircraft systems (ICUAS)*, pages 339–346. IEEE, 2014.

[72] Parikshit Maini and PB Sujit. On cooperation between a fuel constrained uav and a refueling ugv for large scale mapping applications. In *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pages 1370–1377. IEEE, 2015.

[73] Mavros. mavros - ros wiki. http://wiki.ros.org/mavros, March 2018. (Accessed on 02/26/2018).

[74] Ivan Maza and Anibal Ollero. Multiple uav cooperative searching operation using polygon area decomposition and efficient coverage algorithms. In *Distributed Autonomous Robotic Systems 6*, pages 221–230. Springer, 2007.

[75] L. Meier, D. Honegger, and M. Pollefeys. PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, may 2015.

[76] Nathan Michael, Ethan Stump, and Kartik Mohta. Persistent surveillance with a team of mavs. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 2708–2714. IEEE, 2011.

[77] Derek Mitchell, Ellen A Cappo, and Nathan Michael. Persistent robot formation flight via online substitution. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4810–4815. IEEE, 2016.

[78] Farhan Mohammed, Ahmed Idries, Nader Mohamed, Jameela Al-Jaroodi, and Imad Jawhar. Uavs for smart cities: Opportunities and challenges. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 267–273. IEEE, 2014.

[79] Fabio Morbidi, Roel Cano, and David Lara. Minimum-energy path generation for a quadrotor uav. In *IEEE International Conference on Robotics and Automation*, 2016.

[80] Scott Morton, Ruben D'Sa, and Nikolaos Papanikolopoulos. Solar powered uav: Design and experiments. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 2460–2466. IEEE, 2015.

[81] Yash Mulgaonkar and Vijay Kumar. Autonomous charging to enable long-endurance missions for small aerial robots. In *SPIE Defense+ Security*, pages 90831S–90831S. International Society for Optics and Photonics, 2014.

[82] Huan Nguyen, Frank Mascarich, Tung Dang, and Kostas Alexis. Autonomous aerial robotic surveying and mapping with application to construction operations. *arXiv preprint arXiv:2005.04335*, 2020.

[83] Jorge Nieto, Emanuel Slawinski, Vicente Mut, and Bernardo Wagner. Online path planning based on rapidly-exploring random trees. In *2010 IEEE International Conference on Industrial Technology*, pages 1451–1456. IEEE, 2010.

[84] Farzad Niroui, Ben Sprenger, and Goldie Nejat. Robot exploration in unknown cluttered environments when dealing with uncertainty. In *2017 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*, pages 224–229. IEEE, 2017.

[85] Charles E Noon and James C Bean. An efficient transformation of the generalized traveling salesman problem. *INFOR*, 31(1):39, 1993.

[86] So-Ryeok Oh, Kaustubh Pathak, Sunil Kumar Agrawal, Hemanshu Roy Pota, and Matt Garrett. Autonomous helicopter landing on a moving platform using a tether. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 3960–3965. IEEE, 2005.

[87] Tolga Ozaslan, Shaojie Shen, Yash Mulgaonkar, Nathan Michael, and Vijay Kumar.

Inspection of penstocks and featureless tunnel-like environments using micro uavs. In *International Conference on Field and Service Robotics*, 2013.

[88] Fabio Pasqualetti, Antonio Franchi, and Francesco Bullo. On optimal cooperative patrolling. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 7153–7158. IEEE, 2010.

[89] Liam Paull, Carl Thibault, Amr Nagaty, Mae Seto, and Howard Li. Sensor-driven area coverage for an autonomous fixed-wing unmanned aerial vehicle. *IEEE transactions on cybernetics*, 44(9):1605–1618, 2013.

[90] Cheng Peng and Volkan Isler. Adaptive view planning for aerial 3d reconstruction. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2981–2987. IEEE, 2019.

[91] Richard Pito. A solution to the next best view problem for automated surface acquisition. *IEEE Transactions on pattern analysis and machine intelligence*, 21(10): 1016–1030, 1999.

[92] Aravind Preshant Premkumar, Kevin Yu, and Pratap Tokekar. Combining geometric and information-theoretic approaches for multi-robot exploration. *arXiv preprint arXiv:2004.06856*, 2020.

[93] PX4. Cube (pixhawk 2) · px4 user guide. https://docs.px4.io/en/flight_controller/pixhawk-2.html, March 2020. (Accessed on 09/14/2018).

[94] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[95] Mike Roberts, Debadeepta Dey, Anh Truong, Sudipta Sinha, Shital Shah, Ashish Kapoor, Pat Hanrahan, and Neel Joshi. Submodular trajectory optimization for aerial 3d scanning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5324–5333, 2017.

[96] Eduard Semsch, Michal Jakob, Dušan Pavlicek, and Michal Pechoucek. Autonomous uav surveillance in complex urban environments. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology-Volume 02*, pages 82–85. IEEE Computer Society, 2009.

[97] Hazim Shakhatreh, Abdallah Khreishah, Jacob Chakareski, Haythem Bany Salameh, and Issa Khalil. On the continuous coverage problem for a swarm of uavs. In *Sarnoff Symposium, 2016 IEEE 37th*, pages 130–135. IEEE, 2016.

[98] Prajwal Shanthakumar, Kevin Yu, Mandeep Singh, Jonah Orevillo, Eric Bianchi, Matthew Hebdon, and Pratap Tokekar. View planning and navigation algorithms for autonomous bridge inspection with uavs. In *International Symposium on Experimental Robotics (ISER)*, 2018.

[99] Shaojie Shen, Nathan Michael, and Vijay Kumar. Autonomous indoor 3d exploration with a micro-aerial vehicle. In *2012 IEEE international conference on robotics and automation*, pages 9–15. IEEE, 2012.

[100] Tristan Sherman, Joshua Tellez, Tristan Cady, Josue Herrera, Hana Haideri, Jimmy Lopez, Mitchell Caudle, Subodh Bhandari, and Daisy Tang. Cooperative search and rescue using autonomous unmanned aerial vehicles. In *2018 AIAA Information Systems-AIAA Infotech@ Aerospace*, page 1490. ARC, 2018.

[101] Aydin Sipahioglu, Gokhan Kirlik, Osman Parlaktuna, and Ahmet Yazici. Energy

constrained multi-robot sensor-based coverage path planning using capacitated arc routing approach. *Robotics and Autonomous Systems*, 58(5):529–538, 2010.

[102] S. L. Smith and F. Imeson. GLNS: An effective large neighborhood search heuristic for the generalized traveling salesman problem. *Computers & Operations Research*, 87: 1–19, 2017.

[103] Seungwon Song, Sungwook Jung, Hyungjin Kim, and Hyun Myung. A method for mapping and localization of quadrotors for inspection under bridges using camera and 3 d-lidar. In *Proceedings of the 7th Asia-Pacific Workshop on Structural Health Monitoring, Hong Kong SAR, PR China*, 2019.

[104] Soohwan Song, Daekyum Kim, and Sungho Jo. Online coverage and inspection planning for 3d modeling. *Autonomous Robots*, pages 1–20, 2020.

[105] Sparkfun. Scanse sweep - sen-14117 - sparkfun electronics. https://www.sparkfun.com/products/retired/14117, 2020. (Accessed on 06/01/2020).

[106] Kaarthik Sundar, Saravanan Venkatachalam, and Sivakumar Rathinam. Formulations and algorithms for the multiple depot, fuel-constrained, multiple vehicle routing problem. In *American Control Conference (ACC), 2016*, pages 6489–6494. IEEE, 2016.

[107] Yoonchang Sung and Pratap Tokekar. Algorithm for searching and tracking an unknown and varying number of mobile targets using a limited fov sensor. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 6246–6252. IEEE, 2017.

[108] Koji AO Suzuki, Paulo Kemper Filho, and James R Morrison. Automatic battery replacement system for uavs: Analysis and design. *Journal of Intelligent & Robotic Systems*, 65(1):563–586, 2012.

[109] Kurt A Swieringa, Clarence B Hanson, Johnhenri R Richardson, Jonathan D White, Zahid Hasan, Elizabeth Qian, and Anouck Girard. Autonomous battery swapping system for small-scale helicopters. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3335–3340. IEEE, 2010.

[110] Cornelius A Thiels, Johnathon M Aho, Scott P Zietlow, and Donald H Jenkins. Use of unmanned aerial vehicles for medical product transport. *Air medical journal*, 34(2): 104–108, 2015.

[111] Pratap Tokekar, Joshua Vander Hook, David Mulla, and Volkan Isler. Sensor planning for a symbiotic UAV and UGV system for precision agriculture. *IEEE Transactions on Robotics*, 2016.

[112] Tuna Toksoz, Joshua Redding, Matthew Michini, Bernard Michini, Jonathan P How, Matthew Vavrina, and John Vian. Automated battery swap and recharge to enable persistent uav missions. In *AIAA Infotech@ Aerospace Conference*, 2011.

[113] Kentaro Wada. labelme: Image Polygonal Annotation with Python. https://github.com/wkentaro/labelme, 2016.

[114] Sonia Waharte and Niki Trigoni. Supporting search and rescue operations with uavs. In *2010 International Conference on Emerging Security Technologies*, pages 142–147. IEEE, 2010.

[115] Minghan Wei and Volkan Isler. Coverage path planning under the energy constraint. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2018.

[116] Wingtra. Wingtraone – mapping drone for high-accuracy aerial surveys | wing-

tra. https://wingtra.com/mapping-drone-wingtraone/#intro, 2020. (Accessed on 05/27/2020).

[117] Jack Wrangham. Partnering with aerobotics - drone ag. https://droneag.farm/partnering-with-aerobotics/, February 2017. (Accessed on 06/17/2020).

[118] Anqi Xu, Chatavut Viriyasuthee, and Ioannis Rekleitis. Optimal complete terrain coverage using an unmanned aerial vehicle. In *2011 IEEE International conference on robotics and automation*, pages 2513–2519. IEEE, 2011.

[119] Brian Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'*, pages 146–151. IEEE, 1997.

[120] Z. Yao. Finding efficient robot path for the complete coverage of a known space. In *Proc. IEEE International Conference on Robotics and Automation*, 2006.

[121] Kevin Yu, Ashish Kumar Budhiraja, and Pratap Tokekar. Algorithms for routing of unmanned aerial vehicles with mobile recharging stations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–5. IEEE, 2018.

[122] Kevin Yu, Ashish Kumar Budhiraja, Spencer Buebel, and Pratap Tokekar. Algorithms and experiments on routing of unmanned aerial vehicles with mobile recharging stations. *Journal of Field Robotics*, 36(3):602–616, 2019.

[123] Kevin Yu, Jason M O'Kane, and Pratap Tokekar. Coverage of an environment using energy-constrained unmanned aerial vehicles. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3259–3265. IEEE, 2019.

[124] Kevin Yu, Prajwal Shanthakumar, Jonah Orevillo, Eric Bianchi, Matthew Hebdon, and Pratap Tokekar. View planning and navigation algorithms for autonomous bridge inspection with uavs. *arXiv preprint arXiv:1910.02786*, 2019.

[125] Kevin Yu, Harnaik Dhami, Kartik Madhira, and Pratap Tokekar. Gatsbi: An online gtsp-based algorithm for targeted surface bridge inspection. *arXiv preprint arXiv:2012.04803*, 2020.

[126] X. Yu, T. A. Roppel, and J. Y. Hung. An optimization approach for planning robotic field coverage. In *Proc. Annual Conference of the IEEE Industrial Electronics Society*, 2015.

[127] Cheng Zhu, Rong Ding, Mengxiang Lin, and Yuanyuan Wu. A 3d frontier-based exploration tool for mavs. In *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 348–352. IEEE, 2015.

Appendices

A Choosing Baseline Parameters

Frontier-based exploration explores an environment using frontiers, which are free voxels on the boundary of unknown and known space. We consider a general version of frontier exploration where the workspace is divided into a grid with a cell size. The grid size parameter was evaluated in previous work and the size we used, 13x13 cells, was found to have the best results [3]. For potential target frontiers, the algorithm checks whether the robot has previously explored the grid cell the target frontier is in. If it has visited it more than a given threshold, it ignores that frontier and moves onto the next one. This prevents the robot from exploring multiple frontiers in the same vicinity of frontiers that have already been explored.

We create a bounding box surrounding the bridge and restrict the path of the UAV to be inside this box. The parameters that may affect the outcome are the buffer volume surrounding the bridge, the number of frontiers before replanning, the number of voxels per grid cell for the grid-based exploration baseline, and choosing between random vs closest frontiers. In Figure A.1, it shows the effect of the buffer distance on the ability of the UAV to inspect bridge voxels. It is seen that a 5 meter buffer gives the best performance.

In Figure A.2, we show the effect of varying the number of frontiers that the UAV will visit before replanning. It is seen that the number of frontiers that the UAV goes to before replanning does not seem to directly affect the number of voxels inspected.

In Figure A.3 we evaluate the number of frontiers per grid cell before that grid cell is not visited again. For the grid-based exploration baseline, we settle on 5 frontiers per cell. It can be seen from the figure that 5 frontiers does the best.

The last parameter that we evaluate is whether to choose between random or closest frontier exploration. It can be seen from Figure A.4 that the closest method may provide results

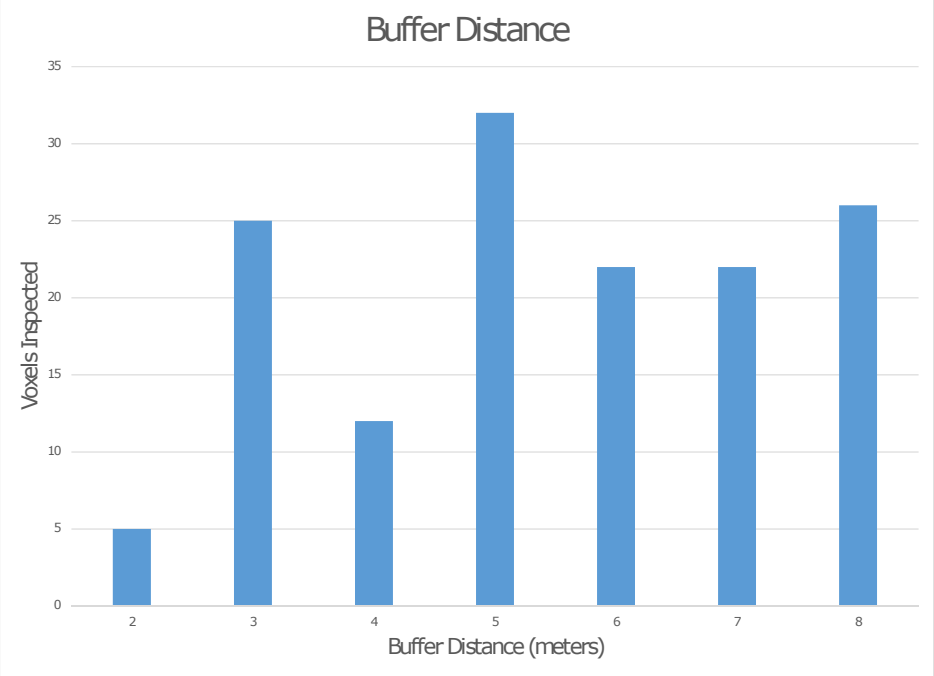


Figure A.1: Evaluating the change of buffer distance from the bridge.

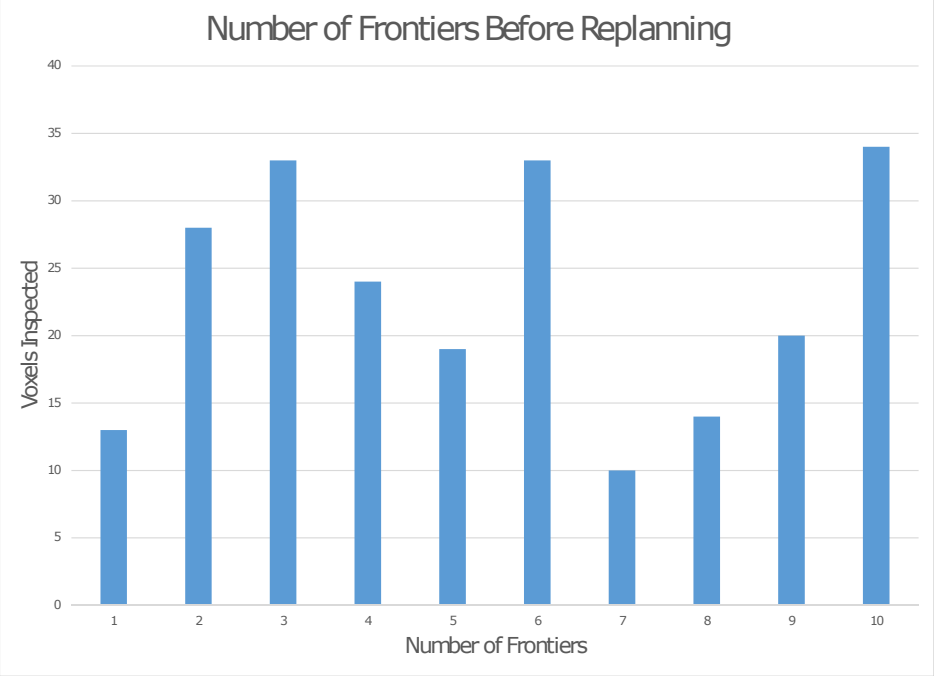


Figure A.2: Evaluation of the number of frontiers the UAV goes to before replanning.

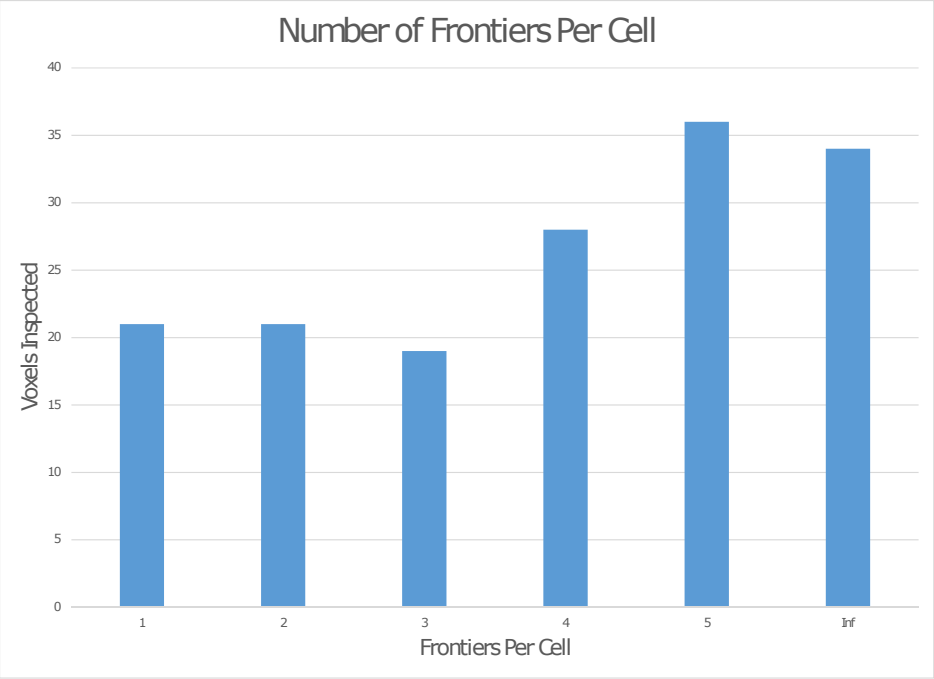


Figure A.3: Evaluation of the number of frontiers per grid if a grid was overlaid on the environment.

faster, but ends up terminating before the random method. Also, the random method does inspect more bridge voxels. Because of this, no conclusions can be drawn between if random or closest is the better method for bridge inspection and choose the random strategy for comparisons with the GATSBI algorithm.

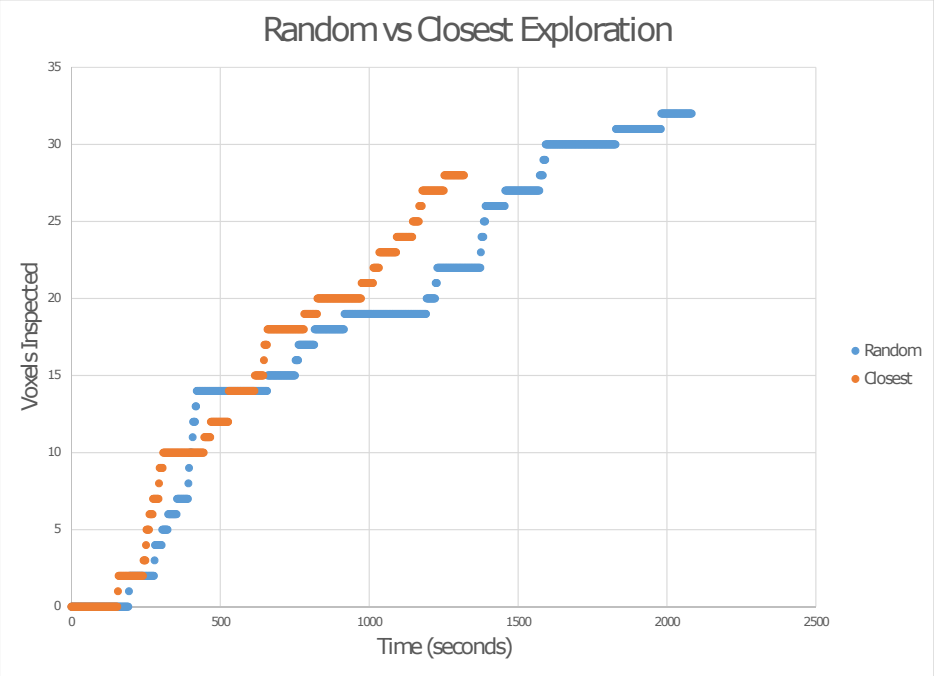


Figure A.4: Comparison of the two methods for choosing frontiers, random or closest.