Occlusion-Aware Motion Planning of Autonomous Robots in Cluttered and Unknown Environments

---

A Thesis

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the requirements for the Degree

Masters of Science (Computer Science)

by

Nicholas N Mohammad

May 2022

Approval Sheet

This thesis is submitted in partial fulfillment of the requirements for the degree of

Masters of Science (Computer Science)

Nicholas N Mohammad

This thesis has been read and approved by the Examining Committee:

Nicola Bezzo, Advisor

Sebastian Elbaum, Committee Chair

Madhur Behl

Accepted for the School of Engineering and Applied Science:

Jennifer L. West, Dean, School of Engineering and Applied Science

May 2022

Abstract

Navigation through unknown, cluttered environments is a fundamental and challenging task for autonomous vehicles as they must deal with a myriad of obstacle configurations typically unknown a priori. Challenges arise because obstacles of unknown shapes and dimensions can create occlusions limiting sensor field of view and leading to uncertainty in motion planning. There have been limited studies on the topic of occlusion-based motion planning, and they are primarily centered around safety assurance under uncertainty. However, taking advantage of properties of these occlusions can allow for fast, agile navigation. The work presented in this thesis builds around this concept and proposes a framework which leverages such occlusions to quickly navigate cluttered, unknown environments. The proposed framework presents a novel occlusion-aware motion planner which provides agile exploration by estimating gaps in point cloud data and shadows in the field of view to generate waypoints for navigation. We extend this planner to navigate quickly to a predefined goal in cluttered, unknown environments. Our scheme also proposes a breadcrumbing technique to save states of interest during exploration that can be exploited in future missions. For the latter aspect we focus primarily on the generation of the minimum number of breadcrumbs that will increase coverage and visibility of an explored environment. Extensive simulations and experiment results on an unmanned ground vehicle (UGV) are demonstrated to validate the framework, showing improvements over traditional state of the art frontier-based exploration methods.

Acknowledgements

First and foremost, I would like to thank my advisor Nicola Bezzo for his guidance and support throughout my time at the University of Virginia, and for making me feel at home in his lab. He helped push me to constantly strive for better, always asking the tough questions to ensure that I would produce a work that I could be proud of. I couldn't have asked for a better advisor.

I would like to express my gratitude and appreciation to the current and former members of the AMR Lab - Esen, Jacob, Lauren, Paul, Phil, Rahul, Shijie, and Will - for being so kind, supportive, and bringing me many laughs over the past year. I would also like to thank Dr. Aaron Bloomfield and Dr. Mark Floryan for their support during my time at UVA as well. While working with them over the pandemic, their kind words and enthusiasm were always appreciated and helped me get through those tough quarantine times.

Finally, I would like to thank my family for their constant love and support. I would like to especially thank my late grandmother, Barbara, for inspiring me to pursue my graduate studies, without her and the rest of my family, I wouldn't be the person I am today.

Contents

List of Figures

Chapter 1

Introduction

Autonomous exploration and mapping of unknown, cluttered environments is one of the most active areas of research in robotics with far reaching applications. Robots with these capabilities can be leveraged in inspections [25], surveillance [18], search and rescue [19], and even household applications like the common robotic vacuum cleaning. A critical component in each of these tasks is that maps of the environment are built either before or during exploration in order to assist with path planning and keep track of where the robot has been and has yet to go.

Figure 1.1: Autonomous systems are used for a variety of purposes including inspections, surveillance, search and rescue, mapping, and household applications.

In order to perform such exploration operations, the robot is equipped with range and vision sensors like

lidar, IR, sonar, and RGBd cameras that create point cloud data to help build maps of the environment and navigate around obstacles. These sensors' fields of view, however, are often occluded by objects in the environment, reducing the robot's visibility and thus reducing the speed at which exploration occurs, in many cases also limiting the complete coverage in complex environments. For example, consider a vehicle deployed in a dense, heavy forested area tasked to map the environment for search and rescue purposes or to find an item of interest, or deployed in a warehouse tasked to clean, or inspect and survey the area. Occlusions created by different types of obstacles surrounding the robot can create several unknowns restricting the possible reachable regions. If the robot could infer and extract information about the expected environment around such occlusions, it could increase its performance, in particular its ability to quickly cover the environment.

Current state-of-the-art approaches rely on using known free space within the map built at every step to generate waypoints, however, occlusions in cluttered environments greatly reduce the number of candidate waypoints that can be generated. We note here that such occlusions often hide accessible regions that, if leveraged, can increase the throughput of the robot during navigation tasks. This work builds around this idea, that is, *to design a method to reason about occlusions and infer and extract useful information about expected hidden traversable regions to generate waypoints that can lead to faster map coverage updates.* As an intuition, for example, large jumps in data between any consecutive point cloud data in a lidar measurement typically indicate the presence of a traversable area like a corridor. Similar considerations can be made for shadowed regions behind obstacles which can be assumed to be reachable by the vehicle.

Finally we note that exploration can be leveraged beyond just the purpose of building a map. As a robot explores an unknown environment, it may discover and save states of interest, for example regions of higher visibility, regions that were difficult to navigate and thus should be avoided in the future, or regions that contain some optimal properties. These states of interest can be thought of as virtual breadcrumbs that a robot drops as they get discovered, which robots in future missions may leverage to help complete tasks like search and rescue or inspections faster than they would be able to without them.

With these motivations in mind, we propose a novel exploration path planning method for autonomous robots in which frontier points are inferred and selected based on information extracted from sensor data patterns around occlusions. The three main cases considered at runtime are *gaps* in range sensor data, *shadows* in the sensor field of view, and *open space* frontier waypoints. By considering these cases we demonstrate that a robot can cover a completely cluttered environment faster than state of the art frontier-based exploration methods. We also propose a breadcrumbing method to facilitate exploitation of these explored environments. Specifically in this work we focus on breadcrumbing to increase visibility and propose a solution to solve the so called watchman tour problem, where a shortest route is found such that every point in the environment is visible from at least one point along the route [3]. Our approach solves this problem by utilizing a greedy

maximum coverage algorithm based on the geometry of saved sensor data at strategic positions during exploration.

The remainder of this thesis is organized as follows: in Chapter 2, we provide an overview of related work in both exploration path planning and watchman tour generation. In Chapter 3 we outline the mathematical notation used in this work. The proposed path planner and tour generation frameworks are presented in Chapters 4 and 5 respectively and are tested with extensive simulations and experiments in Chapter 6. Occlusion based agile navigation formulation and simulations are covered in Chapter 7. Lastly, we draw conclusions and discuss future work in Chapter 8.

1.1 Contribution

The contribution of this work is two-fold:

Occlusion-based Frontier Exploration — The first contribution is a robust occlusion-based frontier exploration path planner that enables a robot to quickly map an unknown, cluttered environment by observing patterns in pointcloud data. The planner detects occlusions in the environment by leveraging gaps and contiguous points in pointcloud data, then sends these occluded regions as waypoints to guide the exploration process. Given that the framework only requires pointcloud data to generate exploration waypoints, it is model agnostic and can be deployed in a wide variety of vehicle and sensing configurations. The approach is also demonstrated in both simulations and experiments to outperform traditional frontier exploration methods in terms of exploration speed.

Breadcrumbing and Approximate Watchman Tour Generation — The second contribution is a breadcrumbing technique which saves a tuple of the robot's pose and lidar data at spatially uniform points in the environment. These breadcrumbs are then fed into a greedy maximization algorithm to generate approximately shortest paths which observe an environment. Since the points used to generate the tours are breadcrumbs of actual states the vehicle reached during exploration, the approach inherently respects the configuration space of the vehicle. Furthermore, the framework is built in such a way that it can generalize to any pointcloud configuration since it takes into consideration important sensor features like maximum sensing range and FOV. In this way, our approach is robust enough to generate watchman tours regardless of the platform it's running on.

Chapter 2

Survey of Related Work and State of the Art in Autonomous Exploration

A large body of work is available on autonomous exploration of unknown environments. Often times a robot will be required to navigate and map an environment in which no a-priori information is given. All previous works attempt this problem through growing a known map by sending waypoints generated at runtime for the robot to reach. Over the years, three popular strategies have been developed for generating such waypoints: *frontier-based*, *sampling-based*, and *hybrid* navigation which uses both frontiers and sampling. This literature review is broken into four sections. First we will discuss frontier based exploration methods along with their advantages and disadvantages. We will then do the same for sampling and hybrid methods. Next we will review work in navigating within occluded environments, where the vehicle explicitly includes a notion of occluded sensor regions into its path planner. Finally, we end this chapter with a review of breadcrumbing and watchman tours, specifically within the context of exploration.

2.1 Frontier Based Navigation

The classic approach to exploration path planning is to utilize the notion of frontiers, which are intermediate regions between known and unknown spaces in a map. At each map update, a detection algorithm is run to detect new frontiers for use in the next planning step.

In the formative work of [29], frontiers are detected within an occupancy grid map, and the closest is targeted as the next waypoint for the vehicle, terminating once all reachable frontiers are visited. While this does achieve map coverage, naively choosing the closest frontier is not optimal, especially when the frontier is

not aligned in heading with a non-holonomic vehicle. Selecting a frontier behind the current position requires the vehicle to slow down or turn-in-place to reach the selected goal, reducing exploration speed. Furthermore, the global frontier detection scheme must be run each map update, which is a computationally expensive operation.

In order to increase exploration speed, [4] considers frontiers only within the camera frustum as candidates for the next waypoint to navigate towards. While this does achieve faster exploration, the expensive global frontier detection is still being performed. In order to avoid a global frontier update, [30], [1] take advantage of the fact that only the portion of the map within their camera frustum is update at each time step. Therefore, the frontiers within the frustum must be updated at each map update.

2.2 Sampling and Hybrid Based Navigation

In sampling based exploration frameworks, the goal is to sample poses which could grow the known regions of a map. A major benefit in using such sampling based approaches is that they remove the need to perform expensive global frontier detection algorithms at every map update and allow any desired utility definition to be used for pose selection [8]. [24] leverages the Next-Best-Views (NBV) model [5] to sample views which aim to maximize a utility function based on volumetric gain and time-of-flight for the vehicle. The NBV waypoints are generated by growing a Rapidly-exploring Random Tree (RRT) to sample positions and yaws from the configuration space of the vehicle [17].

In order to achieve fast exploration of underground tunnels, [9] generates a tree from the current robot configuration into the observed map by using motion primitives. The leafs of this tree are scored based on volumetric gain, time-of-flight to reach the end configuration and its alignment with the current heading of the vehicle. By allowing end configurations to have velocity, the approach allows fast, continuous exploration of an environment while selecting waypoints which respect the dynamics of the vehicle. However, this approach along with most NBV approaches incur large computational costs when performing the raycasting required to perform exploration gain calculations. Despite the success of sampling-based techniques, they can still get stuck in local minima (e.g., dead-ends). [28] provides a history-based technique to mitigate these limitations.

While many approaches use either frontier or sampling frameworks to solve the mapping problem, in recent years hybrid approaches have become increasingly popular. [7] updates frontiers only within the camera frustum and uses the properties of the *supereight* [27] mapping algorithm to quickly generate frontier clusters. They sample the frontiers uniformly and select a waypoint to navigate towards via raycasting-based entropy metrics. To mitigate quantization effects of [7], [1] use a similar approach but cluster frontiers using a bilateral filtering method leveraging permutohedral lattices.

Unlike the previous hybrid approaches which use NBV to select the next waypoint, [30] samples viewpoints in the vicinity of frontier clusters. This is done for each cluster to generate a list of local viewpoint candidates which maximize coverage. A travelling salesman algorithm is then performed to determine the global order to visit each frontier. A local refinement is then used via Dijkstra's to evaluate which viewpoint candidates in each cluster should be visited in order to minimize the overall flight time.

2.3 Occlusion Based Navigation

Despite the accomplished work in exploration path planning, less progress has been made on occlusion-based navigation. [15] discusses optimal path planning when obstacles in the environment are occluded by other closer or larger obstacles. In their approach, they use safety as the metric for optimality and use the known occluded regions of the space to inform their motion planner. Given that their safety conditions for optimality are intractable, they define several pseudo-optimal planners which successfully navigate their vehicle through unknown, occluded environments safely. However, their approach only considers the go-to-goal problem rather than leveraging occluded regions to determine where next to explore.

While [15] consider only static environments, [14] leverages an MPC-based approach for safe navigation in cluttered environments in which dynamic obstacles may be occluded by other obstacles. Their approach works by approximating the unknown area behind an obstacle and attempt to minimize it while also navigating to a goal. This work also leverages occluded regions to maximize safety in go-to-goal settings but does not consider the use of occlusions to inform exploration.

2.4 Breadcrumb Based Navigation

Breadcrumbing is a technique that can be used by a robot during missions to record regions of interest like difficult to explore areas, locations of landmarks of interest, or in the context of autonomous vacuum cleaners, areas which are more dirty than average. Beyond breadcrumbing during a mission, these breadcrumbs can be placed ahead of time to assist the robot in localization or communication. In all of these contexts, the breadcrumbs are assisting the robot during both current and future missions by recording important details for later use.

In regards to localization, [13] uses wireless beacons as breadcrumbs, allowing a UAV to localize itself during navigation to goal locations within an environment. However, the breadcrumbs must be placed manually by a human prior to navigation. [26] deploys wireless communication nodes autonomously in GPS denied underground environments by leveraging convolutional neural networks to assist in optimal placement

for maximum coverage, allowing nearly full network coverage with a minimal number of nodes. While both of these works leverage breadcrumbs to assist in navigation, they don't use them for the purpose of generating watchman tours – shortest paths which cover an entire environment.

In regards to exploration, [12] use a frontier based approach to generate at most $r+1$ vantage points which cover a 2D polygonal environment with r reflex angles. While the vantage points are shown to fully explore the environment, their approach is difficult to implement on physical systems using practical algorithms like occupancy grids for mapping. Furthermore, it is difficult to generalize to 3D.

To resolve the 3D problem, [18] uses their *City-CNN* approach to find vantage points for area coverage. They demonstrate that their CNN based approach can cover an entire environment with fewer points than previous techniques, but their approach is not easily transferable to vehicles with limited FOV sensors and they also don't generate watchman tours.

Chapter 3

Problem Formulation and Preliminaries

In this chapter, we formally introduce the problems covered in this thesis: map coverage and watchman tours. First, we discuss the notation used in this thesis to describe our framework. Next we provide a background on autonomous mapping and map coverage, in which a vehicle must map an a priori unknown environment quickly without colliding into obstacles. We follow this introduction with the formal problem definition. In the final section, we provide a similar introduction for watchman tours, where a shortest path which covers an environment must be found, and link it back to the problem of map coverage. We then finish the chapter by presenting the watchman problem definition.

3.1 Notation

In this thesis we denote vectors with bold, italic letters (e.g., $\boldsymbol{x}$) and sets with upper case greek and calligraphic letters (e.g, Ω and $\mathcal{S}$). We use the indicator $\mathbb{I}_{\mathcal{S}}(b(s))$ to denote if there exists $s \in \mathcal{S}$ which satisfies the boolean condition b. Given a set $\mathcal{S}$ denoting a region of space, we use $|\mathcal{S}|$ to denote the area of $\mathcal{S}$ and $\#\mathcal{S}$ to denote the cardinality. We reserve the set $\mathcal{M}$ for denoting the occupancy grid of a traversable region in the world $\mathcal{W}$. A $\wedge$ symbol on top of a variable represents its estimated value (e.g. $\hat{\boldsymbol{x}}$). $||\cdot||$ represents the Euclidean norm and the robot state is denoted by $\boldsymbol{x} = (\boldsymbol{p}_u, \theta)$ where $\boldsymbol{p}_u = [x, y]^T \in \mathbb{R}^2$ refers to the position. $\boldsymbol{z}$ denotes the sampled point cloud distances and is indexed as $\boldsymbol{z}_i$. The corresponding point to $\boldsymbol{z}_i$ is referred to as $\boldsymbol{p}_i \in \mathbb{R}^2$. poly($\boldsymbol{z}$) takes in point cloud data and returns the polygon defined by those points. Lastly, abs($\cdot$) denotes the absolute value.

3.2 Problem Formulations

3.2.1 Map Coverage

One of the most important problems in robotics is to enable autonomous coverage of known and unknown environments. What makes this problem challenging for unknown environments especially is that a robot must deal with a wide variety and potentially large number of obstacles. Because of this, defining an optimal path through these environments is often not possible, or at best computationally intractable in the case of known environments.

In general, a UGV will have several included sensors to assist with localization, like wheel encoders and an inertial measurement unit (IMU). However, in order to map an environment, an additional range sensor is required which collects information about the vehicle's surroundings. Typically this type of sensor returns a point cloud, a collection of distance measurements to obstacles near the robot, by using camera or laser based techniques. Common examples shown in Fig. 3.1 are RGBD sensors like Microsoft's Kinect, stereo cameras, and lidar.

(a) RGBD Sensor (b) Stereo Camera (c) Lidar

Figure 3.1: Examples of point cloud sensors

Using these point cloud sensors allows the robot to create occupancy grid maps which contain information about the location of obstacles in an environment. Furthermore, the sensors can be leveraged to localize the vehicle within this map by using a particle filter. By using both algorithms, Simultaneous Localization and Mapping (SLAM) can be leveraged to create reasonably high fidelity maps of an environment in the presence of sensor noise and environmental disturbances.

However, these algorithms alone cannot map an environment without the use of an underlying path planner to control the robot to a goal pose. Fortunately, given a map of what the robot has explored so far,

the path planner can quickly generate safe trajectories to waypoints which encourage the vehicle to explore. With the combination of SLAM and path planning, a UGV becomes capable of performing exploration within unknown environments, as shown in Fig. 3.2. The problem then becomes how to generate these waypoints such that, when the vehicle has navigated to all waypoints successfully, the entire reachable environment has been mapped. In chapter 4, we discuss in depth our waypoint selection algorithm for fast and complete map coverage.

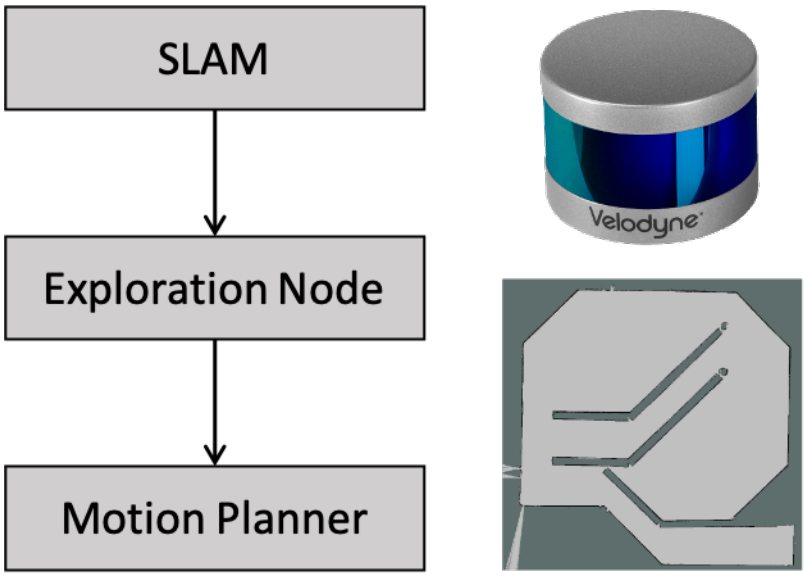


Figure 3.2: Exploration System Architecture

From the above discussion of the autonomous exploration setup, we can formulate the problem as follows. Given an a priori unknown cluttered environment $\mathcal{W}$ with N obstacles in which a region $\mathcal{M} \subseteq \mathcal{W}$ is traversable by a robot, the problem of map coverage can be defined as a multi objective optimization problem to find a policy which completely covers $\mathcal{M}$ while also trying to minimize the time needed to cover and map $\mathcal{M}$. Formally, let $\boldsymbol{x}$ denote the pose state of the robot (e.g., $\boldsymbol{x} = (x, y, \theta)$, the positions and angles for a ground vehicle configuration) and $M(t)$ be the map covered by the robot at time t. The fast map coverage problem is then defined as finding the control policy $\mathcal{U}(t)$ to minimize the total time T to cover the entire space such that the following constraint is satisfied:

$$\left| \mathcal{M} \setminus \bigcup_{t=t_0}^{T} M(t) \right| < \epsilon$$

where ϵ is an arbitrarily small threshold and $\setminus$ is the set-minus operation. The total covered map after this operation is $M_T = \bigcup_{t=t_0}^{T} M(t)$.

We solve this problem by considering a modified frontier-based exploration method that reasons about

consecutive data in point cloud measurements, inferring environmental properties behind occlusions. Our approach is discussed thoroughly in Chapter 4.

3.2.2 Watchman Tour Generation

The problem of autonomous exploration can be thought of as a watchman who is trying to take the shortest route which guards an entire environment. Here, the watchman is the robot and "guarding" simply means to observe or cover the environment as it navigates. The watchman tour problem is exactly this, an optimization problem which computes the shortest route the vehicle must take to "guard" or explore an environment given a map of the space M_T.

To solve this kind of problem, M_T is broken into a 2-level hierarchy of polygons. At the top level is the bounding polygon, which represents the geometry for the border of the region to be explored. The second level of polygons are called holes, which are representations of obstacles within the environment. Fig. 3.3 demonstrates this setup with one polygonal boundary and the holes that lie within.

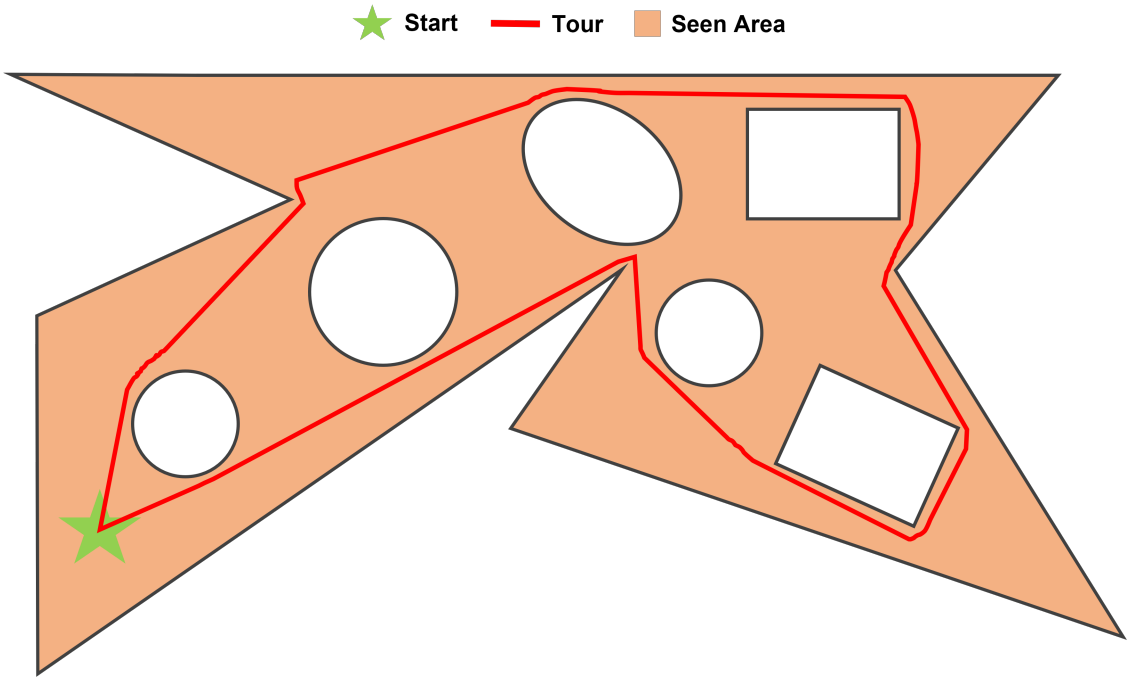


Figure 3.3: Graphical depiction of the watchman tour problem

In the case where the region to be covered is a simple polygon (i.e no holes), the problem can be solved in polynomial time, but becomes intractable once holes are introduced [2]. The problem becomes even more difficult within the context of robotics, since the path has additional constraints based on the vehicle's dynamics, field-of-view (FOV), and footprint, which cannot collide with any obstacles along the path. One

approach to construct such a constrained path is to leverage the notion of the art gallery problem (depicted in Fig. 3.4), where the objective is to cover an environment using many stationary guards.

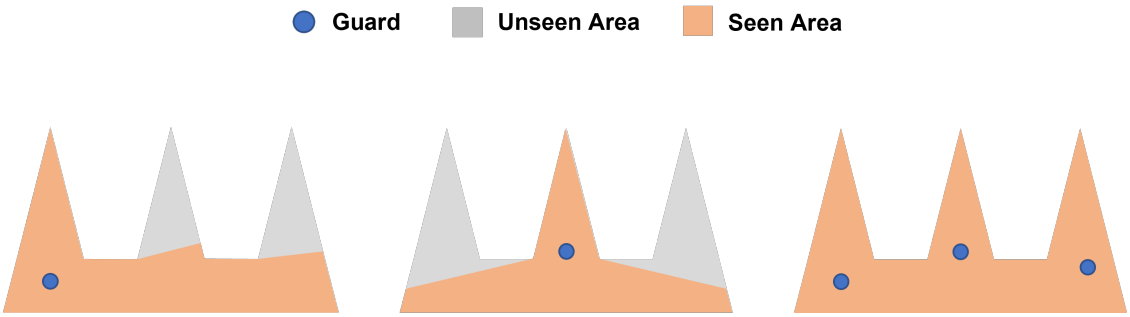


Figure 3.4: Graphical depiction of the art gallery problem

While the art gallery problem is also intractable for polygons with holes[22], it can be approximated through the use of greedy maximization algorithms to find a set of guards which covers the space. Chapter 5 discusses extensively how we use both the art gallery and travelling salesman problems to generate approximate watchman tours which account for the vehicle's physical and sensing constraints by leveraging breadcrumbing.

Given the background discussed above, we can formulate the watchman tour problem within the context of autonomous exploration as follows. Let M_T be the total map generated from time 0 to T after solving the map coverage problem. The objective of the watchman tour problem is to minimize the total travelled trajectory $x^*_{0:T}$ to cover M_T such that every point in M_T is in line of sight. We solve this problem by monitoring and recording the robot's state along with its corresponding sensor readings as a collection of breadcrumbs during the exploration step. Treating these breadcrumbs as stationary guards as in the art gallery problem, we select the approximately minimal collection which still covers the space, then perform a traveling salesman with path simplification to construct the final watchman tour. In Chapter 5, we describe our framework in depth.

Chapter 4

Occlusion Based Exploration

In this section, we describe our proposed path planner for fast exploration of unknown, cluttered environments. The diagram in Fig. 4.1 summarizes the architecture of our framework. We first take as input 2D point cloud sensor readings $\boldsymbol{z}$ and generate waypoints which allow the robot to cover the entire environment (i.e., there will be no more reachable frontiers).

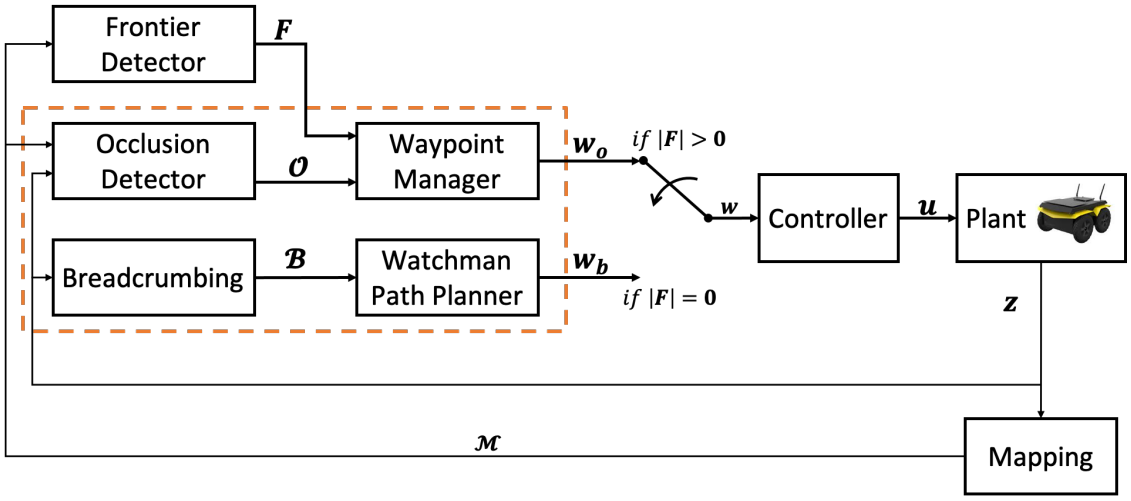


Figure 4.1: Block diagram of proposed approach. The contributions of this thesis are within the orange box.

The waypoints can be broken down into three types based on the three main situations a robot can encounter while exploring an unknown environment: *gap occlusions*, *shadow occlusions*, and *frontiers*. Gap occlusions are measured by discontinuities in contiguous point cloud samples as depicted in Fig. 4.2(a) while shadow occlusions are measured by the occluded region behind an obstacle, shown in Fig. 4.2(b). Lastly, frontiers are the points in open space which lie on the border of explored space by the robot [29].

As these waypoints are constructed, they are saved and evaluated based on criteria like the robot's state and waypoint position. In the following sections, we describe in detail each component of our framework depicted in Fig. 4.1, starting with occlusion detection.

4.1 Gap Occlusion Detection

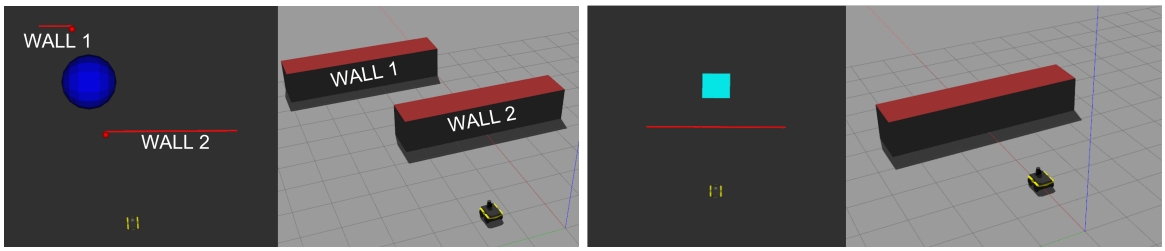
The primary waypoint we consider in our approach is the gap occlusion, which encourages the robot to consider regions of space between two obstacles, where a distant obstacle is being partially occluded by a closer obstacle (see Fig. 4.2(a)). These gap occlusions can be defined as a region of space located between a large gap in two contiguous point cloud readings, $\boldsymbol{z}_i$ and $\boldsymbol{z}_{i+1}$.

Each gap occlusion can be represented as a ball centered at $\boldsymbol{p}_o = \frac{\boldsymbol{p}_i + \boldsymbol{p}_{i+1}}{2}$ with radius $r_o = \tau_o ||\boldsymbol{p}_i - \boldsymbol{p}_{i+1}||$, where τ_o is a tunable parameter used to account for noise in sensor data and prior knowledge about the size of obstacles.

Given sensor readings $\boldsymbol{z}$ with cardinality $\#\boldsymbol{z}$, the set of gap occlusions $\mathcal{G}$ is defined as:

$$\mathcal{G} = \{(\boldsymbol{p}_o, r_o) \mid \text{abs}(\boldsymbol{z}_i - \boldsymbol{z}_{i+1}) > \delta_g \; \forall i \in [1, \#\boldsymbol{z}]\} \tag{4.1}$$

Since occluded regions are considered traversable depending on the size of the robot, the tuning parameter, δ_g, is introduced which controls the minimum distance to be considered a gap occlusion. Given that (4.1) should only detect gaps that the vehicle could navigate through, δ_g should be set at least to the width of the robot. Fig. 4.3 shows the effect of δ_g on gap detection. Smaller gaps that the vehicle can't navigate through don't generate an occlusion, while the larger gaps do. Even though the point cloud readings all belong to the same obstacle, the occlusions are still published since the robot has no way to know if the perceived gaps are valid or not until it navigates closer.



(a) Gap occlusion (b) Shadow centroid

Figure 4.2: Demonstrating the environment conditions responsible for the creation of specific waypoints in the proposed approach. Fig. 4.2(a) demonstrates a large discontinuity in point cloud data responsible for creating a gap occlusion. Fig. 4.2(b) shows an obstacle casting a shadow from the range sensor, which generates a shadow occlusion behind the obstacle.

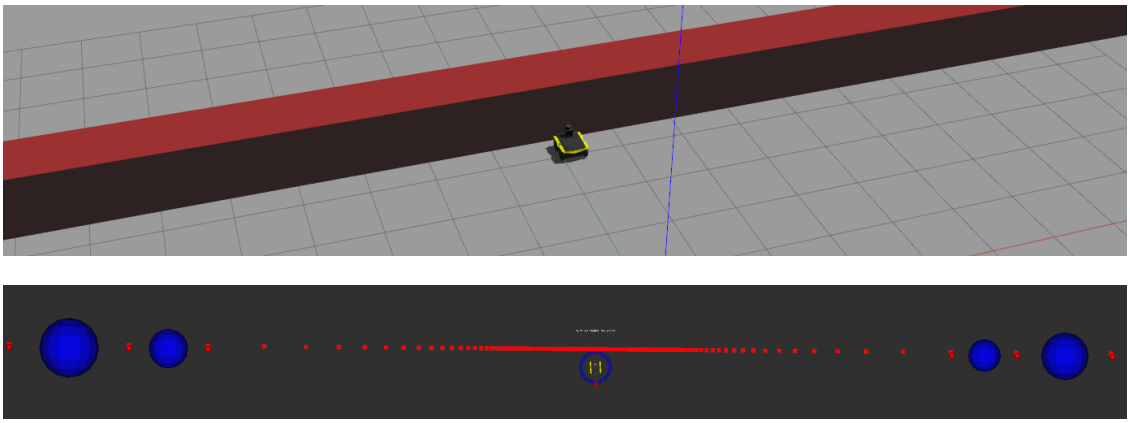


Figure 4.3: Example of point cloud distance samples becoming sparse as the distance from the vehicle increases.

There are cases shown in Fig. 4.4(a) and Fig. 4.4(c), where the vehicle is located near a long, narrow corridor that it cannot fit through, but a gap occlusion is still published. In Fig. 4.4(a), A and B represent $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i+1}$ respectively, where $\boldsymbol{z}_i < \boldsymbol{z}_{i+1}$ and the gap between their measurement readings is $\delta_{AB} > \delta_g$. Fig. 4.4(c) shows the mirrored scenario, where $\boldsymbol{z}_i > \boldsymbol{z}_{i+1}$. To solve this problem, a window-based approach is used to check if the gap is traversable. More formally, we define two windows of points $\mathcal{L}^-$ and $\mathcal{L}^+$ with size κ as

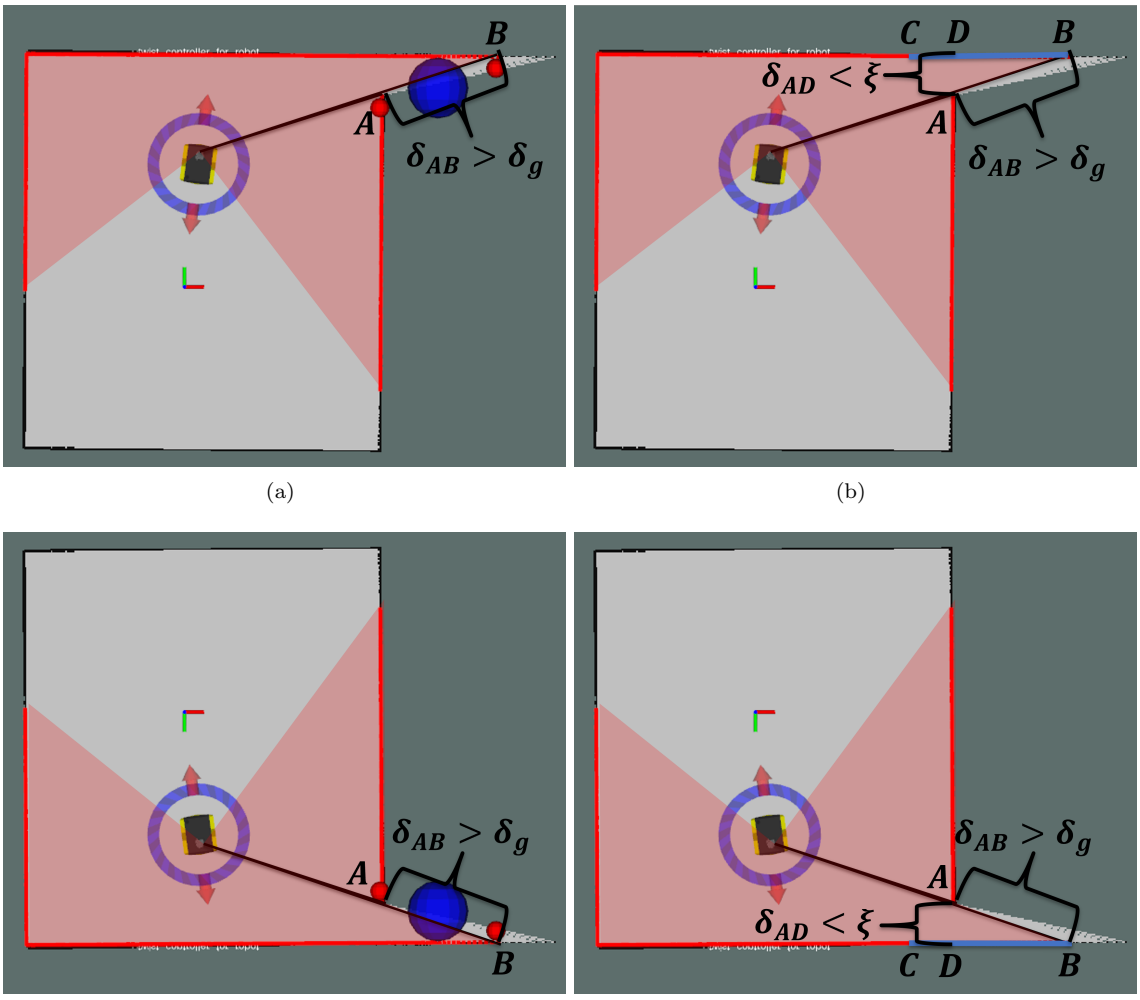
$$\begin{aligned} \mathcal{L}^- &= \{||\boldsymbol{p}_{i-\kappa} - \boldsymbol{p}_i||, \ldots, ||\boldsymbol{p}_{i-1} - \boldsymbol{p}_i||\} \\ \mathcal{L}^+ &= \{||\boldsymbol{p}_{i+2} - \boldsymbol{p}_{i+1}||, \ldots, ||\boldsymbol{p}_{i+1+\kappa} - \boldsymbol{p}_{i+1}||\} \end{aligned} \tag{4.2}$$

$\mathcal{L}^+$ is highlighted blue from point B to C in Fig. 4.4(b), and similarly with $\mathcal{L}^-$ in Fig. 4.4(d). Given both sets, the boolean function Λ which filters gap occlusions as either valid or invalid can be defined as follows:

$$\Lambda(\mathcal{L}^-, \mathcal{L}^+, \xi) = \begin{cases} \mathbb{I}_{\mathcal{L}^-}(d < \xi) & \text{if } \boldsymbol{z}_i < \boldsymbol{z}_{i+1} \\ \mathbb{I}_{\mathcal{L}^+}(d < \xi) & \text{otherwise} \end{cases} \tag{4.3}$$

If $\boldsymbol{z}_i < \boldsymbol{z}_{i+1}$ and there exists distance $d < \xi \in \mathcal{L}^-$, then the occlusion is not published. Conversely, if $\boldsymbol{z}_i > \boldsymbol{z}_{i+1}$ and there exists $d < \xi \in \mathcal{L}^+$, $\Lambda = 0$ and thus the occlusion is not published. Fig. 4.4(b) shows that some point D in $\mathcal{L}^+$ is within distance ξ to $\boldsymbol{p}_i$ and thus the gap occlusion is not published. The mirror case is shown in Fig. 4.4(d).

Continuing with validation of waypoint placement, it is desirable to prevent the planner from creating waypoints in already explored regions. To accomplish this, the occupancy grid map $\mathcal{M}$ is consulted to determine where the robot has yet to explore. For a cell $m_i \in \mathcal{M}$, it can fall into one of three classes depending on its value: 1) **Open:** $m_i < .5$; 2) **Unknown:** $m_i = .5$, and 3) **Occupied:** $m_i > .5$.



(a) (b)

(c) (d)

Figure 4.4: Leveraging a filtering window prevents the placement of erroneous gap occlusions. Fig. 4.4(a) shows a gap occlusion incorrectly being placed in a narrow corridor when $\boldsymbol{z}_i < \boldsymbol{z}_{i+1}$. Fig 4.4(b) shows that no gap occlusion is placed in the corridor when a distance-based filter (highlighted in blue) is applied. Fig. 4.4(c) and 4.4(d) show the mirror case when $\boldsymbol{z}_i > \boldsymbol{z}_{i+1}$.

To prevent a gap occlusion g from being generated in explored space, the occupancy grid is consulted to ensure that the ratio of known to unknown cells around g is below some tunable threshold ψ_g. Let $A_g \subset \mathcal{M}$ denote the bounding box of g, then the occlusion is valid only if the ratio of known cells in A_g to the area $|A_g|$ is below ψ_g. That is,

$$\frac{1}{|A_g|}\left[\sum_{m_i \in A_g, m_i < .5} (1 - m_i)\right] < \psi_g \tag{4.4}$$

4.2 Shadow Occlusion Detection

The second type of waypoint we consider in the path planner is the shadow occlusion, which is defined as the centroid of the region of space behind an obstacle in the environment (see Fig. 4.2(b)).

The shadow occlusion's formulation follows from the definition of an obstacle in the environment. Given $\boldsymbol{z}_i$, $\boldsymbol{z}_{i+1}$ and the tunable parameter α, the obstacle Ω is defined as

$$\Omega = \{\boldsymbol{p}_i \mid \text{abs}(\boldsymbol{z}_i - \boldsymbol{z}_{i+1}) < \alpha\} \tag{4.5}$$

In other words, an obstacle Ω is defined as a collection of points $\boldsymbol{p}_i$ that are no further than distance α from their adjacent points. If the points are too far apart, then they may belong to separate obstacles and thus a gap occlusion should be considered.

The planner keeps track of a set of these obstacles so long as they meet a size criterion. An obstacle Ω with two points does not produce an occluded region large enough to significantly obscure the point cloud sensor. Thus, a parameter β is introduced defining the minimum cardinality Ω. The obstacle set, then, is defined as

$$\mathcal{O} = \{\Omega \mid \#\Omega > \beta\} \tag{4.6}$$

From $\mathcal{O}$, the shadow occlusions are defined as the centroids of the occluded regions behind these obstacles. More formally, let ρ be a tunable parameter for how far behind an obstacle a centroid can be, then given the robot position $\boldsymbol{p}_u$, the coordinates for the shadow centroid $f(\Omega, \boldsymbol{p}_u)$ of obstacle Ω are defined as:

$$f(\Omega, \boldsymbol{p}_u) = \frac{1}{2(\#\Omega)} \left[\sum_{\boldsymbol{p} \in \Omega} \boldsymbol{p} + (\boldsymbol{p} - \boldsymbol{p}_u) * \rho \right] \tag{4.7}$$

Just as in the case with gap occlusions, it is possible for shadow occlusions to be detected in regions which have already been explored. To prevent such a centroid $\boldsymbol{c} = f(\Omega, \boldsymbol{p}_u)$ from being considered as a waypoint, we define $A_r \subset \mathcal{M}$ as the bounding box of the robot centered at $\boldsymbol{c}$ and ψ_c as a tunable threshold parameter. Equation (4.4) is used where A_r is used in place of A_g and ψ_c in place of ψ_c.

$$\mathcal{C} = \{f(\Omega) \mid \Omega \in \mathcal{O}\} \tag{4.8}$$

4.3 Frontier Detection

The last type of waypoint generated by the planner is the frontier, which is used to encourage the robot to explore open regions of the map. As shown in Fig. 4.5, any open cell adjacent to an unknown cell in occupancy grid $\mathcal{M}$ is denoted as a frontier cell, and all adjacent frontier cells are then grouped together into frontier regions [29].

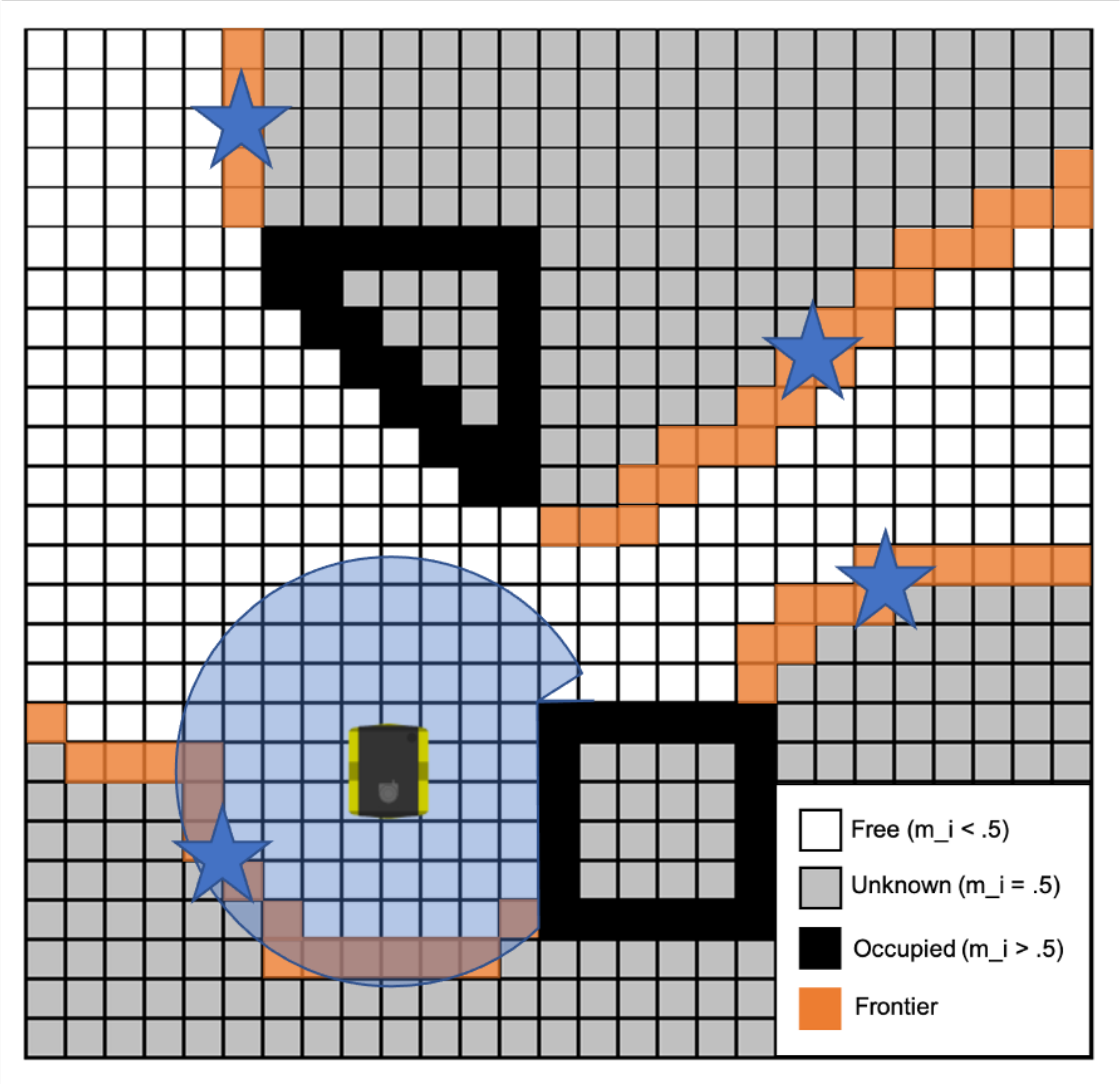


Figure 4.5: Example of frontiers in an occupancy grid [20]

There are a number of approaches to determine these frontier regions, but the algorithm we use, wavefront detection, utilizes a double breadth first search on only newly mapped regions in $\mathcal{M}$ [16]. By only updating the frontiers within the updated regions of the map, we avoid having to perform costly global frontier updates.

This is especially helpful given that the occupancy grids for our applications are typically larger than 20mx20m with a resolution of 5mm.

4.4 Occlusion Manager

At every time step the planner is generating new waypoints from the sensor readings $\boldsymbol{z}$ and the occupancy grid $\mathcal{M}$. These are sent to the occlusion manager, which decides when to add new waypoints and remove old ones.

Let a waypoint be defined as $\boldsymbol{w_i} = (x_i, y_i)$, $\mathcal{W}$ denote the set of waypoints already present in the waypoint manager, and $\mathcal{W}_t$ denote all waypoints detected at the current time step t. The first criterion the manager checks is to see if the vehicle has reached any waypoint in $\mathcal{W}$ and removes it if so.

The second criterion deals with safety and is considered for both old and new waypoints. If any waypoint $\boldsymbol{w} \in \mathcal{W} \cup \mathcal{W}_t$ is within a minimum distance threshold δ_s to an obstacle, it is discarded as a valid waypoint. This distance threshold is set based on the size of the vehicle such that it can safely reach any waypoint in the manager without colliding with other obstacles.

The last criterion dictates that, if a new occlusion $\boldsymbol{w}_n \in \mathcal{W}_t$ is created within distance δ_d to any old occlusions $\boldsymbol{w}_o \in \mathcal{W}$, the old occlusions are removed. More formally, the update to $\mathcal{W}$ is as follows

$$\mathcal{W} \leftarrow \{\boldsymbol{w}_o \mid ||\boldsymbol{w}_n - \boldsymbol{w}_o|| > \delta_d \; \forall \boldsymbol{w}_o \in \mathcal{W}\} \tag{4.9}$$

This criterion is useful for keeping the waypoint list up to date as the vehicle explores more of its environment. In the case of gap occlusions, the jump in lidar data may shift as the vehicle moves towards it, and thus the newly generated occlusion waypoint $\boldsymbol{w_t}$ should replace the out of date waypoint $\boldsymbol{w_o}$.

Once all criteria have been checked for the current timestep, the final set of waypoints $\mathcal{W} \cup \mathcal{W}_t$ is generated. The next step is to score each waypoint to decide where the vehicle should move next in the environment. In the next section we discuss how our scoring scheme works.

4.5 Goal Selection

Provided with the waypoints in the occlusion manager, at each time step the planner generates a cost for each waypoint $\boldsymbol{w}$ based on the vehicle's position $\boldsymbol{p}_u$ and heading θ

$$\Gamma(\boldsymbol{w}) = \tau_D * ||\boldsymbol{p}_u - \boldsymbol{p}_w|| + \tau_H * \text{abs}(\theta - \phi(\boldsymbol{p}_u - \boldsymbol{p}_w)) \tag{4.10}$$

where $\phi(\cdot)$ is a function which gives the angle $\gamma \in [-\pi, \pi]$ from the origin of a passed in vector. Together with the tuning parameter τ_D, the distance term penalizes the waypoints which may be far away from the robot, therefore encouraging local exploration of a region before navigating elsewhere. Due to the distance between waypoints constantly changing as the robot navigates, it becomes possible for a different waypoint to be selected as the new goal at every time step, causing sporadic navigation. To mitigate this, the heading term, defined as the angle between the robot heading and a straight line connecting the robot and waypoint, was introduced along with its tuning parameter τ_H. This term provides incentive for the planner to select waypoints that minimize the need to turn.

Once navigation to all waypoints is complete, our approach will have left breadcrumbs which can be leveraged for future missions. The next section discusses the proposed breadcrumbing process in detail.

Chapter 5

Breadcrumbing

In this section, we describe our framework for generating watchman tours at runtime during an exploration mission by leveraging breadcrumbs. Our proposed approach has three steps: 1) breadcrumbing, 2) max coverage optimization, and 3) tour generation. In the first step, the robot saves its position and sensor readings periodically during navigation as a breadcrumb. As these breadcrumbs are stored, a greedy maximum coverage algorithm is run to find the approximately smallest set of breadcrumbs that cover the observed environment. Once the coverage set has been generated, an approximate traveling salesman algorithm is run to get a shortest-path order to visit the points. This processes repeats whenever a new breadcrumb is saved and terminates once the vehicle has mapped out all of the observable environment.

5.1 Dropping Breadcrumbs

A breadcrumb $\boldsymbol{b}_t$ is defined as a tuple of the robot's pose and the corresponding sensor readings at time t, as shown in Fig. 5.1. At each time step, the approach saves $\boldsymbol{b}_t$ into a breadcrumb set $\mathcal{B}$ if it satisfies two conditions. The first is a safety measure, where breadcrumbs are only recorded if they are farther than some minimum safe distance δ_o from an obstacle

$$\text{CA1: } \boldsymbol{z}_i > \delta_o \; \forall i \in [1, \#\boldsymbol{z}] \tag{5.1}$$

The second condition prevents $\mathcal{B}$ from being flooded while the robot is navigating in a small region of space. Formally, breadcrumb $\boldsymbol{b}_t$ with position $\boldsymbol{p}_t$ must be a minimum distance, δ_b, from all recorded breadcrumbs $\boldsymbol{b}_i \in \mathcal{B}$,

$$\text{CA2: } ||\boldsymbol{p}_i - \boldsymbol{p}_t|| < \delta_b \;\; \forall \boldsymbol{b}_i \in \mathcal{B} \tag{5.2}$$

If the new breadcrumb $\boldsymbol{b}_t$ is within δ_b of a crumb $\boldsymbol{b}_i$, then the one with highest sensor coverage is kept.

Given both conditions are satisfied, the breadcrumb $\boldsymbol{b}_t$ is ready to be added to $\mathcal{B}$ after a sensor reduction stage. Since the number of samples in $\boldsymbol{z}$ is large, recording it for every breadcrumb has high memory cost. Thus, the approach treats $\boldsymbol{z}$ as a visibility polygon $\mathcal{P} = \text{poly}(\boldsymbol{z})$. $\mathcal{P}$ is sent through a point reduction algorithm described in [10] to generate an estimate $\widehat{\mathcal{P}}$ with less samples, thus saving memory space at the cost of reducing measurement fidelity. The final breadcrumb $\boldsymbol{b}_t = (\boldsymbol{p}_u, \widehat{\mathcal{P}})$ is saved to $\mathcal{B}$.

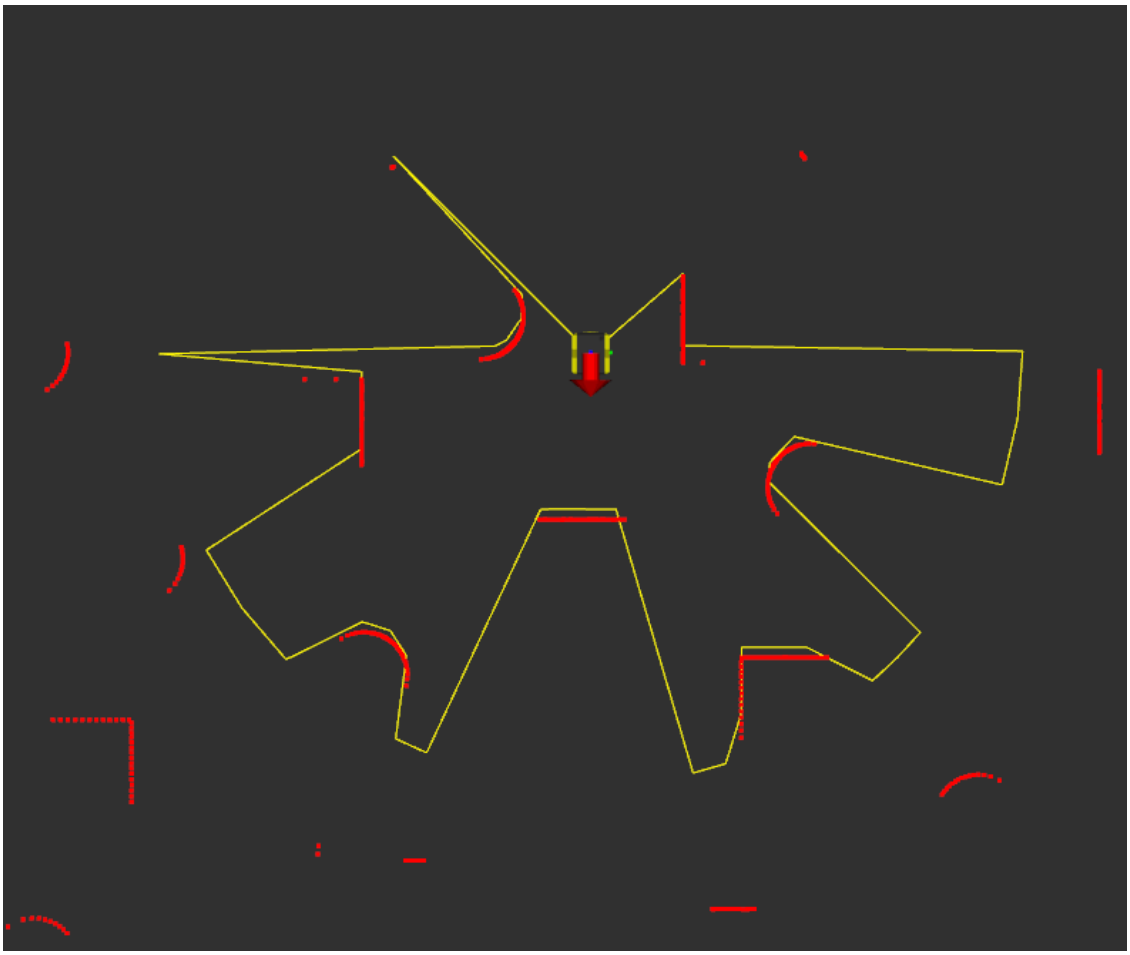


Figure 5.1: Example of a breadcrumb. The robot pose is shown as a red arrow and the reduced sensor visibility is shown in yellow.

In order to keep from running out of memory during large scale explorations, we limit the size of $\mathcal{B}$ to some maximum value N. To decide which breadcrumbs are removed when at capacity, we treat $\mathcal{B}$ as a cache with temporal locality. Deletion occurs at the end of $\mathcal{B}$, and any breadcrumb that was chosen for the optimal coverage set, $\mathcal{B}_t^*$, is moved to the front of $\mathcal{B}$. The assumption driving this decision is that a breadcrumb frequently used in $\mathcal{B}_t^*$ offers high area coverage or visibility into a secluded region and thus should be favored. In the following section, we discuss how $\mathcal{B}_t^*$ is found.

5.2 Finding Approximately Optimal Breadcrumbs

Since $\mathcal{B}$ contains all breadcrumbs recorded by the robot, the total explored region $\mathcal{M}_e$ can be defined as the union of polygons $\widehat{\mathcal{P}}$ of all breadcrumbs in $\mathcal{B}$

$$\mathcal{M}_e = \bigcup_{(\boldsymbol{p}_i, \widehat{\mathcal{P}_i}) \in \mathcal{B}} \widehat{\mathcal{P}_i} \tag{5.3}$$

Whenever a new breadcrumb is recorded, the approach recomputes the approximately minimal cardinality set of breadcrumbs $\mathcal{B}_t^* \subseteq \mathcal{B}$ such that

$$\left| \bigcup_{(\boldsymbol{p}_i, \widehat{\mathcal{P}_i}) \in \mathcal{B}_t^*} \widehat{\mathcal{P}_i} \right| = \zeta |\mathcal{M}_e| \tag{5.4}$$

That is, the area covered by $\mathcal{B}_t^*$ should be equal to a percentage ζ of the total explored area $|\mathcal{M}_e|$. We use ζ because, due to the complexity of environments, it may be that every breadcrumb in $\mathcal{B}$ is required to achieve full coverage, but only a fraction are required to reach an acceptable coverage percentage.

In order to solve this problem, we leverage the fact that finding $\mathcal{B}_t^*$ is a maximum coverage problem where area coverage is to be maximized and the candidate sets are the visibility polygons $\widehat{\mathcal{P}}$ of each breadcrumb. Given that max coverage is an intractable problem, we use an approximate greedy algorithm which leverages the submodular, monotonic properties of area coverage. Doing so provides a solution where $|\mathcal{B}_t^*|$ is no larger than $\frac{e}{e-1}$ times the optimal, where e is the Euler's number [21].

The greedy algorithm, shown below, works by starting with $\mathcal{B}_t^* = \emptyset$ and adding the breadcrumb $\boldsymbol{b}_i = (\boldsymbol{p}_i, \widehat{\mathcal{P}_i}) \in \mathcal{B}$ whose visibility polygon $\widehat{\mathcal{P}_i}$ generates the highest increase in coverage of $\mathcal{B}_t^*$ as computed in (5.4). Once $|\mathcal{B}_t^*| > \zeta * |\mathcal{M}_e|$ or all breadcrumbs are in $\mathcal{B}_t^*$, the algorithm terminates and a watchman tour is ready to be generated.

Algorithm 1 Greedy Coverage

Input: Breadcrumbs $\mathcal{B}$, tolerance ζ
Output: Approximately optimal coverage set $\mathcal{B}_t^*$
$\mathcal{B}_t^* \leftarrow \emptyset$
$\mathcal{B}' \leftarrow \mathcal{B}$
$\mathcal{M}_t \leftarrow 0$
$\mathcal{M}_e \leftarrow \bigcup_{(\boldsymbol{p}_i, \widehat{\mathcal{P}_i}) \in \mathcal{B}_t} \widehat{\mathcal{P}_i}$
while $A(\mathcal{M}_t) < \zeta * A(\mathcal{M}_e)$ **and** $\#B' > 0$ **do**
  $b^* \leftarrow \arg\max_{\boldsymbol{b}^*} \bigcup_{(\boldsymbol{p}_i, \widehat{\mathcal{P}_i}) \in \mathcal{B}_t^* \cup \{b^*\}} \widehat{\mathcal{P}_i}$
  $\mathcal{B}_t^* \leftarrow \mathcal{B}_t^* \cup \boldsymbol{b}^*$
  $B' \leftarrow B' \setminus \{\boldsymbol{b}^*\}$
  $\mathcal{M}_e = \bigcup_{\boldsymbol{b}_i \in \mathcal{B}_t^*} \boldsymbol{b}_i$
end while
return $\mathcal{B}_t^*$

5.3 Watchman Tour Generation

Provided with $\mathcal{B}_t^*$, the objective is to generate a tour through these breadcrumbs such that the robot achieves full coverage faster than during exploration. To accomplish this, 1) we cast the tour as a solution to a traveling salesman problem to determine the visiting order of breadcrumbs and 2) perform a route simplification to remove unnecessary points along the path

Since traveling salesman is an intractable problem, an approximation algorithm called two-opt [6] is used to compute the shortest route $\mathcal{R}$ which has path length no longer than $\sqrt{2}$ times the optimal. Once $\mathcal{R}$ is found, it is reduced by removing breadcrumbs deemed redundant. The reduction works by removing breadcrumbs that the robot will reach by navigating to other breadcrumbs. Let $\boldsymbol{b}_{i-1}$ and $\boldsymbol{b}_{i+1}$ denote the breadcrumbs before and after $\boldsymbol{b}_i$ in $\mathcal{R}$ respectively. There are three conditions that must be satisfied in order for $\boldsymbol{b}_i = (\boldsymbol{p}_i, \widehat{\mathcal{P}}_i)$ to be considered redundant.

The first criterion (CC1), shown in Fig. 5.2(a), checks if $\boldsymbol{b}_{i-1}$, $\boldsymbol{b}_i$, and $\boldsymbol{b}_{i+1}$ form roughly a straight line, meaning that the robot will pass close to $\boldsymbol{b_i}$ while navigating from $\boldsymbol{b}_{i-1}$ to $\boldsymbol{b}_{i+1}$. This is formulated as

$$\text{CC1: } \pi - \arccos\left(\frac{\boldsymbol{u}\cdot\boldsymbol{v}}{||\boldsymbol{u}||\,||\boldsymbol{v}||}\right) < \eta_1 \tag{5.5}$$

where $\boldsymbol{u} = \boldsymbol{p}_i - \boldsymbol{p}_{i-1}$, $\boldsymbol{v} = \boldsymbol{p}_i - \boldsymbol{p}_{i+1}$ and η_1 is a tunable threshold for the angle between $\boldsymbol{u}$ and $\boldsymbol{v}$.

The second criterion (CC2), shown in Fig. 5.2(b), checks if the robot will likely achieve the heading θ_i of breadcrumb $\boldsymbol{b}_i$ while navigating from $\boldsymbol{p}_{i-1}$ to $\boldsymbol{p}_{i+1}$. That is,

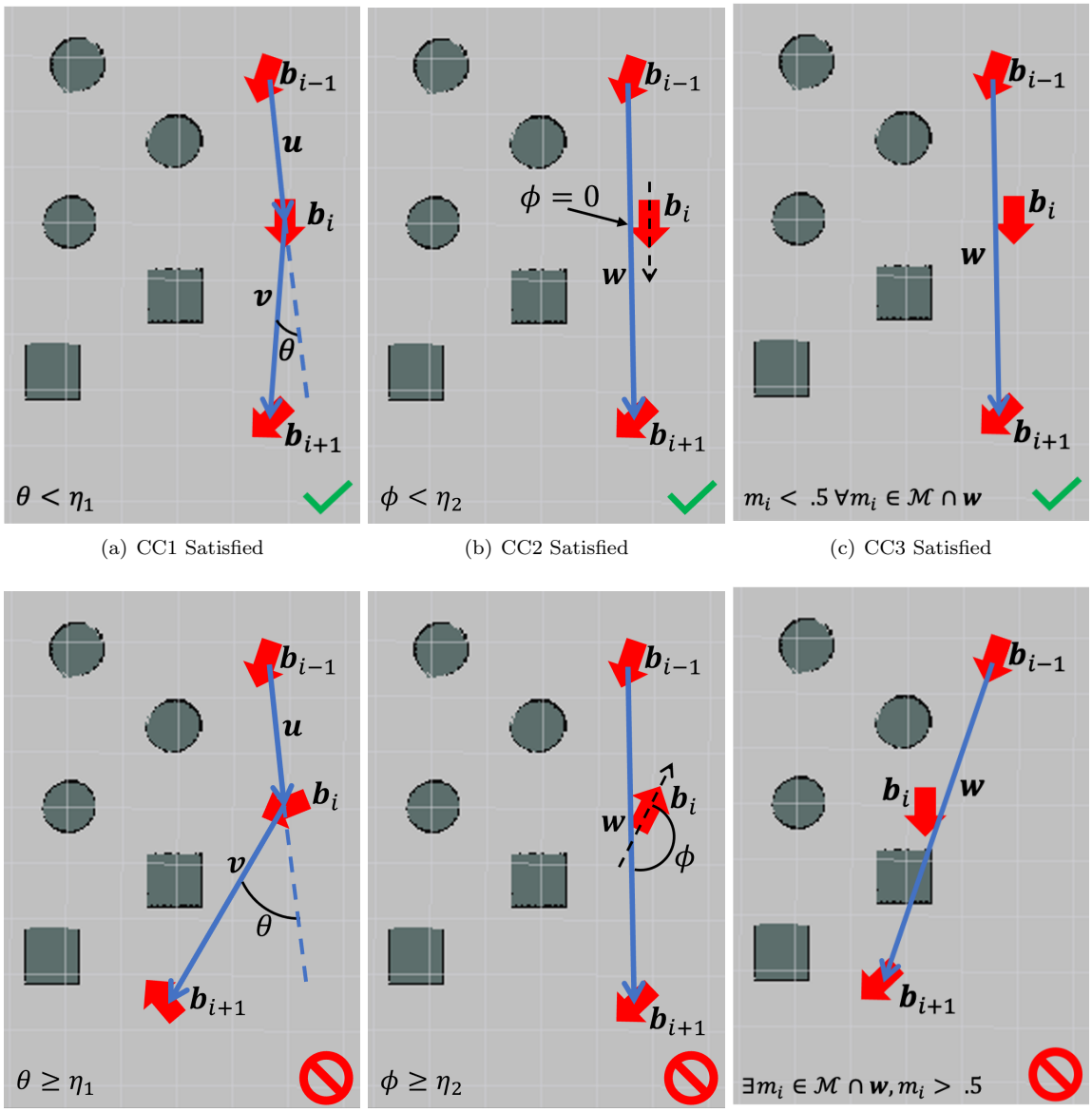
$$\text{CC2: } \theta_i - \arctan\left(\frac{\boldsymbol{w}_2}{\boldsymbol{w}_1}\right) < \eta_2 \tag{5.6}$$

where $\boldsymbol{w} = \boldsymbol{p}_{i+1} - \boldsymbol{p}_{i-1}$ and η_2 a threshold parameter. This constraint can be ignored if the vehicle has a 360° point cloud sensor, since the orientation of the breadcrumb no longer effects the sensor measurements. In Fig. 5.2(b), the line connecting $\boldsymbol{b}_{i-1}$ and $\boldsymbol{b}_{i+1}$ has the same heading as the breadcrumb $\boldsymbol{b}_i$, thus the criterion is satisfied since the difference between the two is less than η_2.

The last criterion (CC3), shown in Fig. 5.2(c), is that the vector lw should not intersect any obstacles in $\mathcal{M}$. More formally, let $\mathcal{M} \cap w$ denote all occupancy map cells $m_i \in \mathcal{M}$ along w, the criterion is

$$\text{CC3: } m_i < .5 \;\forall m_i \in \mathcal{M} \cap w \tag{5.7}$$

Given that all three conditions are true, the breadcrumb $\boldsymbol{b}_i$ is deemed redundant and thus removed as a



(a) CC1 Satisfied (b) CC2 Satisfied (c) CC3 Satisfied

(d) CC1 Not Satisfied (e) CC2 Not Satisfied (f) CC3 Not Satisfied

Figure 5.2: Visual depiction of path simplification criteria

waypoint from the path. The process is repeated until no three consecutive breadcrumbs satisfy all three conditions, giving the final order to navigate through each breadcrumb.

The overall runtime for this breadcrumbing depends on both the number of breadcrumbs N_b in $\mathcal{B}$ and the number of samples N_s in the point cloud data $\boldsymbol{z}$. The bottleneck of this process is the unary union of visibility polygons. For worst case analysis, assume that each visibility polygon could not be reduced, meaning $\widehat{\mathcal{P}}$ has N_s vertices. Also assume that all N_b breadcrumbs were required to cover $\mathcal{M}_e$. The union is performed on the order of N_b^2 times and each union takes $O(N_s \log N_s)$ time, giving a final runtime of

$O(N_b^2 N_s \log N_s)$. Since a theoretical upper bound is still unknown for two-opt, the tour generation is omitted from this runtime discussion.

Chapter 6

Simulations and Experiments

In this section we perform an exhaustive series of simulations and experiments to validate our approach using a UGV with a 2D lidar sensor. The vehicle is dropped into a diverse suite of complex, cluttered testing environments that are unknown to it a priori with the mission of fully mapping the reachable space. We begin the section with an overview of the software stack and vehicle configuration, then move into the simulation and experimental results from the exploration trials.

6.1 Software Stack

In Fig. 6.1, a software stack for our framework is shown. First, the SLAM algorithm localizes our vehicle and generates an updated map of what has been observed thus far in the mission. The map and pose of the vehicle, along with the 2D point scan, are then sent to our occlusion based framework. With this information, an updated list of navigation waypoints are generated for the vehicle to visit. Finally, the waypoints are sent to the motion planner, which will generate a safe trajectory to each waypoint. As the vehicle navigates to these new waypoints, the map is updated and the process repeats until exploration is complete.

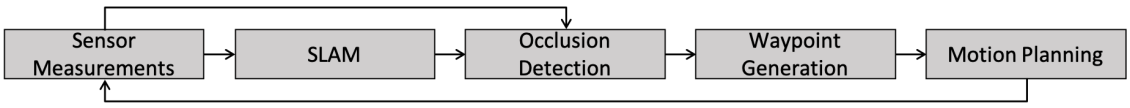


Figure 6.1: Occlusion-based exploration software stack

Fig. 6.2 shows a similar software stack for watchman tour generation. The vehicle's estimated poses via SLAM algorithm are then leveraged by the breadcrumb generation method. Once the approach saves a new pose and point scan into a breadcrumb, the list of all breadcrumbs are sent to the coverage maximization stage. Here, the approximately minimal number of breadcrumbs required to cover a space are found and sent

to the tour generator, which creates the final watchman tour for that iteration. The process repeats as the vehicle moves throughout the environment until exploration has terminated.

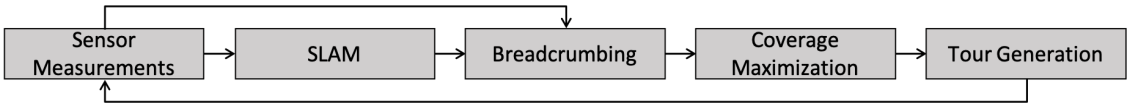


Figure 6.2: Watchman tour software stack

Although the two software stacks are presented separately, these two processes are running concurrently during the exploration phase.

6.2 Gazebo Simulations

We apply our framework to a real-world scenario in which a ground vehicle is deployed to cover an environment and generate watchman tours while also avoiding obstacles. The test-bed we use in this thesis is the Clearpath Jackal, a non-holonomic 4-wheel differential drive ground vehicle.

As discussed in Chapter 3, our vehicle must leverage a pointcloud sensor to properly map the environment it is navigating through. For our simulations, we use the 270° FOV lidar with a maximum range of $30m$. Fig. 6.3 shows the physical system used for simulations.

Everything was implemented on Ubuntu 16.04 under the Robot Operating Systems (ROS), which is an Remote Procedure Call (RPC) based publisher/subscriber communication system. The publishers sent input velocity commands to the vehicle based on our generated waypoints and path planner, and subscribers received the sensor data, distributing the information to our software. In order to simulate the environments for testing with the ground vehicle, we used the ROS enabled Gazebo simulator, which allows for high-fidelity testing of robotic algorithms in user-defined environments. For all testing done in simulation, Gazebo is run along with our entire testing stack on the same computer. As shown in Fig. 6.4, Gazebo is able to generate the sensor measurements that the simulated Jackal is receiving from its environment which we can use to feed into our software stacks in Fig. 6.1 and Fig. 6.2.

For path planning, we used the Dyanmic Window Approach (DWA) local planner [11], informed by an A* global planner with a maximum velocity of $2m/s$ and acceleration of $5m/s^2$. The DWA planner works by sampling many velocity inputs at the current Jackal position, then scores the end configuration based on distance to the goal, nearby obstacles, and the global path. By using this local planner, the Jackal is able to safely and quickly maneuver to each waypoint our framework sends.

Our approach was evaluated in Gazebo using three case studies: 1) a warehouse, 2) a large cluttered environment, and 3) a bookstore. These environments were selected because they cover a large spectrum

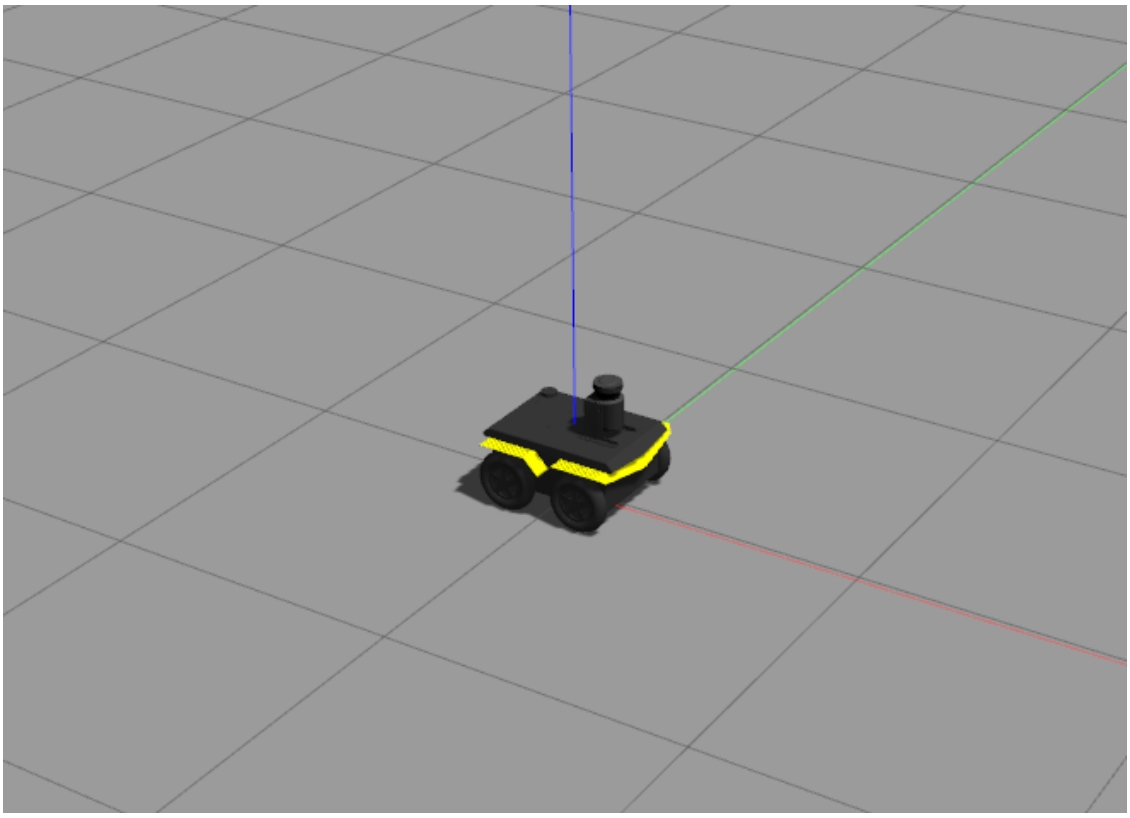


Figure 6.3: Jackal setup in Gazebo

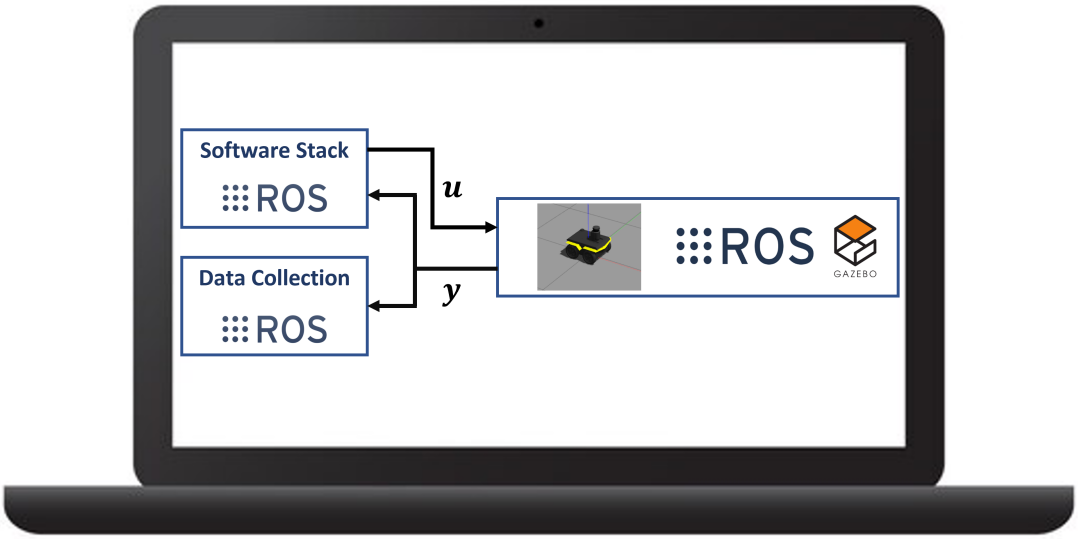
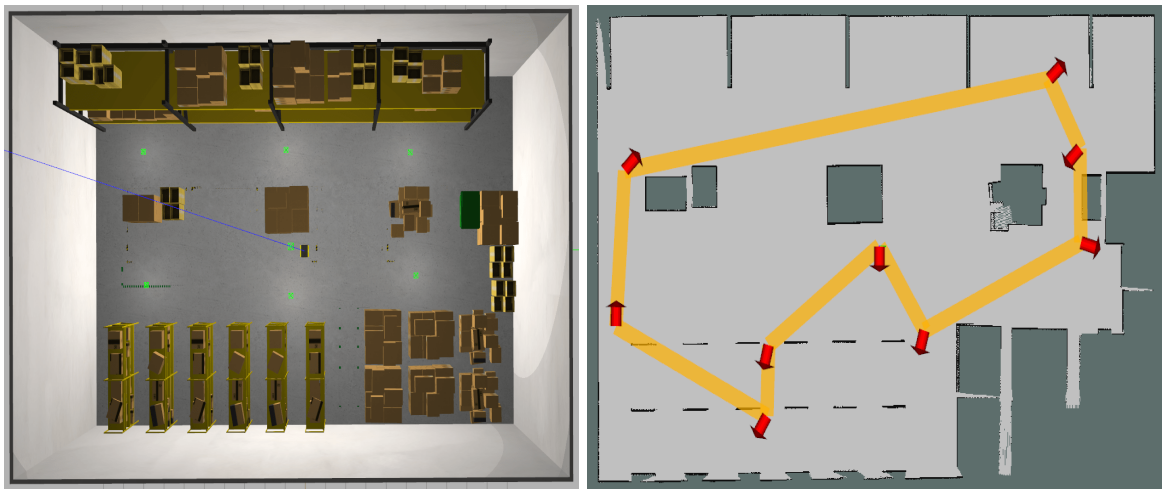


Figure 6.4: Framework for the simulation setup

of common situations encountered in the real world. In all simulations, the watchman tour coverage was set to 99% and each breadcrumb could only use a truncated lidar scan with max range of $5m$. This was done because the map update was only performed in a $5m$ radius around the UGV. Finally, simulations were only terminated when no reachable waypoints are left. In the case of breadcrumb based navigation, the simulations were terminated once the UGV had navigated through all breadcrumbs in the tour.

6.2.1 Warehouse Case Study

Several simulations were performed in the Warehouse environment depicted in Fig. 6.5, where our approach was compared with a frontier-only exploration algorithm. From Table 6.1, it can be observed that our occlusion-based framework can explore the environment significantly faster than the frontier-only approach over 3 runs. Furthermore, when navigating through the space utilizing breadcrumbs collected during initial exploration, we observe an additional reduction in distance traveled with no drop in exploration speed, giving large improvements to exploration time. In Fig. 6.6, it can be seen that our approach achieves a faster rate of exploration on average at all points of time when compared with frontier-only exploration.



(a) (b)

Figure 6.5: Warehouse environment (a) and associated map with watchman tour (b).

Table 6.1: Warehouse Exploration Results

Approach	**avg. speed (m/s)**	**avg. distance (m)**	**avg. time (s)**
frontiers	0.4	108.45	235.3
occlusions (ours)	**0.45**	83.37	171.3
breadcrumbs (ours)	**0.45**	**64.46**	**140.0**

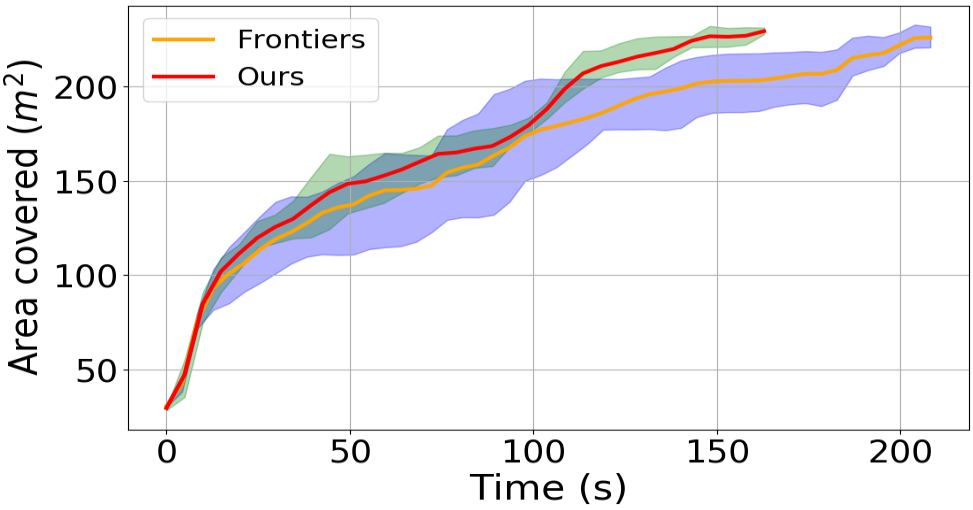


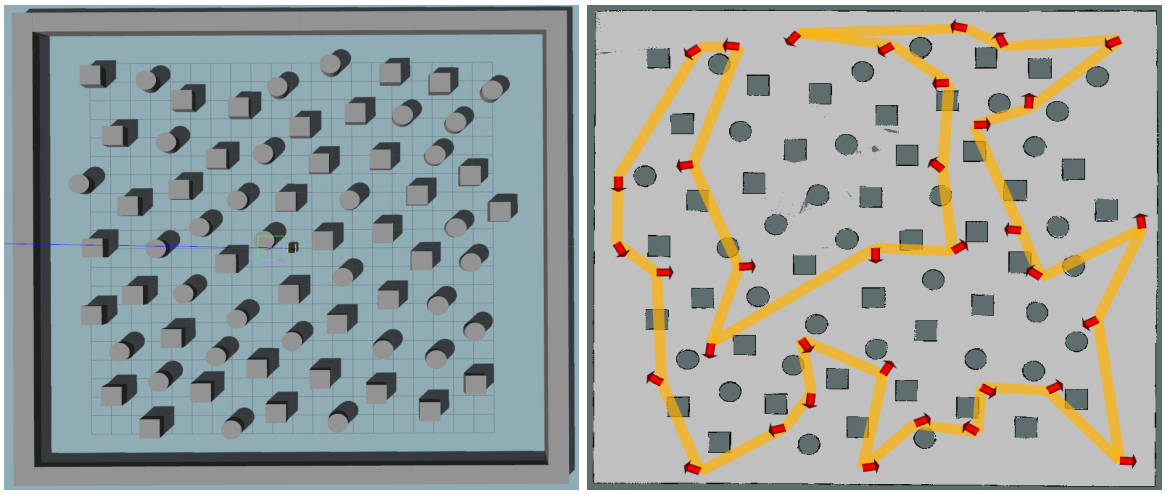
Figure 6.6: Warehouse environment explored area over time.

6.2.2 Cluttered Environment Case Study

In order to evaluate our framework in a realistic outdoor environment, we created a $25m$ x $25m$ cluttered environment with cube and cylindrical obstacles (Fig. 6.7). The environment is meant to be a rough model of a dense forest, where each obstacle is akin to tree trunks and large rocks. As seen by Table 6.2, our approach is further demonstrated to outperform the frontier-based approach, and performance is further improved when utilizing breadcrumbs in a second exploration mission to cover 99% of the environment. Fig. 6.8(a) shows the area covered over time when following the generated watchman tour. As can be seen, it only took 358 seconds to navigate through all breadcrumbs and achieve the desired coverage, which is much faster than both frameworks during initial exploration. Furthermore, Fig. 6.8(b) shows the area covered as a function of the number of breadcrumbs. As expected, coverage increases with the number of breadcrumbs. Additionally, there is a diminishing marginal increase in area as the number of breadcrumbs increases, which shows the submodularity of the coverage function as discussed in Section 5.2.

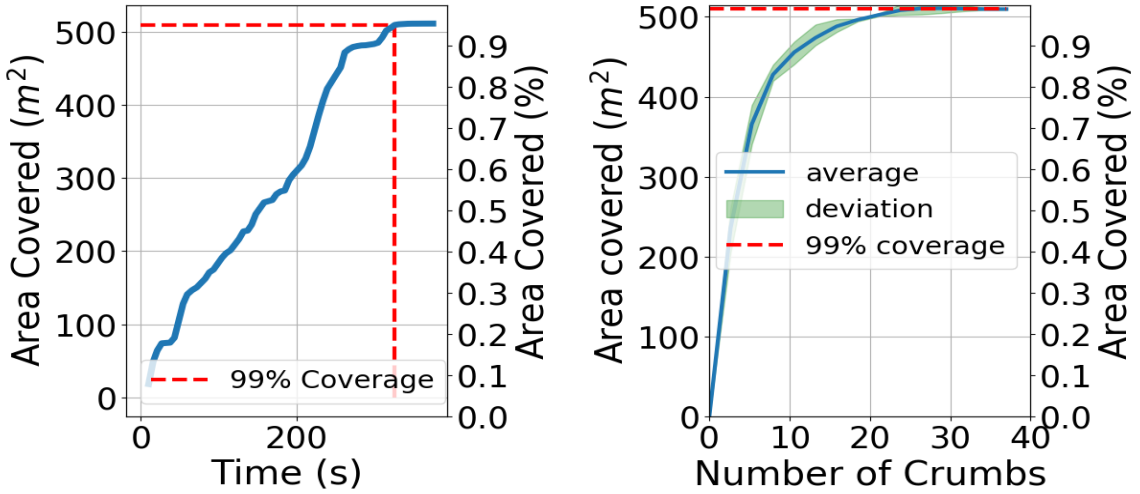
Table 6.2: Cluttered Environment Exploration Results

Approach	**avg. speed (m/s)**	**avg. distance (m)**	**avg. time (s)**
frontiers	0.35	224.36	641.0
occlusions (ours)	**0.42**	211.79	504.3
breadcrumbs (ours)	0.35	**142.35**	**406.7**



(a) (b)

Figure 6.7: Cluttered environment (a) and associated map with watchman tour (b).



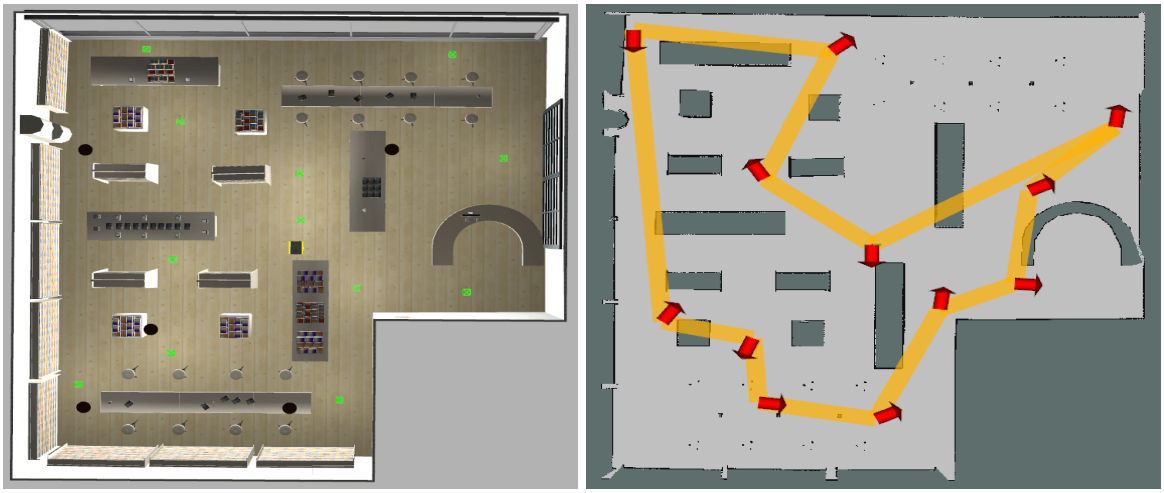
(a) (b)

Figure 6.8: Area covered over time for breadcrumb navigation (a) and area coverage vs number of breadcrumbs (b).

6.2.3 Bookstore Case Study

Simulations were also performed in a bookstore environment (see Fig. 6.9) that is smaller than the warehouse, but more densely packed with obstacles, making it a good case study to test our approach. Table 6.3 shows the speed, distance, and time averaged over 3 runs for each navigation framework. In this environment, our approach achieved a slightly faster exploration speed while travelling less overall distance, thus completing exploration faster than the frontier framework. In the case of breadcrumb based navigation, we observe even

faster exploration speeds while drastically reducing overall distance travelled when compared with both initial exploration missions.



(a) (b)

Figure 6.9: Library environment (a) and associated map with watchman tour (b).

Table 6.3: Bookstore Exploration Results

Approach	**avg. speed (m/s)**	**avg. distance (m)**	**avg. time (s)**
frontiers	0.29	99.11	277.0
occlusions (ours)	0.30	84.99	222.0
breadcrumbs (ours)	**0.32**	**62.83**	**152.0**

6.3 Experiments

We test our framework on several real-world scenarios in which a ground vehicle is deployed to cover an environment and generate watchman tours while also avoiding obstacles. Just as in simulation, the test-bed we use is the Clearpath Jackal, however, we now use the 360° FOV VLP-16 Puck lidar with a maximum range of $100m$. Fig. 6.10 shows the physical system used for experiments

All code was run on board the Jackal for all experiments and we used a separate computer only for recording data, as that is a computationally intensive task that would reduce the vehicles exploration performance. Fig. 6.11 shows the overall setup for our experiments.

Figure 6.10: Jackal setup for experiments

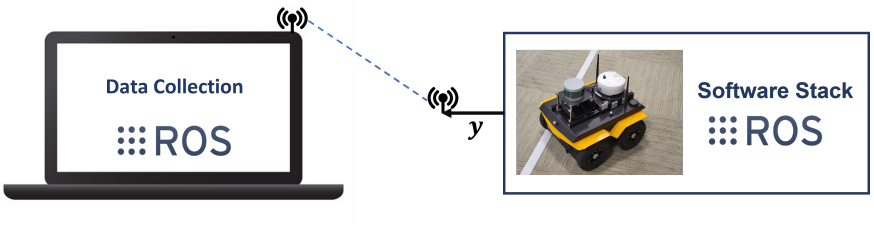


Figure 6.11: Framework for the experiment setup

6.3.1 Basement

The basement environment, shown below in Fig. 6.12, is an ideal case study for our approach. There are several branching segments that force the UGV to consider whether it should continue along its current path, or deviate and explore the newfound branch. In the runs, our approach correctly discerned that it was faster to explore the right-most branch first, before continuing to proceed to the left through the main corridor. By doing so, the Jackal was able to avoid spending a lot of time traveling through already known space.

Moving to the watchman tour generation, a single breadcrumb in each corridor is enough to cover the branches, and our approach is able to determine this. In the right-most vertical corridor in Fig.6.12, our framework understands that in order to cover that region with the fewest number of crumbs, it must be strategically placed near the center. This way, it covers not only the main corridor, but also the bulge that

extends off of it. Another encouraging observation is that redundant breadcrumbs along the main corridor were removed, leaving only two connecting breadcrumbs which have the Jackal cover the region. Lastly, Fig. 6.12 also shows the Jackal was able to continually make progress in mapping with little time spent not making new observations. This means that the Jackal was most often in regions of unknown space, seldom backtracking through completely covered portions of the map.

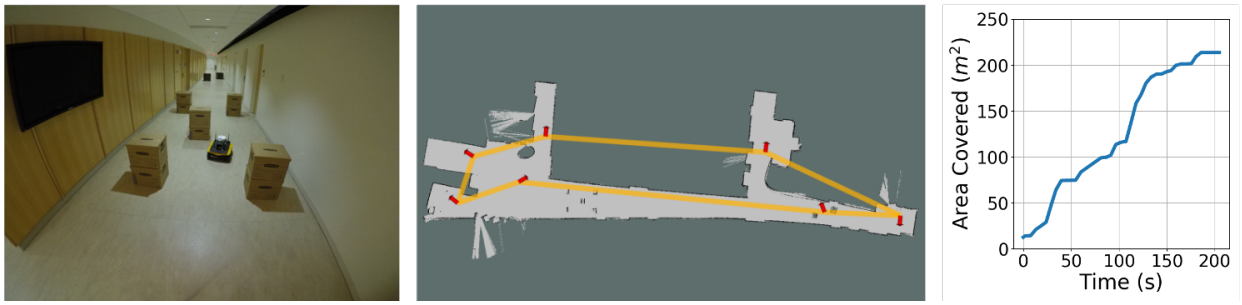


Figure 6.12: Basement setup (left), associated map with watchman tour (middle), and coverage over time (right).

6.3.2 Office

Unlike the basement case study, this office space is more linear in terms of exploration. There are no major branches that the robot has to consider, however, there are several obstacles within the space, and a small isolated region in the upper path of the space which make this an interesting case study. Note in the final watchman tour in Fig. 6.13, our approach was able to drop a single breadcrumb to cover the top section of the space. Furthermore, it was able to remove several redundant breadcrumbs in the lower part of the environment, allowing the Jackal to still observe the space between the couches with fewer overall breadcrumbs.

Figure 6.13: Office setup (left), associated map with watchman tour (middle), and coverage over time (right).

Chapter 7

Current and Future Work

We suggest that our proposed framework for autonomous exploration can be extended to solve other problems beyond the coverage of unknown environments. In particular, we are interested in leveraging our proposed schemes for exploration and exploitation to reach a goal in the shortest amount of time. In this section, we describe our current framework for agile navigation through unknown environments, and then discuss the future work, in which we will apply our exploration framework to improve current performance.

Given an a priori unknown cluttered environment $\mathcal{W}$ with N obstacles in which a region $\mathcal{M} \subseteq \mathcal{W}$ is traversable, the go-to-goal problem is an objective optimization problem to find a control policy $\mathcal{U}(t)$ which reaches a goal pose $\boldsymbol{x}_g = (x_g, y_g, \theta_g)$ in the minimal amount of time while avoiding all obstacles.

Fig. 7.1 shows an example of the situation envisioned for this problem in which the UGV is tasked to traverse a cluttered environment to reach a final goal behind unknown obstacles [23]. Our current work to deal with this problem leverages a hierarchical system which selects local navigation goals based on perceived gaps and open areas within the FOV of the lidar. By performing these steps at every timestep, we can generate a moving local goal which quickly guides the vehicle to the final goal location.

The first step of our approach is to take as input a 2D scan and partition it into a short and long region in order to generate local goals to guide the vehicle. To do this, we create an occupancy function $\Psi(\boldsymbol{z_i}, \alpha)$ which returns 1 if a given lidar point $\boldsymbol{z}$ is below a certain distance α and 0 otherwise.

$$\Psi(\boldsymbol{z}_i, \alpha) = \begin{cases} 1 & \text{if } \boldsymbol{z}_i \leq \alpha \\ 0 & \text{otherwise} \end{cases} \tag{7.1}$$

Next, Ψ can be applied onto the lidar scan $\boldsymbol{z}$ with varying α values to generate the short scan $\mathcal{S}_{\boldsymbol{z}}$ and long scan $\mathcal{L}_{\boldsymbol{z}}$ partitions. We define them as follows:

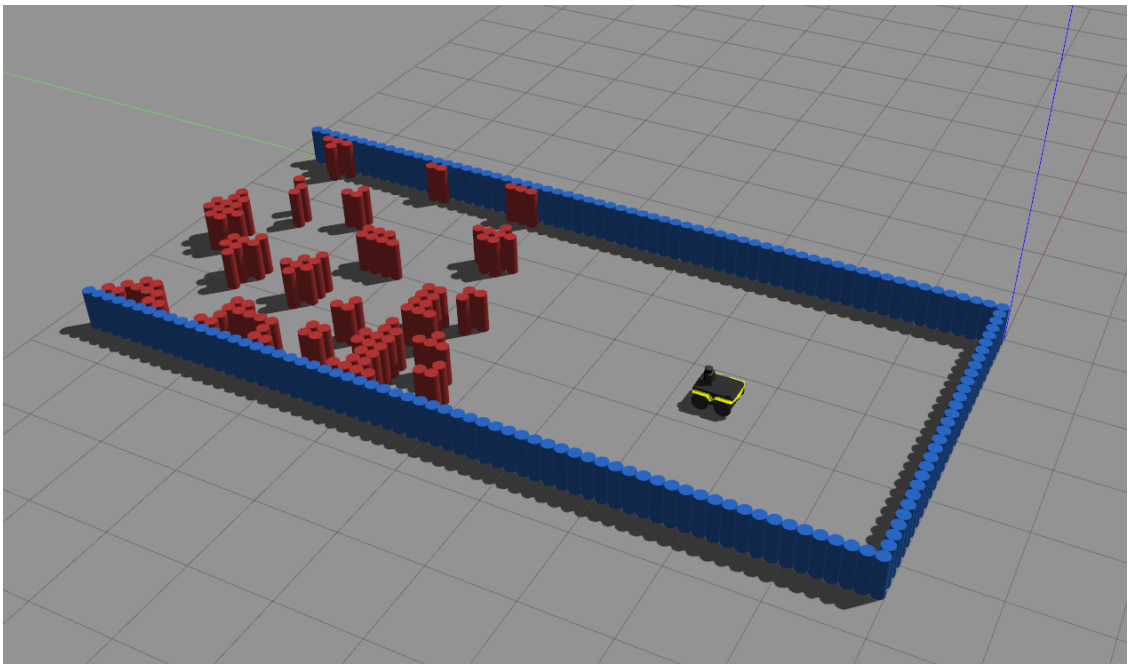


Figure 7.1: Example of cluttered environment for go-to-goal mission.

$$\begin{aligned} \mathcal{S}_{\boldsymbol{z}} &= \{\Psi(\boldsymbol{z}_i, \alpha_s)\ \forall \boldsymbol{z_i} \in \boldsymbol{z}\} \\ \mathcal{L}_{\boldsymbol{z}} &= \{\Psi(\boldsymbol{z}_i, \alpha_l)\ \forall \boldsymbol{z_i} \in \boldsymbol{z}\} \end{aligned} \tag{7.2}$$

where $\alpha_s < \alpha_l$. Now $\mathcal{S}_{\boldsymbol{z}}$ contains 1s for all distance measurements lower than α_s and 0s for all lidar readings larger, giving a local sense to the vehicle of open regions to navigate towards. By using Ψ on the lidar scan in this way, we have generated an occupancy-like data-structure which can inform the vehicle on what regions of its sensing range should be avoided by looking for large clusters of 1s. Similarly, the vehicle can look for regions of 0s which denote open, traversable space.

Given the two partitions $\mathcal{S}_{\boldsymbol{z}}$ and $\mathcal{L}_{\boldsymbol{z}}$, waypoints for navigation can now be sampled. The first step in this process is to find all ranges of consecutive 0s within the sets. Doing so generates a list $\mathcal{I}$ of indices $\boldsymbol{i} = (i_1, i_2)$ for each set denoting the start and end of the collection of 0s. For example, if $\mathcal{S}_{\boldsymbol{z}} = \{0, 0, 0, 1, 1, 1, 0, 0, 1\}$, then $\mathcal{I}_s = \{(1, 3), (7, 8)\}$. These indices represent regions of free space around the vehicle that are potential candidates for navigation waypoints.

Given $\mathcal{I}_s$ and $\mathcal{I}_l$, the next step is to determine if the vehicle can fit through the free space regions defined by the clusters of 0s. In the example above, the largest consecutive collection of 0s in $\mathcal{S}_{\boldsymbol{z}}$ was 3 long. Given that the 2D scan has many samples and high angular resolution, it is very unlikely that a traditionally sized vehicle would fit through that region unless α_s was very large. With this in mind, we define two terms ω_s and ω_l which denote the minimum consecutive 0s required for the vehicle to be able to fit through the open

space in a given partition. Given the padded vehicle's width w_p and angle increment of the 2D depth sensor δ_θ, the two constants can be derived via the law of cosines

$$\begin{aligned} \omega_s &= \left\lceil \frac{1}{\delta_\theta} \arccos\left(\frac{2\alpha_s^2 - w_p^2}{2\alpha_s^2}\right) \right\rceil \\ \omega_l &= \left\lceil \frac{1}{\delta_\theta} \arccos\left(\frac{2\alpha_l^2 - w_p^2}{2\alpha_l^2}\right) \right\rceil \end{aligned} \tag{7.3}$$

With ω_s and ω_l defined, $\mathcal{I}_s$ and $\mathcal{I}_l$ can be filtered into $\widehat{\mathcal{I}}_s$ and $\widehat{\mathcal{I}}_l$ respectively, which only contain the indices of open regions of space that are large enough to fit through.

$$\begin{aligned} \widehat{\mathcal{I}}_s &= \{\boldsymbol{i} \; \forall \; \boldsymbol{i} \in \mathcal{I}_s \mid \boldsymbol{i}_2 - \boldsymbol{i}_1 > \omega_s\} \\ \widehat{\mathcal{I}}_l &= \{\boldsymbol{i} \; \forall \; \boldsymbol{i} \in \mathcal{I}_l \mid \boldsymbol{i}_2 - \boldsymbol{i}_1 > \omega_l\} \end{aligned} \tag{7.4}$$

Once $\widehat{\mathcal{I}}_s$ and $\widehat{\mathcal{I}}_l$ are known, the short waypoints $\boldsymbol{p}_s \in \mathcal{W}_s \subset \mathbb{R}^2$ are generated by taking the average index $M(\boldsymbol{i}) = \left\lfloor \frac{\boldsymbol{i}_1 + \boldsymbol{i}_2}{2} \right\rfloor$ for each $\boldsymbol{i} \in \widehat{\mathcal{I}}_s$. Let $\boldsymbol{p}_{M(\boldsymbol{i})}$ denote the xy coordinate of measurement $\boldsymbol{z}_{M(\boldsymbol{i})}$ at index $M(\boldsymbol{i})$. $\mathcal{W}_s$ is defined as

$$\mathcal{W}_s = \{\boldsymbol{p}_{M(\boldsymbol{i})} \; \forall \boldsymbol{i} \in \widehat{\mathcal{I}}_s\} \tag{7.5}$$

The formulation for $\mathcal{W}_l$ is nearly identical, with the only difference being to replace $\widehat{\mathcal{I}}_s$ with $\widehat{\mathcal{I}}_l$. Once both waypoint sets have been generated, the final selection can be performed. The first step in this process is to determine if there are any waypoints in $\mathcal{W}_l$. If so, we select the waypoint $\boldsymbol{p}_l^* = (x_l, y_l)$ which has the closest euclidean distance to the goal. We then choose the short waypoint $\boldsymbol{p}_s^* = (x_s, y_s)$ that is closest in heading to $\boldsymbol{p}_l^*$ with respect to the vehicle's pose $\boldsymbol{x} = (x, y, \theta)$

$$\boldsymbol{p}_s^* = \underset{\boldsymbol{p}_s = (x_s, y_s) \in \mathcal{W}_s}{\arg\min} \operatorname{abs}\left(\arctan\left(\frac{y - y_s}{x - x_s}\right) - \arctan\left(\frac{y - y_l}{x - x_l}\right)\right) \tag{7.6}$$

In the case that $\#\mathcal{W}_l = 0$, we simply choose the short waypoint $\boldsymbol{p}_s^*$ which is closest in euclidean distance to the goal. If $\#\mathcal{W}_s = 0$, then the vehicle must have navigated to a dead end, and a short point behind the vehicle is placed at a fixed distance α_s.

Fig. 7.2 below shows an example environment in which our approach runs. The purple region denotes the long partition $\mathcal{L}_{\boldsymbol{z}}$ while the blue region represents the partition $\mathcal{S}_{\boldsymbol{z}}$. The green point is the selected long waypoing $\boldsymbol{p}_l^*$ while the yellow ones are the other possible candidates that weren't selected. Since the vehicle is directly facing the goal point, it makes sense that the waypoint in the center of the vehicle was chosen for $\boldsymbol{p}_l^*$. Once $\boldsymbol{p}_l^*$ was selected, a $\boldsymbol{p}_s^*$ was sampled from $\mathcal{W}_s$ which aligns best in heading with $\boldsymbol{p}_l^*$.

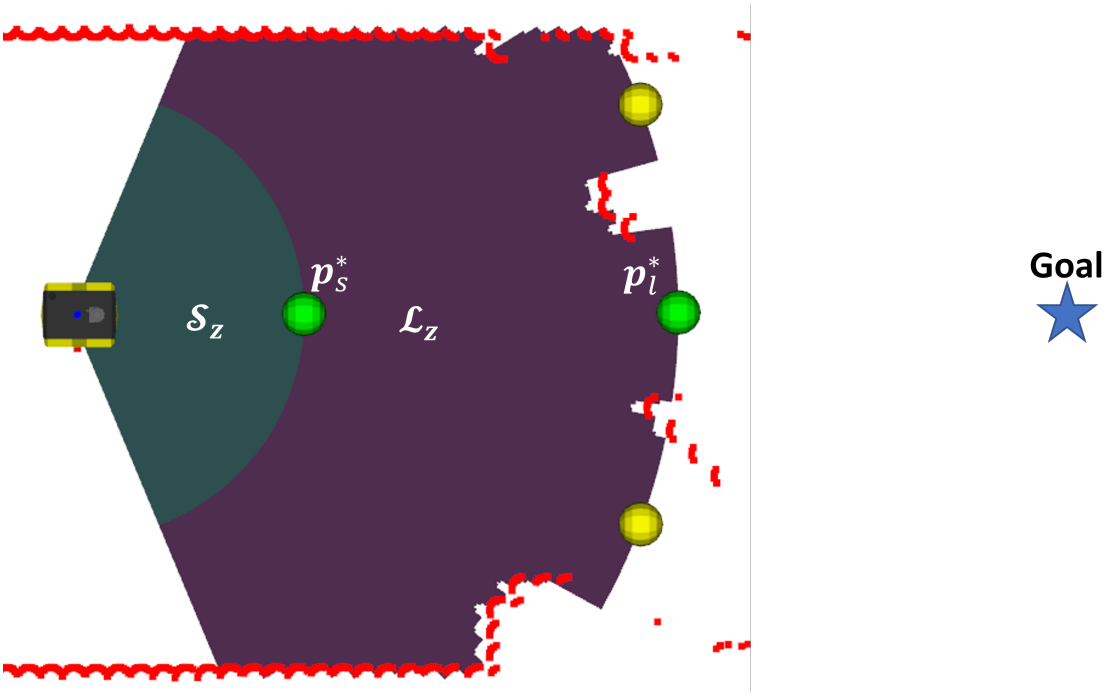


Figure 7.2: Visual depiction of $\mathcal{S}_z$ and $\mathcal{L}_z$

With the short point generated for navigation, the final step is to calculate the velocity command to send to the vehicle at the current time step. To generate velocity commands that take into consideration the limited turning rate at high speeds, we leverage the curvature of the path required to reach the short waypoint to influence velocity, where curvature is defined as

$$\kappa = \frac{2h}{d^2} \tag{7.7}$$

where h is the horizontal distance and d the euclidean distance from the vehicle to $\boldsymbol{p}_s^*$. Since the short point is always on the boundary of the short partition $\mathcal{S}_z$, it is always the case that $d = \alpha_s$, thus no extra computation needs to be performed to find it. Given the vehicle pose $\boldsymbol{x} = (x, y, \theta)$ and the short point $\boldsymbol{p}_s^* = (x_s, y_s)$, h can be calculated using the following equation

$$h = \frac{\text{abs}\left(-\tan(\theta) * x_s + y_s + x\tan(\theta) - y\right)}{\sqrt{\tan^2(\theta) + 1}} \tag{7.8}$$

Given the curvature, the final linear and angular velocities v and ω are defined as

$$\begin{aligned} v &= \min(v_{max}, \frac{\beta}{h}) \\ \omega &= \min(\omega_{max}, \tau \cdot h \cdot v) \end{aligned} \tag{7.9}$$

where β is a tunable parameter controlling how fast the vehicle should move while turning sharply and τ is either 1 or -1 depending on which side of the vehicle $\boldsymbol{p}_s$ lies.

While this approach shows promising results and is able to quickly navigate through cluttered environments when compared with other competitive motion planners like the Dynamic Window Approach (DWA), there are still several shortfalls that need to be addressed. Firstly, the approach can get caught in local minima like dead-ends. Another key issue with the approach is that partitioning the lidar scan $\boldsymbol{z}$ into two sections $\mathcal{S}_{\boldsymbol{z}}$ and $\mathcal{L}_{\boldsymbol{z}}$ and only sampling waypoints at the borders discards a lot of information.

To solve these problems, we plan to create a more natural waypoint selection algorithm based on gap occlusions within the environment. Often times the goal may be occluded by dense clusters of obstacles, and by navigating towards these occluded regions, the vehicle may gain important information about the environment which may inform the path it chooses to follow. If the goal is still occluded after navigating to this region, then the process is repeated until the goal is within the vehicles FOV. Fig. 7.3 shows an example environment with our proposed occlusion based approach.

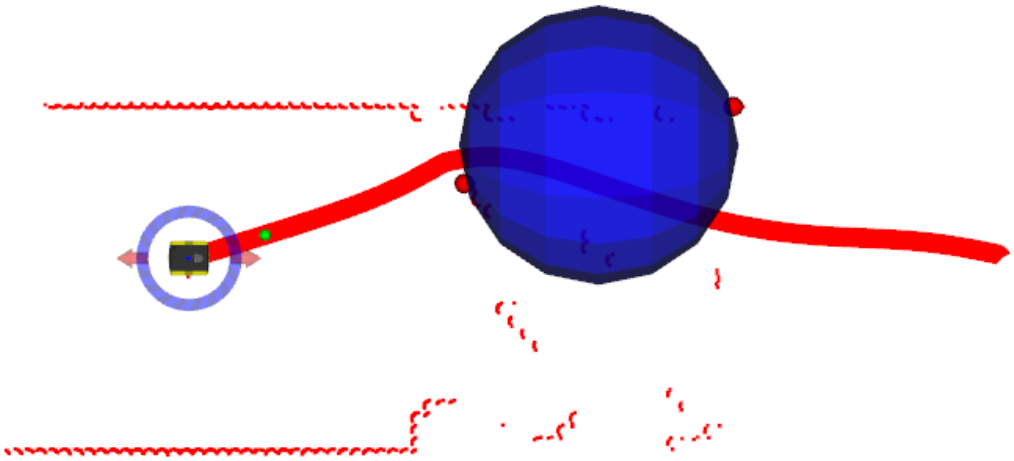


Figure 7.3: Example environment in which a collision free spline is generated to the occlusion and then to the final goal.

Given how these occlusions are generated, it is common for them to be located slightly behind occluding obstacles, making it difficult for navigation. To this end, we plan to add an underlying trajectory generator which can create fast paths through the environment while avoiding obstacles. By picking which occlusions to navigate towards intelligently, we can generate fast, collision free paths to each until the goal is unobstructed by obstacles.

7.1 Preliminary Results

Just as in our exploration framework testing, we use the Gazebo simulator with a Clearpath Jackal as shown in Fig. 6.4 to run preliminary tests of our navigation approach. In order to test on a wide variety of environments with varying complexities of difficulty, we use the Benchmark for Autonomous Robot Navigation (BARN) dataset [23]. The primary advantage of this dataset is that it can randomly generate worlds of a specified difficulty, which is defined as the time to navigate through the shortest path to goal at maximum speed. We can utilize this optimal traversal time to benchmark the optimality of our approach and compare this to other navigation techniques. In this work, we compare with the DWA planner, which was used in the exploration simulations and experiments. While this approach has a high success rate through these environments, the approach achieves traversal times that are far from the optimal.

We ran both our framework and DWA on a gauntlet of worlds generated by the BARN dataset and compared their performance. Fig. 7.4 shows an example BARN world along with the path the Jackal is following to navigate through it. Out of 300 randomly generated worlds, where world 0 is the easiest and world 299 the hardest, we sampled worlds in 3 ranges to observe the performance of both approaches in easy, medium, and hard scenarios. The easy band was from worlds 20 to 40, medium from 140 to 160, and hard from 260 to 280. Table 7.1 shows the success rate in each difficulty along with the average traversal time for both approaches. For the purposes of these simulations, success was defined as the vehicle making it to the coordinate $(0, 10)$ within 100 seconds and without colliding into any obstacles along the way.

	DWA		Ours	
Difficulty	Success (%)	Time (s)	Success (%)	Time (s)
Easy	90.5	31.27	95.2	12.49
Medium	80.9	34.75	66.7	13.22
Hard	71.4	46.98	28.6	14.2

Table 7.1: Comparison of performance between DWA and our approach

From these preliminary results, our approach is able to achieve significantly faster traversal times through all worlds regardless of difficulty when compared with DWA. Furthermore, in the easy worlds, we also have a higher success rate. However, our success rate drops off significantly as the difficulty of the worlds increases. Many of these failures in the higher difficulty worlds are due to the fact that they are riddled with dead-ends, which require the vehicle to make sharp turns once identified. Our approach has difficulty detecting these dead ends effectively due to the inherent information loss in our lidar partitioning scheme. Furthermore, as expected the vehicle struggles to correctly navigate away from these dead ends based on the waypoints generated from our approach, often resulting in collisions while trying to make aggressive, sharp turns.

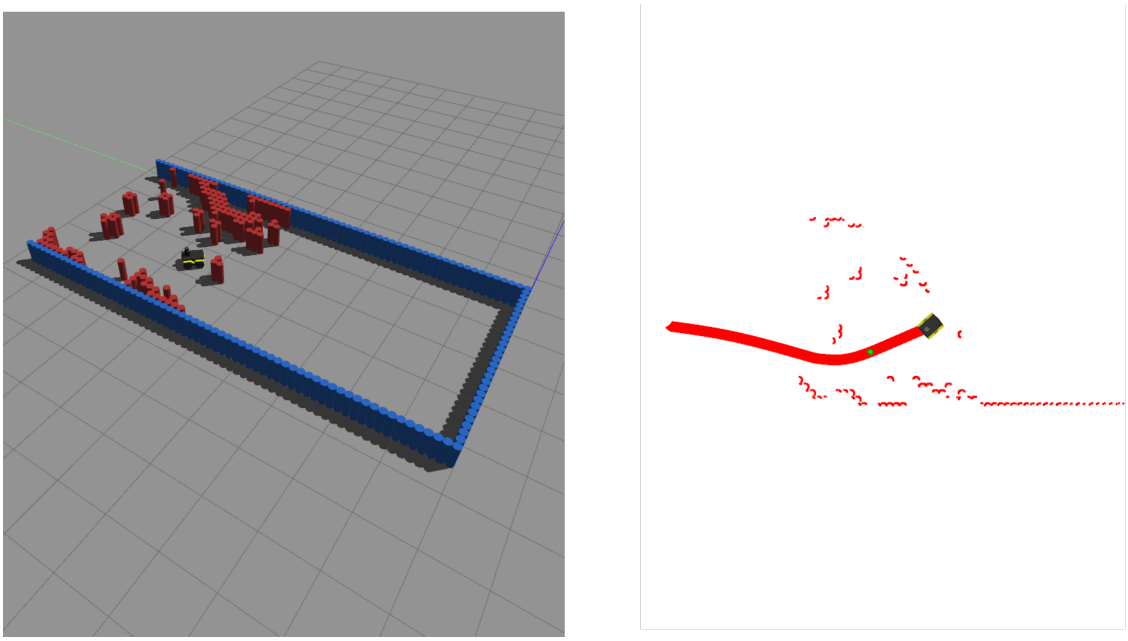


Figure 7.4: Vehicle during go-to-goal mission along with a visualization of the collision free path being tracked.

In order to resolve these issues, we plan to include our occlusions method to inform the vehicle about where to go next since this can provide more natural waypoints for navigation. By saving the occlusions previously observed into a tree structure as the vehicle moves through the environment, we can leverage the structure to escape dead-ends. If the current occlusion being used for navigation leads to a dead-end, the vehicle can navigate back to the parent node of the occlusion and start moving towards its next child. In this way, the tree serves as breadcrumbs which the robot saves to assist in navigation. Another benefit of this approach is that it utilizes the entire lidar scan without partitioning, which assists in preventing the placement of waypoints which lie too close to obstacles or dead-ends in the horizon past and in between the short and long bands.

Chapter 8

Conclusions and Future Work

8.1 Conclusion

In this thesis, we have presented a novel strategy that leverages both frontiers and occlusions to efficiently explore unknown cluttered environments.

Furthermore, within our proposed framework we propose a breadcrumbing method to record important information in the form of breadcrumbs for use in future missions. The principle behind our approach is that, since occluded regions infer unexplored space, they should be targeted as prioritized navigation goals during exploration. As shown in both simulations and experiments, our approach is capable of quickly exploring unknown environments, and can outperform frontier-only approaches in terms of exploration speed. From this, we learned that occlusions do, in fact, serve as useful guides for exploration, and can be exploited to achieve fast coverage of a space.

In regards to exploitation, we validated our breadcrumbing based watchman tour generation framework, which is able to construct short, full coverage paths of an environment. These tours were shown to reduce both exploration time and distance traveled significantly, demonstrating that the approach is working as designed. Another benefit of our greedy maximization approach is the computational efficiency, since it is able to generate the watchman tours online. Additionally, given the model agnostic nature of this approach, regardless of the sensing configuration of the vehicle, the framework can generate watchman tours which respect the FOV of the pointcloud sensor. While we focused primarily on breadcrumbing for use in covering an environment, the benefits of these watchman tours extend beyond the contexts of mapping and coverage. Examples include surveillance applications for guarding regions of environments or even search and rescue. Another use-case could be vacuum cleaning, in which the vehicle can refine the path it takes to clean an

entire space.

Finally, we discussed in Chapter 7 how we are currently attempting to solve the problem of agile navigation through cluttered, unknown environments to reach a desired goal. While our current work shows preliminary results in which we are able to navigate environments faster than traditional approaches like DWA, success rates are still too low. Given this, we will be extending this framework to incorporate our occlusion-based navigation to assist in navigation through the environment. Furthermore, to overcome the issue of local minima, we will be leveraging the ideas of breadcrumbing through the use of a history tree, which stores previously observed occlusions for use as backtracking waypoints for when the vehicle encounters dead-ends.

Bibliography

[1] Anthony Brunel et al. "SplatPlanner: Efficient Autonomous Exploration via Permutohedral Frontier Filtering". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 608–615. DOI: 10.1109/ICRA48506.2021.9560896.

[2] W Chin and S Ntafos. "Optimum Watchman Routes". In: *Proceedings of the Second Annual Symposium on Computational Geometry*. SCG '86. Yorktown Heights, New York, USA: Association for Computing Machinery, 1986, pp. 24–33. ISBN: 0897911946. DOI: 10.1145/10515.10518. URL: https://doi.org/10.1145/10515.10518.

[3] Wei-Pang Chin and Simeon Ntafos. "Optimum watchman routes". In: *Proceedings of the second annual symposium on Computational geometry*. 1986, pp. 24–33.

[4] Titus Cieslewski, Elia Kaufmann, and Davide Scaramuzza. "Rapid exploration with multi-rotors: A frontier selection method for high speed flight". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 2135–2142. DOI: 10.1109/IROS.2017.8206030.

[5] C. Connolly. "The determination of next best views". In: *Proceedings. 1985 IEEE International Conference on Robotics and Automation*. Vol. 2. 1985, pp. 432–435. DOI: 10.1109/ROBOT.1985.1087372.

[6] Georges A Croes. "A method for solving traveling-salesman problems". In: *Operations research* 6.6 (1958), pp. 791–812.

[7] Anna Dai et al. "Fast Frontier-based Information-driven Autonomous Exploration with an MAV". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 9570–9576. DOI: 10.1109/ICRA40945.2020.9196707.

[8] Tung Dang, Christos Papachristos, and Kostas Alexis. "Autonomous exploration and simultaneous object search using aerial robots". In: *2018 IEEE Aerospace Conference*. 2018, pp. 1–7. DOI: 10.1109/AERO.2018.8396632.

[9] Mihir Dharmadhikari et al. “Motion Primitives-based Path Planning for Fast and Agile Exploration using Aerial Robots”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 179–185. DOI: 10.1109/ICRA40945.2020.9196964.

[10] David H. Douglas and Thomas K. Peucker. “ALGORITHMS FOR THE REDUCTION OF THE NUMBER OF POINTS REQUIRED TO REPRESENT A DIGITIZED LINE OR ITS CARICATURE”. In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 10 (1973), pp. 112–122.

[11] D. Fox, W. Burgard, and S. Thrun. “The dynamic window approach to collision avoidance”. In: *IEEE Robotics Automation Magazine* 4.1 (1997), pp. 23–33. DOI: 10.1109/100.580977.

[12] Subir Kumar Ghosh et al. “Online Algorithms with Discrete Visibility - Exploring Unknown Polygonal Environments”. In: *IEEE Robotics Automation Magazine* 15.2 (2008), pp. 67–76. DOI: 10.1109/MRA.2008.921542.

[13] David P. Grabowsky, James M. Conrad, and Aidan F. Browne. “A Breadcrumb System for Assisting Outdoor Autonomous Robots with Path Identification and Localization”. In: *SoutheastCon 2021*. 2021, pp. 1–6. DOI: 10.1109/SoutheastCon45413.2021.9401846.

[14] Jacob Higgins and Nicola Bezzo. “Negotiating Visibility for Safe Autonomous Navigation in Occluding and Uncertain Environments”. In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4409–4416. DOI: 10.1109/LRA.2021.3068701.

[15] Lucas Janson, Tommy Hu, and Marco Pavone. “Safe Motion Planning in Unknown Environments: Optimality Benchmarks and Tractable Policies”. In: *CoRR* abs/1804.05804 (2018). arXiv: 1804.05804. URL: http://arxiv.org/abs/1804.05804.

[16] Matan Keidar and Gal A. Kaminka. *Efficient Frontier Detection for Robot Exploration*. 2013.

[17] Steven M. Lavalle. *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. Tech. rep. Iowa State University, 1998.

[18] Louis Ly and Yen-Hsi Richard Tsai. “Autonomous Exploration, Reconstruction, and Surveillance of 3D Environments Aided by Deep Learning”. In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 5467–5473. DOI: 10.1109/ICRA.2019.8794426.

[19] Nicola Mimmo, Pauline Bernard, and Lorenzo Marconi. “Avalanche victim search via robust observers”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 4066–4072. DOI: 10.1109/ICRA40945.2020.9196646.

[20] Rasoul Mojtahedzadeh. "Robot obstacle avoidance using the Kinect". In: *Master of Science Thesis Stockholm, Sweden* (2011).

[21] G.L Nemhauser, L.A Wolsey, and M.L Fisher. "An analysis of approximations for maximizing submodular set functions". In: *Mathematical Programming*. 1978, pp. 1–30. DOI: 10.1007/BF01588971.

[22] J. O'Rourke and K. Supowit. "Some NP-hard polygon decomposition problems". In: *IEEE Transactions on Information Theory* 29.2 (1983), pp. 181–190. DOI: 10.1109/TIT.1983.1056648.

[23] Daniel Perille et al. *Benchmarking Metric Ground Navigation*. 2020. DOI: 10.48550/ARXIV.2008.13315. URL: https://arxiv.org/abs/2008.13315.

[24] Victor Massagué Respall et al. "Fast Sampling-based Next-Best-View Exploration Algorithm for a MAV". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 89–95. DOI: 10.1109/ICRA48506.2021.9562107.

[25] P.-L. Richard et al. "LineRanger: Analysis and Field Testing of an Innovative Robot for Efficient Assessment of Bundled High-Voltage Powerlines". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 9130–9136. DOI: 10.1109/ICRA.2019.8794397.

[26] Michael Tatum. "Communications coverage in unknown underground environments". In: *Masters's thesis, The Robotics Institute, Carnegie Mellon University, USA* (2020).

[27] Emanuele Vespa et al. "Efficient Octree-Based Volumetric SLAM Supporting Signed-Distance and Occupancy Mapping". In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 1144–1151. DOI: 10.1109/LRA.2018.2792537.

[28] Christian Witting et al. "History-Aware Autonomous Exploration in Confined Environments Using MAVs". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 1–9. DOI: 10.1109/IROS.2018.8594502.

[29] B. 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'*. 1997, pp. 146–151. DOI: 10.1109/CIRA.1997.613851.

[30] Boyu Zhou et al. "FUEL: Fast UAV Exploration Using Incremental Frontier Structure and Hierarchical Planning". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 779–786. DOI: 10.1109/LRA.2021.3051563.