2019

Sensor-Based Topological Coverage And Mapping Algorithms For Resource-Constrained Robot Swarms

Rattanachai Ramaithitima
University of Pennsylvania, hanuman337@gmail.com

Sensor-Based Topological Coverage And Mapping Algorithms For Resource-Constrained Robot Swarms

Abstract
Coverage is widely known in the field of sensor networks as the task of deploying sensors to completely cover an environment with the union of the sensor footprints. Related to coverage is the task of exploration that includes guiding mobile robots, equipped with sensors, to map an unknown environment (mapping) or clear a known environment (searching and pursuit- evasion problem) with their sensors. This is an essential task for robot swarms in many robotic applications including environmental monitoring, sensor deployment, mine clearing, search-and-rescue, and intrusion detection. Utilizing a large team of robots not only improves the completion time of such tasks, but also improve the scalability of the applications while increasing the robustness to systems' failure.

Despite extensive research on coverage, mapping, and exploration problems, many challenges remain to be solved, especially in swarms where robots have limited computational and sensing capabilities. The majority of approaches used to solve the coverage problem rely on metric information, such as the pose of the robots and the position of obstacles. These geometric approaches are not suitable for large scale swarms due to high computational complexity and sensitivity to noise. This dissertation focuses on algorithms that, using tools from algebraic topology and bearing-based control, solve the coverage related problem with a swarm of resource-constrained robots.

First, this dissertation presents an algorithm for deploying mobile robots to attain a hole-less sensor coverage of an unknown environment, where each robot is only capable of measuring the bearing angles to the other robots within its sensing region and the obstacles that it touches. Next, using the same sensing model, a topological map of an environment can be obtained using graph-based search techniques even when there is an insufficient number of robots to attain full coverage of the environment. We then introduce the landmark complex representation and present an exploration algorithm that not only is complete when the landmarks are sufficiently dense but also scales well with any swarm size. Finally, we derive a multi-pursuers and multi-evaders planning algorithm, which detects all possible evaders and clears complex environments.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Computer and Information Science

First Advisor
Vijay Kumar

Second Advisor
Subhrajit Bhattacharya

Keywords
exploration, mapping, sensor coverage, swarms

Subject Categories
Robotics

SENSOR-BASED TOPOLOGICAL COVERAGE AND MAPPING ALGORITHMS FOR RESOURCE-CONSTRAINED ROBOT SWARMS

Rattanachai Ramaithitima

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2019

Vijay Kumar, Supervisor of Dissertation
Nemirovsky Family Dean of Penn Engineering

Subhrajit Bhattacharya, Co-Supervisor of Dissertation
Assistant Professor, Mechanical Engineering and Applied Mechanics

---

Rajeev Alur, Graduate Group Chairperson
Zisman Family Professor, Computer and Information Science

Dissertation Committee

Camillo J. Taylor, Professor of Computer and Information Science
M. Ani Hsieh, Research Associate Prof. of Mechanical Engineering and Applied Mechanics
Robert W. Ghrist, Professor of Mathematics and Electrical and Systems Engineering
Alberto Speranzon, Technical Fellow, Honeywell

Acknowledgments

This thesis would not have been possible without the support of many people. First and foremost, I would like to thank my advisors, Vijay Kumar and Subhrajit Bhattacharya, for all of their support and guidance throughout my Ph.D. It has been my pleasure to work under their guidance and be involved in many interesting projects. I would also like to thank my entire committee for all the valuable feedback that helped improve my writing and presentation of this project.

Some of the projects included in this thesis were collaboration projects that involved efforts of many people. In particular, I would like to thank Michael Whitzer, who helped me with all the Scarab experiments. Alberto Speranzon, Siddharth Srivastava, and Shaunak Bopardikar also provided many ideas that helped shape this thesis during my internship at United Technologies Research Center. Additionally, I would like to thank everyone at MRSL and PERCH for making my time in graduate school as enjoyable as I could hope for.

I also want to thank all my Thai friends at Penn who made me feel at home despite being halfway around the world. Last but not least, I would like to thank my family overseas for being patient and supportive throughout my long journey at Penn.

ABSTRACT

SENSOR-BASED TOPOLOGICAL COVERAGE AND MAPPING ALGORITHMS FOR RESOURCE-CONSTRAINED ROBOT SWARMS

Rattanachai Ramaithitima

Vijay Kumar

Subhrajit Bhattacharya

Coverage is widely known in the field of sensor networks as the task of deploying sensors to completely cover an environment with the union of the sensor footprints. Related to coverage is the task of exploration that includes guiding mobile robots, equipped with sensors, to map an unknown environment (mapping) or clear a known environment (searching and pursuit-evasion problem) with their sensors. This is an essential task for robot swarms in many robotic applications including environmental monitoring, sensor deployment, mine clearing, search-and-rescue, and intrusion detection. Utilizing a large team of robots not only improves the completion time of such tasks, but also improve the scalability of the applications while increasing the robustness to systems' failure.

Despite extensive research on coverage, mapping, and exploration problems, many challenges remain to be solved, especially in swarms where robots have limited computational and sensing capabilities. The majority of approaches used to solve the coverage problem rely on metric information, such as the pose of the robots and the position of obstacles. These geometric approaches are not suitable for large scale swarms due to high computational complexity and sensitivity to noise. This dissertation focuses on algorithms that, using tools from algebraic topology and bearing-based control, solve the coverage related problem with a swarm of resource-constrained robots.

First, this dissertation presents an algorithm for deploying mobile robots to attain a hole-less sensor coverage of an unknown environment, where each robot is only capable of measuring the bearing angles to the other robots within its sensing region and the obstacles that it touches. Next, using the same sensing model, a topological map of an environment can be obtained using graph-based search techniques even when there is an insufficient number of

robots to attain full coverage of the environment. We then introduce the landmark complex representation and present an exploration algorithm that not only is complete when the landmarks are sufficiently dense but also scales well with any swarm size. Finally, we derive a multi-pursuers and multi-evaders planning algorithm, which detects all possible evaders and clears complex environments.

Contents

List of Figures

Chapter 1

Introduction

The recent advent of technologies called the Internet of Things (IoT) clearly holds significant potential benefit to individuals, society, and the economy [19, 33]. As the demand for sensors and processors continues to increase, their price/performance ratio has been steadily falling; thus leading to the new wave of technological advancement in autonomy including smart home, smart city, and self-driving vehicles. Nevertheless, robustness has been one of the main hurdles in deployment of these technologies. The current trend to achieve robustness has been focused on using a large number of high precision sensors, resulting in high production costs as well as intensive computational requirements such as processor and memory usage. These components would also come with high energy consumption and a large amount of payload. For instance, the autonomous vehicle, developed by Waymo, consists of three Lidar (Light Detection and Ranging) sensors, five Radar (Radio Detection and Ranging) sensors, and a vision system [91]. The total cost of these sensors alone would be more than $50,000. Certainly, the price of these sensors will drop as their demand increases. However, the specification for other corresponding requirements will likely increase. For instance, the complexity of associating sensing information increases as large numbers of autonomous vehicles will eventually need to share their sensing information.

A similar situation occurs in sensor networks which form an important component of the smart city/home. The majority of existing solutions assume that the absolute global

position of the sensors and the map are available. In contrast, this dissertation explores an alternative approach to robustness through use of a large number of basic sensors. There is very little work on how small, resource-constrained sensors and processors can be utilized in large-scale environments, especially when the map is not available *a priori*. These small, resource-constrained sensors are particularly appealing to commercial applications such as crop monitoring [60], vacuum cleaning [100], structural inspection [32, 68], environmental monitoring [83], and intruder detection [45] due to the low production cost, the robustness to failure of individual sensors, and the scalability of the environment they can operate in.

The fundamental problems of these applications include information gathering, mapping, and exploration, all of which require the mobile robots to cover an environment with their sensors either permanently, persistently, or temporarily. Designing a swarm of robots to solve the sensor coverage problem in these applications encompasses many different areas of robotic research including, but not limited to, planning, localization, control, perception, and mapping. For instance, each robot must be able to autonomously navigate to various parts of the environment. This step requires the robot to plan a path that does not collide with obstacles, and then generate control inputs driving the actuators to follow this path. Each robot also needs to determine where it is in the configuration space, a task known as localization. This generally requires a robot to perceive and process information gathered by its sensor(s). Additionally, the robot may need to construct a representation of the environment, using the gathered information if the environment is unknown. Last but not least, the robots must also coordinate with each other to determine where each robot should move given an overall task based on the available information about the environment.

Although each of these components deserves substantial attention, the primary focus of this dissertation is to design efficient planning and control strategies that enables robot swarms to address the coverage and mapping problem with minimal resources. The *resource* here refers to the capabilities of the robots in terms of sensing, computing, and sharing information. Ultimately, our goal is to provide a guideline for developing an algorithm to solve more complex problems in robotics using resource-constrained robot swarms. In

the remainder of this chapter, we identify the main challenges in designing algorithms for resource-constrained robots and discuss some potential remedies. We then summarize the remainder of this dissertation which discusses a series of sensor coverage problems and the proposed algorithms to solve them.

1.1 Research Problems

Most of the modern state-of-the-art approaches developed to solve the sensor coverage problem focus on a metric space where the environment is represented by a map constructed by integrating range measurement of the obstacles with the poses of the robots. The pose of the robot is either provided by external sensors such as Global Positioning Systems (GPS) or motion capture systems, or estimated through a combination of internal sensors including accelerometer and gyroscopes (IMU), visual and/or range sensors (Lidar, camera). Although this metric information provides highly accurate and efficient solutions, external sensors are often not available while the integration of internal sensors generally requires high precision sensing information and huge computational resources which are not achievable with low budget sensors and processors.

On the other hand, a topological approach, where information is represented by a set of points without geometric coordinate system, is more suitable for handling a swarm with these types of mobile sensors. In recent years sensor coverage has been studied more formally using *simplicial complex* and *homological tools* from algebraic topology [24, 26, 43]. Such approaches are completely topological and require little to no metric information, which is particularly attractive for our purposes. Nevertheless, there still remains many challenges in the adaptation of these topological approaches to solve the coverage problem, especially in guidance, navigation, and control of these mobile sensors in a coordinate-free system. To this end, the vision-based controllers which have been inspired by biological systems that are capable of utilizing minimal resources to carry out their everyday tasks can be applied. Specifically, these controllers utilize bearing information which can be obtained from the cameras with relatively high accuracy compared to the range information.

1.1.1 Key Contributions

This dissertation presents algorithms for some core problems, namely (sensing) coverage, topological mapping, exploration, and pursuit-evasion, which can be generalized to many other applications in robotics. In each problem, the proposed algorithm is designed to solve such a task using as little sensing capability as possible while still being efficient and complete.

1.2 Thesis Overview

To provide a guideline for developing coverage and mapping algorithms for these resource-constrained robots, this document introduces some notation from algebraic topology and bearing-based controllers in Chapter 2. This chapter also discusses some of the related work on these sensor coverage problems.

The algorithm for deployment of a swarm of mobile robots into an unknown environment to attain complete sensor coverage is presented in Chapter 3. Each robot is only capable of measuring the bearing to other robots within its convex sensing footprint without knowledge of any other metric or global information. Using the same sensing model, an algorithm to construct a topological map - a global description of an environment - is presented in Chapter 4.

Chapter 5 introduces a new sensing model where each robot is capable of identifying a set of unique landmarks in the environment to solve an exploration problem. The proposed approaches apply a frontier-based strategy to a special type of topological map called a *landmark complex*. The first strategy directly constructs a navigation roadmap from the dual of the landmark complex and utilizes an existing bearing-based controller to guide the robots around the environment. Under the assumption of sufficiently dense landmarks, the proposed method could construct a map that encapsulates all topological features of the environment. However, the existing bearing-based controller does not guarantee a collision-free path and requires a knowledge of global orientation to achieve globally asymptotic stability. To address these limitations, we propose a new control policy that utilizes a local

range information obtained from consecutive bearing measurements to navigate the robots around and explore the environment. Additionally, we demonstrate the performance of our proposed method in a more realistic setup, discuss the limitations of landmark-based exploration, and then propose solutions to address some of those limitations.

Next, we consider a variation of the multi-pursuer and multi-evader planning problem where the objective is to clear a given environment from any potential intruders in Chapter 6. We propose a framework that partitions the configuration space into sets of topologically similar configurations that preserve the possible intruder positions and then synthesizes solutions in the reduced space. We first propose a framework to solve the pursuit-evasion problem on a metric map and then explain how to apply similar framework to the topological map such as one introduced in Chapter 5, as its key idea focuses on the topological features of the map which is readily available in the topological map.

Lastly, we conclude this entire thesis as well as discuss possible future directions in Chapter 7.

Chapter 2

Background and Related Work

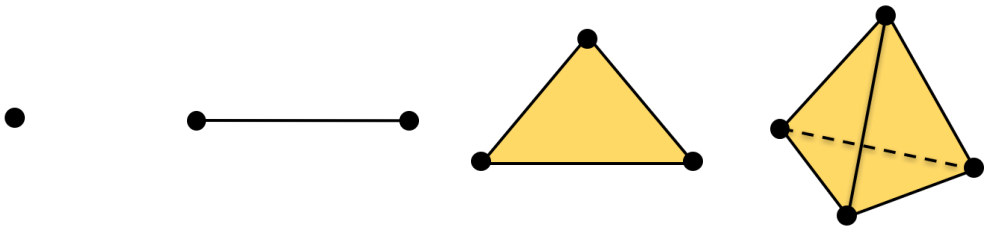
This chapter introduces the tools from algebraic topology and the bearing-based controllers that will be used in this dissertation. It then surveys some of the recent research that is related to the sensor-based coverage and mapping problem discussed in the following chapters.

2.1 Algebraic Topology

This section briefly summarizes the key definitions and results in algebraic topology that are used extensively in this thesis. The reader should refer to the literature [24, 41, 43, 47] for examples and details of the topics discussed in this chapter.

2.1.1 Topological Background

A *topological space* is a nonempty set, $\mathcal{X}$, equipped with a *topology*, a collection of open subsets of $\mathcal{X}$, such that a topology (i) contains both the empty set and the set $\mathcal{X}$ and (ii) remains closed under a finite intersection and arbitrary union of its member. Two topological spaces $\mathcal{X}$, and $\mathcal{Y}$ are then homotopy equivalent if there exists continuous maps $f : \mathcal{X} \to \mathcal{Y}$ and $g : \mathcal{Y} \to \mathcal{X}$ such that $f \cdot g \simeq Id_{\mathcal{Y}}$ and $g \cdot f \simeq Id_{\mathcal{X}}$. One particularly simple class of spaces that we will work with is a *simplicial complex* whose purely combinatorial counterpart is an *abstract simplicial complex*.



(a) 0-simplex **(b)** 1-simplex **(c)** 2-simplex **(d)** 3-simplex

Figure 2.1: Examples of simplices in a simplicial complex.

Simplicial Complex

A simplicial complex, informally speaking, is a higher dimensional generalization of a graph, which not only consists of 0-simplices (vertices) and 1-simplices (edges), but also their higher-dimensional counterparts such as 2-simplices, 3-simplices and so on (illustrated in Figure 2.1). Consequentially, it is possible to create a topologically equivalent piece-wise linear representation of a configuration space of 2 and higher dimensions using a simplicial complex. We start with the formal combinatorial definition of a simplicial complex.

Definition 1 (Simplicial Complex)**.** A simplicial complex, $\mathcal{S}$, constructed over a set V (the *vertex set*) is a collection of sets $C_n,\ n = 0, 1, 2, \cdots$, such that

i. An element in $C_n,\ n \geq 0$ is a subset of V and has cardinality $n+1$ (*i.e.*, For all $\sigma \in C_n, \sigma \subseteq V, |\sigma| = n+1$). σ is called a "*n-simplex*" (*Simplex definition*).

ii. If $\sigma \in C_n,\ n \geq 1$, then $\sigma - v \in C_{n-1},\ \forall v \in \sigma$. Such a $(n-1)$-simplex, $\sigma - v$, is called a "*face*" of the simplex σ (*Boundary property*).

The simplical complex is the collection $\mathcal{S} = \{C_0, C_1, C_2, \cdots\}$.

Two examples of simplicial complexes that have been widely used to convert data in a metric space to a topological space are the Čech complex and the Rips complex [24, 25].

Definition 2 (Čech Complex)**.** A Čech complex, $\mathcal{C}_\epsilon(\mathcal{X})$, constructed over a set of points $\mathcal{X} = \{x_\alpha\}$ in a metric space and a fixed $\epsilon > 0$, is a simplicial complex whose n-simplices correspond to unordered $(n+1)$-tuples of points in $\mathcal{X}$ such that an intersection of $\epsilon/2-$balls around each point is nonempty.

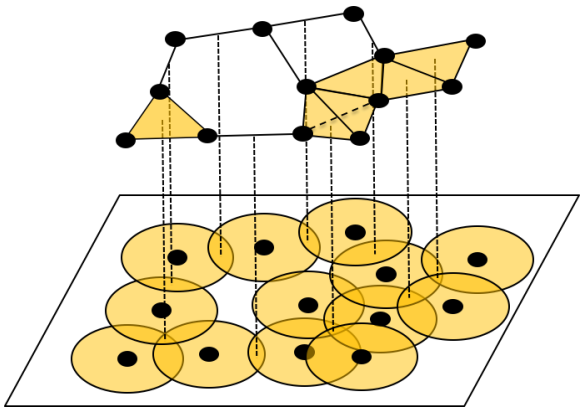


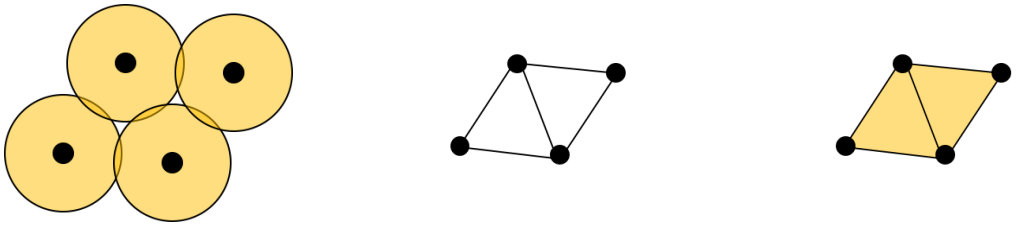
Figure 2.2: The Čech complex correctly approximates the topology of the covered regions (one connected component and two holes).

Given a sensor network where $\mathcal{X}$ represents the positions of all sensors whose sensing footprints are circular disks with radius of $\epsilon/2$, a Čech complex, $\mathcal{C}_\epsilon(\mathcal{X})$, is homotopy equivalent to the regions covered by the sensor network (homotopy equivalence) as illustrated in Figure 2.2. Nevertheless, the Čech complex is generally difficult to compute as checking nonempty intersection requires precise distances between the sensors. As a result, we turn to the Vietoris-Rips complex or simply the Rips complex.

Definition 3 (Rips Complex)**.** A Rips complex $\mathcal{R}_\epsilon(\mathcal{X})$, constructed over a set of points $\mathcal{X} = \{x_\alpha\}$ in a metric space and with a fixed $\epsilon > 0$, is a simplicial complex whose n-simplices correspond to unordered $(n+1)$-tuples of points in $\mathcal{X}$ which are pairwise within distance ϵ of each other.

The computation of the Rips complex is less expensive. Nevertheless, it may not correctly capture the topology of the covered regions due to the geometric properties of the disk as depicted in Figure 2.3.

The topological invariant of a space can be captured by computing the homology of a simplicial complex. Given $\mathcal{S} = \{C_0, C_1, C_2, ...\}$ with d_n being the boundary map from C_n to C_{n-1}, the homology of $\mathcal{S}$ is defined as $H_n(\mathcal{S}) = \ker d_n / \text{im } d_{n+1}$ [47]. The resulting homology groups indicate the topological features of the space. For instance, $H_0(\mathcal{S})$ corresponds to the number of connected components in the simplicial complex while $H_1(\mathcal{S})$ corresponds to the number of holes in the simplicial complex.



(a) A sensor network **(b)** Čech complex **(c)** Rips complex

Figure 2.3: The Rips complex does not always capture the topology of the cover as there are two holes in the sensor network (and in the corresponding Čech complex) but no hole in the Rips complex.

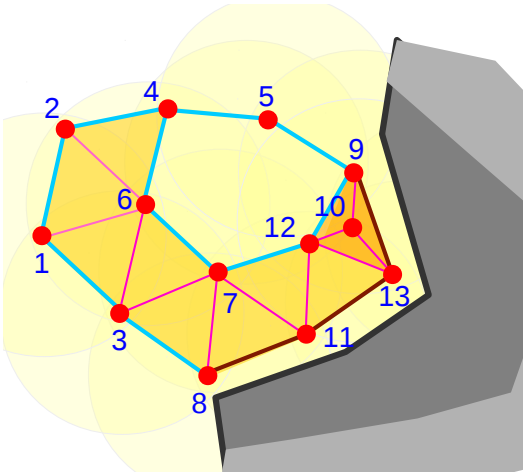
2.1.2 Coverage via Homology

De Silva and Ghrist formally studied the coverage problem in an idealized sensor network using tools from computational homology [24]. They assumed that the Rips complex $\mathcal{R}$ was constructed on a set of sensors $\mathcal{X}$ with the broadcast radius r_b and the cover radius of r_c.

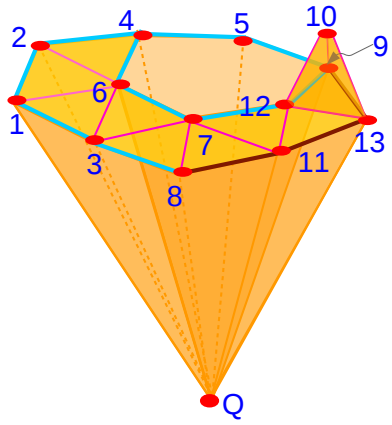
Remark 1. $r_c \geq r_b/\sqrt{3}$ is the optimal ratio such that the topological features of the Rips complex, $\mathcal{R}_{r_b}(\mathcal{X})$, suffice to conclude the geometric properties such as the number of connected components and the number of holes of the union of the cover with radius r_c.

Let $\mathcal{F} \subset \mathcal{R}$ denote a 1-dimensional fence cycle which is the boundary of the covered regions. One can then construct a relative chain complex, $C_*(\mathcal{R}, \mathcal{F})$ for computing the relative homology $H_2(\mathcal{R}, \mathcal{F})$. This chain complex, in essence, is the complex obtained by quotienting out the subcomplex $\mathcal{F}$. This can be viewed as collapsing $\mathcal{F}$ to a single point, or introducing a new 0-simplex, Q, and connecting 2-simplices $\{i, j, Q\}$ to every $\{i, j\} \in \mathcal{F}$ – as illustrated in Figure 2.4b.

Remark 2. The 0-simplices (sensors) making up a nontrivial *relative cycle* in $C_2(\mathcal{R}, \mathcal{F})$ such that it passes through all the 0-simplices in the fence cycle, $\mathcal{F}$, is sufficient for maintaining the sensor coverage.



(a) A Rips complex $\mathcal{R}$ with the fence cycle $\mathcal{F}$, marked in cyan and brown.



(b) Topology of the space where the fence cycle $\mathcal{F}$ is connected to a new 0-simplex Q.

Figure 2.4: An instance where the redundant sensor ($\#10$) can be identified from a nontrivial element of the relative homology $H_2(\mathcal{R}, \mathcal{F})$.

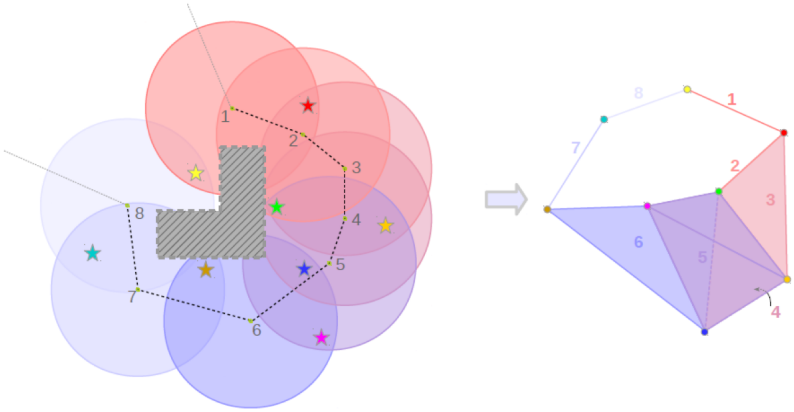


Figure 2.5: *Landmark Complex* (right) is constructed by a robot wandering in an environment (dashed trajectory) and observing the landmarks (colored stars), the landmark complex captures the topology of the workspace (left) correctly where a hole corresponds to an obstacle.

2.1.3 Landmark Complex

Introduced by Ghrist *et al.* [43], the *landmark complex* – a simplicial complex constructed from the observation of landmarks in the environment. The construction of a landmark complex requires that the robots detect the presence of distinguishable landmarks within their respective sensing footprints, but no distance or bearing measurements to the landmarks are necessary.

A *landmark complex*, $\mathcal{L} = \{C_0, C_1, C_2, \cdots\}$, is an abstract simplicial complex constructed over a set of distinct landmarks, $\{L_1, L_2, \cdots\}$, based on observations made by robots with limited-range sensors navigating in the environment, and is described as follows: Every time a robot observes a n-tuple of landmarks, it inserts the corresponding $(n-1)$-simplex in C_{n-1} (along with all its faces and sub-faces in $C_i, i < n-1$), as illustrated in Figure 2.5.

Under certain density assumption on the landmarks and robot observations, a landmark complex is a truthful topological representation of the environment (homotopy equivalence). The landmark density requirement itself is that at least one landmark needs to be visible from every point in the environment. Or, in other words, the *visibility domains* of the landmarks (disks of same radii as the sensing radius, centered at each landmark), $\{\mathcal{V}_j\}$, need to cover the entire workspace of the robots. With this condition, and sufficiently dense sampling of observations by the robots, the landmark complex is identical to the *Čech complex* (or *nerve*) of the visibility domains of the landmarks (since every n-way overlap of the visibility domains, $\mathcal{V}_{i_1 i_2 \cdots i_n} = \mathcal{V}_{i_1} \cap \mathcal{V}_{i_2} \cap \cdots \cap \mathcal{V}_{i_n}$, corresponds to a potential observation of those n landmarks by a robot in $\mathcal{V}_{i_1 i_2 \cdots i_n}$), and hence by the *Nerve theorem*, homotopy equivalent to the workspace W.

Additionally, by Dowker's theorem [27], the dual of landmark complex, denoted by *observation complex* is an abstract simplicial complex, whose vertices are the set of observations, and an $n-$simplex represents the set of $n+1$ observations that share some landmarks in common.

2.2 Bearing-based Controllers

Bearing-based controllers refer to the control policies that rely solely on the bearing measurement towards static landmarks or features to navigate the robot around the environment. These type of controllers have been widely used in the vision-based navigation since cameras yield a relatively good bearing measurement but poor range information (or no range information with single camera). Nevertheless, the existing bearing-based controllers generally require global compass direction to get a globally asymptotically stable solution. On the other hand, if the global compass direction is unknown, there exist solutions that use the relative bearing angles (difference between bearing) [5, 62, 63]. Nevertheless, to the best of the author's knowledge, none of them provide of the proof of global convergence.

In this section, we introduce two bearing-based controllers that will be used in this thesis.

2.2.1 Gradient Field Approach

The bearing-based *visual homing controller*, presented in [94], utilizes a gradient field of a Lyapunov function to control the robots toward the given home location.

Let $x_g \in \mathbb{R}^d$ denote the coordinate of the home location, and $\{x_i\}_{i=1}^N \in \mathbb{R}^d$ denote the locations of N landmarks. For $i = \{1, ..., N\}$, the unit vector representing the bearing direction at point $x \in \mathbb{R}^d$ is defined as

$$\beta_i(x) = \frac{x_i - x}{\|x_i - x\|}, \tag{2.1}$$

or simply β_i. The home bearing direction, β_i^*, is then denoted by

$$\beta_i^* = \beta_i(x_g) = \frac{x_i - x_g}{\|x_i - x_g\|}. \tag{2.2}$$

Let c_i denote the inner product $c_i = \beta_i^{*T}\beta_i$ and $f : \mathbb{R} \to \mathbb{R}$ denote a univariate reshaping

function. The control input u is defined from the gradient of a cost function $\varphi : \mathbb{R}^d \to \mathbb{R}$ as

$$u = -\text{grad}_x \varphi \tag{2.3}$$

$$= \sum_{i=1}^{N} \text{grad}_x \varphi_i \tag{2.4}$$

$$\text{grad}_x \varphi_i = -f(c_i)\beta_i - \dot{f}(c_i)(I - \beta_i \beta_i^T)\beta_i^*. \tag{2.5}$$

For $f(c_i) = 1 - c_i$, the gradient function can be simplified to

$$\text{grad}_x = -\sum_{i=1}^{N} \beta_i + \sum_{i=1}^{N} \beta_i^*, \tag{2.6}$$

which is equivalent to the well-known Average Landmark Vector method [58].

2.2.2 Biologically Inspired Approach

This approach, in contrast to the previous, uses the perpendicular vector to the bearing measurement to create the navigation vector field [64]. Using the same notation as the previous controller, the perpendicular vector of bearing direction, $\beta_i^\perp$, is defined as

$$\beta_i^\perp \triangleq \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \beta_i \tag{2.7}$$

The control law u:

$$u = -\sum_{i=1}^{N} \left(\beta_i^{*T} \beta_i^\perp \right) \beta_i^\perp, \tag{2.8}$$

is globally asymptotically stable. The proof has been shown in the original paper [64].

2.3 Related Work

This section reviews some of the recent findings in literature that are closely related to the problems discussed in this thesis.

2.3.1 Coverage

Coverage of indoor environments using teams of mobile robots is a well-studied problem in robotics. There are many variations of coverage problems. For instance, *coverage path planning (CPP)* refers to the task of visiting every point (or within a certain distance of every point) in a given environment as has been addressed in [1, 37, 97]. This variation of the problem only requires the robot(s) to temporarily cover each region of the environment. Such behavior is highly applicable to the house cleaning robots, one of the most well-known robotic applications. In contrast, the problem of attaining persistent *coverage by a sensor network* is the task of deploying and distributing a team of robots such that they attain and maintain constant sensory coverage of every point in the environment, which is a characteristic required in applications related to environmental monitoring. In the presence of a limited number of robots, this problem has often been handled using the Voronoi partition of the environment and the minimization of a coverage functional [11, 22]. However such approaches invariably rely on a global localization capability for each robot (for example, using GPS). Complete sensor coverage of indoor environments using swarm of robots have been studied in [28, 86], where the known world is modeled as a graph, robots are assumed to have global localization and can be made to navigate independently from one location to another in a global coordinate frame. In almost all of these lines of research, global localization of the robots, a priori knowledge of the environment (obstacle configurations), availability of metric information and ability to control the robots from one point in the environment to another have been assumed.

Biologically inspired multi-robot coverage algorithms have also been proposed [53, 97], which are most often distributed and the robots rely on local sensing only. Similar local communication-based algorithms for robot swarms have been used to construct various shapes [84] in an environment. However, such behavior-based algorithms come with very limited theoretical guarantee. Likewise, distributed coverage algorithm with no global localization have been studied in [8]. But the notion of coverage being based purely on a graph gives limited to no guarantee on the attainment of sensor coverage or the optimality.

Furthermore such approaches inherently assume availability of some metric information.

On the other hand, the topological approach which requires little to no metric information has been recently applied to the coverage of the idealized sensor network in [24] (also previously mentioned in Section 2.1). In general, homology computation has been extremely useful in detecting holes in sensor network coverage. While some progress has indeed been made in controlling the network as to mend the holes in sensor coverage [26, 70], all these methods only work in obstacle-free environments and require some localization of the robots to control them.

At approximately the same period that we developed our algorithm, a similar research study conducted by Lee, *et al.* [61] studies the problem of maximization of coverage area with a limited number of robots equipped with limited local sensors. Similar to our approach, their solution uses bearing measurement only in deploying the team of robots to achieve the maximal coverage of the unknown environment. The proposed method iteratively constructs the triangulation of the explored regions using the local bearing information, which appears to be similar to the simplicial complex. Then, the new robot is deployed to the unexplored regions in a breadth-first search pattern. There are three differences between our approach and their proposed method. First, we use the Rips complex to represent the structure of the explored regions which automatically yields the topological information of the environment. Secondly, our approach can repair the holes caused by the failure of any individual sensors. Lastly, our deployment strategy minimizes the sum of the travel distance square, resulting in a shorter mission time.

2.3.2 Topological Mapping

A topological map (also, a *topological graph*), by definition, is a roadmap [80, 92] which is a sparse representation of the configuration space capturing its topological features such as the connectivity of the free space. A topological map provides information for fast navigation in many human environments (urban/rural environments, homes, shops, office buildings), as well as hazardous environments (collapsed buildings, underground tunnels) and is beneficial for time-critical applications like disaster response and search/rescue. In a resource-

constrained environment and in the absence of global localization or metric information, a topological map not only provides a coarse topological representation of the environment, but also a roadmap for transporting physical robots, equipment or supplies from one point to another.

The topological map that is of particular interest in this dissertation is the Generalized Voronoi Graph (GVG, also called a Generalized Voronoi Diagram or a GVD), which is the locus of points in the configuration space which have more than one distinct "closest" points on the boundary of the configuration space. Given a configuration space, the algorithms for constructing a GVG are well-studied [7, 18, 95]. The practical motivation and applications of a GVG representation of an environment are diverse, including sensor based mapping [92], efficient motion planning [38], and computer graphics [71].

Nevertheless, there has been very little work in literature on how small, resource-constrained sensors and processors can yield global information that is relevant to operation in large-scale environments. Topological representations that are robust to sensor and actuator noise, and have reliable local-to-global integration properties, are an ideal choice when working with resource-constrained robots in GPS-denied environments. Existing works in the topological mapping literature ([52, 55, 56, 92]) require robot(s) equipped with sensors that yield metric information and are able to localize themselves in an inertial frame. However, in the scenario where robots are prone to failure, such as in hazardous environment, the swarms of resource-constrained robots would be preferred due to the higher rate of success.

2.3.3 Exploration

Autonomously exploring an unknown environment is another well-studied problem in robotics as it is one of the most essential task for operations in an unknown environment. This problem encompasses many areas of research in robotics including planning, control, mapping, and localization. The exploration can be carried out using a single robot [98] or a team of robots [13, 36, 99]. While the goal of a single-robot exploration is quite straight forward as minimizing the overall exploration time, the multi-robot exploration problem introduces many variables that can be utilized to achieve the optimal results. Some of the main chal-

lenges for multi-robot exploration include coordination between robots, map merging, and limited communication.

The strategies for exploration can be roughly divided into the frontier-based strategies [13, 36, 99] and the information-based strategies [3, 81, 89]. The frontier-based strategies focus on the coordination between robots to achieve the minimum exploration time while assuming that the robots have perfect localization and mapping capability. On the other hand, the information-based strategies consider the uncertainty in the pose of the robots and the sensors. By utilizing the notions of entropy from information theory, the objective is to minimize the uncertainty of the constructed map and robot's poses by choosing actions that yield high information gain while maintaining low uncertainty. In almost all of these approaches, the map is represented by a probabilistic map that requires some range measurement and the pose estimation of the robots. As a result, these approaches inherently require robots to be equipped with high sensing and computing capabilities. Additionally, the integration of information is challenging with noisy sensors as the uncertainty from each robot will not yield a consistent map.

Another task that is closely related to exploration is the well-known simultaneous localization and mapping (SLAM). Similarly, most state-of-the-art methods that require precise metric information (such as range measurements to obstacles), rely on relatively precise odometry measurements, and require extensive post-processing in order to correct for accumulated errors to build a complete map [4, 14, 17, 23, 29, 69]. Hence the representations of the environment that the robots build is not coordinate-free (e.g., an occupancy grid). The problem becomes even more complex when multiple robots need to build such a map simultaneously since the amount of information that needs to be shared between robots is extremely large and a precise transformation between the robots' local coordinates is difficult to compute [49, 93].

While there has been extensive work in swarm robotics [21], the need for swarm algorithms to be scalable has been the bottleneck in addressing the exploration and mapping problems using large swarms. The focus of swarm literature has thus primarily been on

solving problems such as coverage [30] and estimation [31, 66].

2.3.4 Pursuit-Evasion

The pursuit-evasion problem has been studied extensively from many perspectives including differential game theory [50], graph-based search [54, 73], visibility-based search [40, 45], probabilistic search [72], and sensor placement [2]. Isaacs [50] studied the sufficient and necessary conditions for a pursuer to capture an evader in the scenario where a pursuer and an evader alternatively take turn moving in finite space.

Parsons [73] pioneered the graph theory aspect of pursuit-evasion problem in 1976 by solving the problem of searching for a lost man in a known cave structure, which he represents as a searching on a discrete graph. Evaders reside on the edges and can be adversarial. To detect the evaders, the pursuer must move along the edge occupied by the evaders and touch the evader. Initially all edges are contaminated, and will become cleared if they certainly do not contain any evaders. The edges can also be recontaminated if the evader move back to the cleared edge without being detected. The pursuers' goal is to find a trajectory that clears all edges.

Later in 2007, Kolling and Carpin [54] proposed an algorithm to solve a similar problem called GRAPH-CLEAR, where the goal was to solve for an optimal solution in clearing the graph under special circumstances. Nevertheless, this type of edge-search, where evaders reside on the edges, are not directly applicable to most robotics applications, especially in the unstructured environment where graph construction is non-trivial. In more recent work, Hollinger et al. [48] discussed an algorithm called GSST for adversarial search where evaders reside on the node. Nevertheless, GSST still suffers from the graph construction in the unstructured environment and the solution is limited to a tree structure, which may solve the non-tree structure by placing some stationary pursuers to break up the cycles.

On the other hand, Guibas et al. [45] and Gerkey et al. [40] formulated the problem as searching in a continuous space where each pursuer has infinite line of sight and the goal of the pursuers is to see all possible evaders. Their method includes partitioning a free space based on a critical point and then applying a graph-based searching technique. Nevertheless,

their approach only works with very few pursuers and the critical point technique which is adapted from a trapezoidal decomposition is limited to a 2-dimensional environment.

An alternative formulation of the pursuit-evasion is considered in the probabilistic search by relaxing the worst-case scenario with the model of evaders or some uncertainty. Ong et al. [72] proposed an approximate sampling-based algorithm to solve the pursuit-evasion as a Partially Observable Markov Decision Process (POMDP) problem.

In sensor network, Adams and Carlsson [2] use tools in algebraic topology combined with information about the affine structure of covered region to determine the sufficient and necessary condition for the existence of an evasion path given the positions of each mobile sensor in the space-time dimension of 2D environment. Later, Ghrist and Krishnan [42] generalize the previous criterion to an arbitrary dimension without information regarding the affine structure.

Chapter 3

Sensor Coverage

Coverage by a sensor network is the task of deployment and distribution of mobile sensors such that they attain and maintain constant sensory coverage of every point in the environment as illustrated in Figure 3.1. This task is useful in a wide range of applications such as health monitoring, traffic monitoring, environmental risks monitoring, and prevention of disasters. These monitoring tasks often involve a large number of sensors with limited computational and communication capabilities. In the event that the map of an environment is not available a priori, a sensor network (or a swarm of robots) also needs to explore and learn about an environment in order to complete the global mission. In this chapter, we focus on the problem of efficiently exploring an unknown indoor environment with a rapidly expanding swarm of robots with limited and noisy local sensing with no global localization or sensing capabilities.

This chapter makes two primary contributions. First, we design a distributed control algorithm for deploying the robots to attain full coverage of the entire environment using their disks of visibility. That means, at least one robot should be able to *see* each and every point in the environment after the coverage is attained. However, the robots only have information regarding the bearing to their neighbors in their respective local coordinate frames and a directional touch information with obstacles. This means that there is absolutely no range information or any localization capability available. Second, we present an experimental

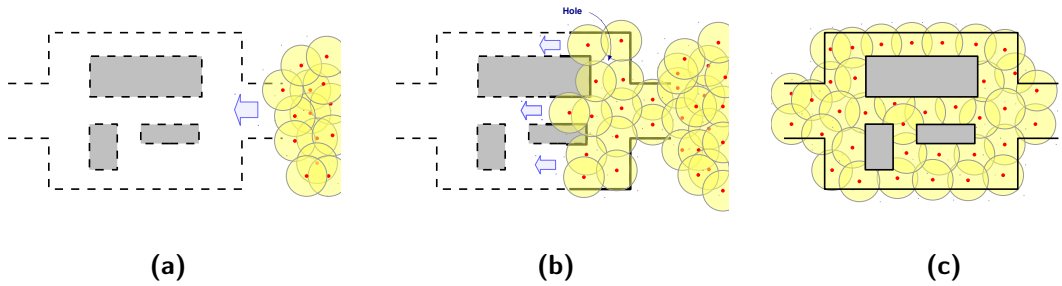


Figure 3.1: Illustration of a swarm of robots entering an environment (a) and attaining full coverage (c). The hole shown on figure (b) is something we would like to avoid.

platform involving a heterogeneous team of real and virtual robots for demonstrating the proposed algorithm.

The research contained in this chapter was originally published in [76] and part of [78].

3.1 Preliminaries

We denote by $\mathcal{W} \subset \mathbb{R}^2$ the obstacle-free region where the robots are being deployed and the sensor coverage of which needs to be attained. If there are n robots deployed in $\mathcal{W}$, we assign IDs to them, $1, 2, \cdots, n$, and represent their joint configuration by $\mathcal{X} = [x_1, x_2, \cdots, x_n]$, $x_i \in \mathcal{W}$.

A robot i, is equipped with an omnidirectional camera that can measure the bearing to a neighbor, $j \in N_i$, in its local coordinate frame, where $N_i = \{j \mid \|x_i - x_j\| \leq r_v\}$ are the neighbors of i. We call this measurement $\theta^i_j \in [-\pi, \pi)$. If it measures the bearing to another robot, k as θ^i_k, then we define $\theta^i_{jk} = \Big((\theta^i_k - \theta^i_j) \bmod 2\pi\Big) - \pi$ — *i.e.*, the bearing to k relative to the bearing to j (and the angle converted to a value in $[-\pi, \pi)$).

Additionally, each robot is also equipped with a collection of touch/contact sensors at the base in order to detect the obstacles. The touch sensors are binary sensors, and are triggered when in contact with an obstacle/wall or another robot (Figure 3.2). The presence of multiple touch sensors (N_T) at the base also provides a rough estimate of the direction of contact (within an error of $\tau = \frac{\pi}{N_T}$ when a single touch sensor is activated).

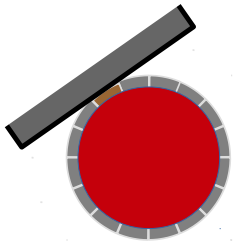
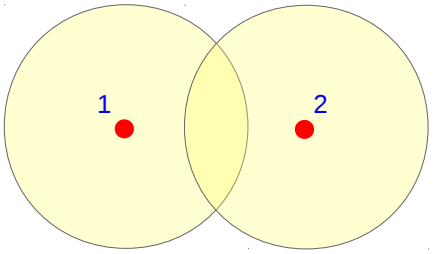
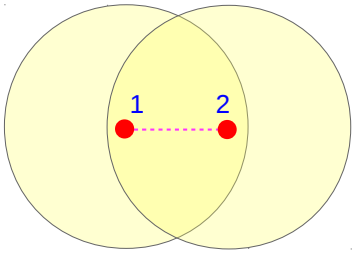


Figure 3.2: The touch/contact sensors (gray protrusions) at the base of a robot (red). Contact with an obstacle or another robot triggers one or more touch sensors providing a rough estimate of the direction of contact.



(a) Robots can't see each other, and hence have no way of detecting that their disks of visibility overlap.



(b) Robots can see each other, and hence know that their disks of visibility overlap. Visibility is represented by the dotted magenta line.

Figure 3.3: Detection of overlap of *disks of visibility* using only local visibility-based sensing.

3.2 Rips Complex of Visibility Disk

We do not assume that the robots can localize themselves and the only way of sensing/identifying neighbors is by using the equipped camera. Thus, if the disks of visibility of two robots merely overlap, there is no way of detecting that fact (Figure 3.3a). We need to use a *stronger notion of overlap* — two robots know that their disks of visibility overlap if and only if they are in each other's disks of visibility and their line of sight is not blocked (Figure 3.3b). Using the IDs of their neighbors, we can then construct a simplicial complex that is homotopy equivalent to the Rips complex over a set of X with $\epsilon = r_v$ as described in Section 2.1. According to Remark 1 in 2.1.2, the Rips complex $\mathcal{R}_{r_v}(X)$ can capture all topological features of the union of the regions covered by the swarm of robots with cover radius $r_c = r_v$. Note that we do not construct any 3 or higher dimensional simplices since

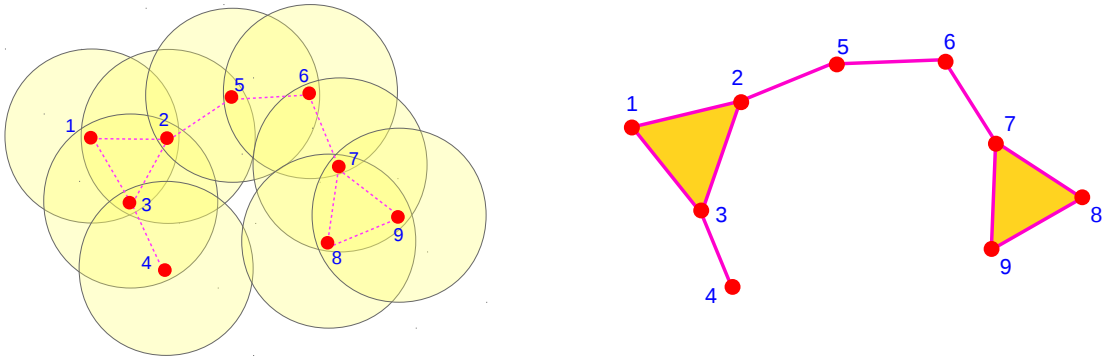


Figure 3.4: Nine robots, their disks of visibility and the corresponding abstract simplicial complex, $\mathcal{R}_{r_v}$.

on a planar environment with obstacles, we are only concerned with the H_1 homology.

In contrast to [24], the fence subcomplex (or fence cycle), $\mathcal{F}$, is not known a priori and will be identified as the swarm expands. Due to the present of obstacles and the process of exploration, the fence subcomplex will be further divided into the frontier subcomplex $\mathcal{K}$ and the obstacle subcomplex $\mathcal{O}$, $\mathcal{F} = \mathcal{K} \cup \mathcal{O}$.

Additionally, we periodically compute a nontrivial 2-cycle of the relative complex $(\mathcal{R}_{r_v}, \mathcal{F})$ so that we can identify redundant/extra robots that can be removed from the complex without sacrificing sensor coverage (see Remark 2 in 2.1.2). It is important to note that $\mathcal{R}_{r_v}$ can be constructed with local visibility information only, as described earlier, and does not require the entire configuration, X, of the robots to be known in a centralized manner. Furthermore, as will be evident in the next section, we do not need to construct the entire simplicial complex in a centralized fashion for most of the algorithmic components. It's only when we compute generators for the relative homology $H_2(\mathcal{R}_{r_v}, \mathcal{F})$, for optimization purposes, that we will need to store $\mathcal{R}_{r_v}$ in a centralized manner.

3.3 Algorithmic Designs

The outline of our swarm coverage algorithm is presented in Algorithm 1. We begin by deploying robot 1 into the unknown environment using an open-loop control so that it maintains visual contact with the source/base. Then, in line 3, we start our deployment cycle by constructing the Rips complex in a distributed manner as described in Section 2.1.1 (and

Algorithm 2). Using the current Rips complex $\mathcal{R}_{r_v}$, we update the frontier, $\mathcal{K}$, and obstacle, $\mathcal{O}$, subcomplexes (line 4, and Algorithm 3) along with computing the target location for deployment of the new robot in the local coordinates of a frontier robot. Note that at the end of the first deployment, the robot 1 belongs to $\mathcal{K}$ (as described in Section 3.3.1). Although our implementation uses a centralized server that collects local information from the individual robots through an emulated wireless communication channel, most of the algorithmic components described in this section can be done in a decentralized fashion.

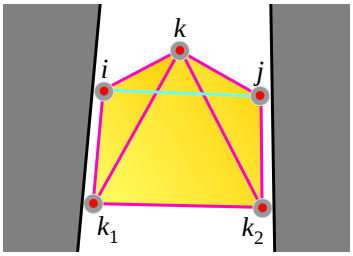
We periodically compute relative H_2 homology to identify redundant robots for redeployment (line 5). We then find the shortest path to a frontier robot (from the source or a redundant robot) through the 1-skeleton of the complex (line 6), along which we execute the "push" action (described in Section 3.3.2) for deployment of the next robot (line 7). In presence of multiple sources, deployment can be performed in parallel along multiple paths as long as the paths do not intersect (which can be computed using an optimal routing algorithm).

Algorithm 1 Swarm Coverage Overview

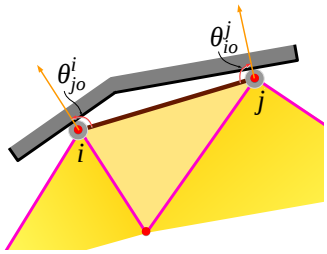
1: Deploy Robot 1; $n \leftarrow 1$
2: **do**
3: Construct Rips cplx. through local communication:
$\mathcal{R}_{r_v} = \textsc{ComputeRipsComplex}(\{N_i\}_{i=1,2,\cdots,n})$
4: Compute fence subcplx. using local bearing info.:
$[\mathcal{K}, \mathcal{O}] = \textsc{FenceSubcomplex}(\mathcal{R}_{r_v}, \{\theta^a_{bc}\})$
5: (Periodically) Identify redundant robots using H_2 hom.
6: Find *path* in 1-skeleton for "Pushing" robots
7: "Push" robots in *path*
8: Deploy $(n+1)^{th}$ robot; $n \leftarrow n+1$
9: **while** $\mathcal{K} \neq \emptyset$

3.3.1 Identifying Frontier and Obstacle Subcomplexes

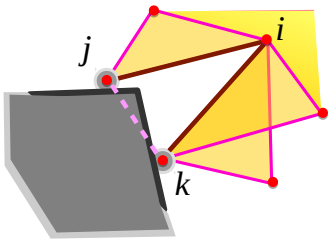
At a particular robot configuration, X, we identify the 1-simplices (and the corresponding 0-simplices that constitute) in the Rips complex, $\mathcal{R}_{r_v}$, which form the *frontier* to the unexplored regions, as well as the ones that are adjacent to the obstacles. They respectively constitute the frontier subcomplex, $\mathcal{K}$, and the obstacle subcomplex, $\mathcal{O}$. We define the *fence subcomplex* as $\mathcal{F} = \mathcal{K} \cup \mathcal{O}$.



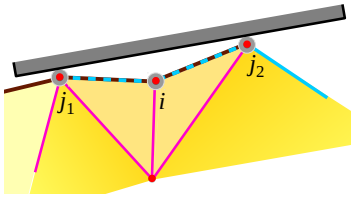
(a) Exception case where a 1-simplex, $\{i, j\}$, has all adjacent 1-simplices lying on the same side, but is not a fence simplex. This can be detected from the perspective of robot k.



(b) Detecting that a 1-simplex, $\{i, j\}$, is in $\mathcal{O} \subseteq \mathcal{R}_{r_v}$ (thick brown line).



(c) Convex corner case where a pair of 1-simplices, $\{i, j_1\}$ and $\{i, j_2\}$, are recognized as obstacle 1-simplices (thick brown lines).



(d) If the robot i is to be "pushed" along a path in the graph to expand frontier $\{i, j_1\}$, it performs a "test drive" to ensure an obstacle is not right in front of it.

Figure 3.5: Identifying simplices for fence subcomplex $\mathcal{F} = \mathcal{K} \cup \mathcal{O}$.

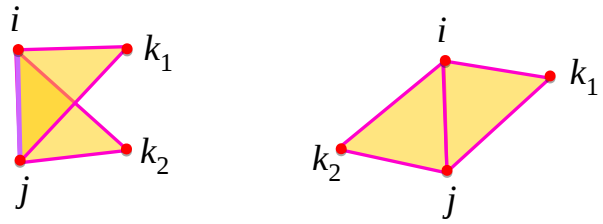


Figure 3.6: $\{i, j, k_1\}$ and $\{i, j, k_2\}$ are two 2-simplices which have $\{i, j\}$ in their boundaries. $\{i, j\}$ is a fence 1-simplex if both k_1 and k_2 lie on the same side of $\overline{ij}$ (thick purple line in left figure), otherwise not (right figure).

Algorithm 2 Compute the Rips complex from lists of neighbors

Input: Neighbor information, $N_i,\ i = 1, 2, \cdots, n$
Output: Rips complex, $\mathcal{R}_{r_v}$

```
1: function ComputeRipsComplex(\{N_i\}_{i=1,2,\cdots,n})
2:     \mathcal{R}_{r_v} \leftarrow \emptyset
3:     for Robot i = 1, ...n do
4:         \mathcal{R}_{r_v} \leftarrow \mathcal{R}_{r_v} \cup \{i\}
5:         for Robot j \in N_i do
6:             \mathcal{R}_{r_v} \leftarrow \mathcal{R}_{r_v} \cup \{i, j\}
7:             for Robot k \in N_i do
8:                 if j \neq k and j \in N_k then
9:                     \mathcal{R}_{r_v} \leftarrow \mathcal{R}_{r_v} \cup \{i, j, k\}
10:                end if
11:            end for
12:        end for
13:    end for
14:    return \mathcal{R}_{r_v}
15: end function
```

Algorithm 3 describes our method of identifying the frontier subcomplex and obstacle subcomplex. We begin (line 3) by identifying the 1-simplices in $\mathcal{R}_{r_v}$ that are part of an exception set, E, as described later in Section 3.3.1 (Figure 3.5a). For each 1-simplex $\{i, j\}$ in $\mathcal{R}_{r_v}$ that is not in E, we then compute the sign of $\theta^i_{jk_u}$ for all of the 2-simplices $\{i, j, k_u\} \in \mathcal{R}_{r_v}$ (*i.e.*, the ones which have both i and j as their vertices).

If all the 2-simplices adjacent to $\{i, j\}$ lie on the same side of the 1-simplex (Figure 3.6), then the bearing angle to all the robots k_u relative to j (resp. i) have the same sign, and thus in line 5, UnCov_{ij} is not empty. Thus, i, j belongs to the fence subcomplex, and we compute and store the location (in the local coordinates of i and j) for potentially deploying a new robot to expand the frontier using a "pushing" action (line 7). The exact computation of the bearings to the potential new location, $\theta^i_{j,new}, \theta^j_{i,new}$, is described in Section 3.3.1 and Algorithm 4.

Finally, in lines 8-12, we classify each fence simplex, $\{i, j\}$, as *frontier* or *obstacle* using touch sensor readings and the outputs of DeploymentAngle as follows:

i. If i and j are in contact with an obstacle (*i.e.*, a touch sensor is activated, and there are no robots visible in the direction of the activated touch sensor) in the expanding direction, then the 1-simplex $\{i, j\}$ and 0-simplices i, j are placed in $\mathcal{O}$ (Figure 3.5b).

Algorithm 3 Identify Frontier and Obstacle subcomplexes

Input: Rips complex, $\mathcal{R}_{r_v}$; Relative bearings, $\theta^a_{bc}, \{a,b,c\} \in \mathcal{R}_{r_v}$
Output: Frontier subcomplex, $\mathcal{K}$; Obstacle subcomplex, $\mathcal{O}$

function FENCESUBCOMPLEX($\mathcal{R}_{r_v}, \{\theta^a_{bc}\}$)
  $\mathcal{K} \leftarrow \emptyset$, $\mathcal{O} \leftarrow \emptyset$
  $E \leftarrow$ COMPUTEEXCEPTION($\mathcal{R}_{r_v}$)
  for $\{i,j\} \in \mathcal{R}_{r_v} \setminus E$ **do**
    $\text{UnCov}_{ij} \leftarrow \{+1,-1\} \setminus \{\text{sign}(\theta^i_{jk_u}) | \{i,j,k_u\} \in \mathcal{R}_{r_v}\}$
    if $\text{UnCov}_{ij} \neq \emptyset$ **then** ▷ A side of $\{i,j\}$ is uncovered.
      $[\theta^i_{j,new}, \theta^j_{i,new}] \leftarrow$ DEPLOYMENTANGLE(i,j,UnCov_{ij})
      if $\{i,j\}$ is an obstacle simplex **then**
        $\mathcal{O} \leftarrow \mathcal{O} \cup \{\{i\},\{j\},\{i,j\}\}$
      else if $\{i,j\}$ is a fronter simplex **then**
        $\mathcal{K} \leftarrow \mathcal{K} \cup \{\{i\},\{j\},\{i,j\}\}$
      end if
    end if
  end for
  return $[\mathcal{K}, \mathcal{O}]$
end function

ii. Otherwise, we check for possibility of $\{i,j\}$ being an obstacle simplex at a convex corner as follows: We compute the "closest" other fence 1-simplex attached to i and j (this is computed as a part of the DEPLOYMENTANGLE procedure – say it is $\{i,k\}$). If the magnitude of the angle between $\overline{ik}$ and $\overline{ij}$ is less than $\frac{\pi}{3} - 2\beta$ (figure 3.5c, where β is the error in measurement of bearings to neighbors), then the two robots, j and k, do not see each other due to occlusion by an obstacle, but every free point (points outside obstacles) in their convex hull is in the visibility disk of at least one robot (aside from possibly small non-convex sub-features present in that convex corner, which we ignore). Thus, these 1-simplices are marked as obstacle 1-simplices to be added to $\mathcal{O}$.

iii. Otherwise, at least one of the robots can be expanded/moved to the unexplored region, and thus $\{i,j\}$ is placed in $\mathcal{K}$ along with the corresponding robots (Figure 3.6, left).

iv. Since the obstacles can only be detected by touch sensors, we perform a "test drive" in the planned deployment direction for a small distance to ensure sufficient space availability for new deployment near obstacles (Figure 3.5d).

The complete illustration of the process of identifying 1-simplices as part of $\mathcal{K}$ or $\mathcal{O}$ is given in Figures 3.6 and 3.8a. We next describe the COMPUTEEXCEPTION and DEPLOYMENTANGLE

procedures.

The Exception Case

The aforesaid approach in detecting fence 1-simplices using UnCov_{ij} may give false positives in some cases when a 1-simplex, $\{i, j\}$, is completely covered by 2 simplices, of which $\{i, j\}$ do not form a boundary, as shown in Figure 3.5a. Nevertheless, this special case can be easily detected from the perspective of a common neighbor, k, of i, j. If it is detected that $\theta^k_{ij} = \theta^k_{ik_1} + \theta^k_{k_1k_2} + \cdots + \theta^k_{k_rj}$ (for some $k_1, \cdots, k_r \in N_k$), such that all the summands have the same sign as the summation, then clearly $\{i, j\}$ lies inside 2-simplices of which $\{i, j\}$ do not form a boundary but k is a vertex. Then $\{i, j\}$ is marked as an *exception* 1-*simplex.*

Identifying Locations for Robot Placement (Triangular Tessellation)

Given a 1-simplex $\{i, j\} \in \mathcal{K}$ and the uncovered direction $\sigma \in \{+1, -1\}$, we need to find, in the local coordinates of i and j, the location for the new robot position. Figure 3.7a illustrates the uncovered side of 1-simplex $\{i, j\}$ in i's local coordinate. Our strategy for choosing the position to deploy next robot is to try and achieve a *triangular tessellation* [15] (which is the most optimal packing on an obstacle-free plane) of robots as much as possible, only to be interrupted by the presence of obstacles or control's error. This essentially boils down to sending robots at an angle of $60°(= \frac{\pi}{3})$ with respect to $\overline{ij}$ into the free region. Algorithm 4 describes our DeploymentAngle function which first determines (lines 4-7) the "closest" other fence 1-simplices attached to i and j (*e.g.*, $\{i, k\}$ in Figure 3.7b). If there is no other fence 1-simplex attached to i, we set $\theta^i_{\text{new}} = \theta^i_j + \sigma^i_j \frac{\pi}{3}$ — the 60° angle for deployment in a triangular tessellation. Otherwise we set the angle to the minimum between the one for triangular tessellation ($\frac{\pi}{3}$) and the one that bisects $\theta^i_{jk_i}$. Likewise for θ^j_{new}.

If i is not attached to a frontier 1-simplex (*e.g.*, i is a frontier robot in a narrow passage with a single file of robots), then we simply choose the direction *away* from the neighbors of i as the bearing to the new location (in the local coordinates of i) for deployment of the new robot.

Algorithm 4 Identify the Location for Robot Placement

Input: Robots i, j; the *side* of $\overline{ij}$ that is open/uncovered
Output: New location for deployment in local coordinates of i, j OR $\{i, j\}$ is marked as an obstacle simplex.

function DEPLOYMENTANGLE(i, j,UnCov$_{ij}$)
  $\theta^i_{j,new} \leftarrow \emptyset$, $\theta^j_{i,new} \leftarrow \emptyset$
  for σ in UnCov$_{ij}$ **do**
    $S_i \leftarrow \{l \mid \{i, l\} \in \mathcal{R}_{r_v}$ and $\text{sign}(\theta^i_{j,l}) = \sigma\}$
    $k_i \leftarrow \text{argmin}_{k' \in S_i} |\theta^i_{j,k'}|$
    $S_j \leftarrow \{l \mid \{j, l\} \in \mathcal{R}_{r_v}$ and $\text{sign}(\theta^j_{i,l}) = -\sigma\}$
    $k_j \leftarrow \text{argmin}_{k' \in S_j} |\theta^j_{i,k'}|$
    if $|\theta^i_{j,k_i}| < \frac{\pi}{3}$ (or $|\theta^j_{i,k_j}| < \frac{\pi}{3}$) **then**
      Mark $\{i, k_i\}$ (or $\{j, k_j\}$) as an obstacle simplex.
    else
      $\theta^i_{j,new} \leftarrow \sigma \min\{\frac{\pi}{3}, |\frac{\theta^i_{j,k_i}}{2}|\}$
      $\theta^j_{i,new} \leftarrow -\sigma \min\{\frac{\pi}{3}, |\frac{\theta^j_{i,k_j}}{2}|\}$
    end if
  end for
  return $[\theta^i_{j,new}, \theta^j_{i,new}]$
end function

3.3.2 Identifying Path in 1-skeleton for "Pushing" Robots

The strategy in our algorithm for robot deployment in every control cycle is to keep the structure of the existing simplicial complex (and hence the positions of the existing robots in W) unchanged. New robots are deployed through the complex simply by *"pushing"* through paths (*i.e.*, making each robot on a path move forward to take the place of the one in front of it) in the 1-skeleton (graph) of the complex (Figure 3.8). For computing this path, a centralized knowledge of the entire graph is used (constructed by the robots

(a) The free side of $\{i, j\}$ where $\text{sign}(\theta^i_{j,new}) = \sigma$ **(b)** The bearing to the new location, $\theta^i_{j,\text{new}} = \min\{\frac{\pi}{3}, \frac{\theta^i_{jk}}{2}\}$, in i's local coordinate.

Figure 3.7: Determining bearing to the new location

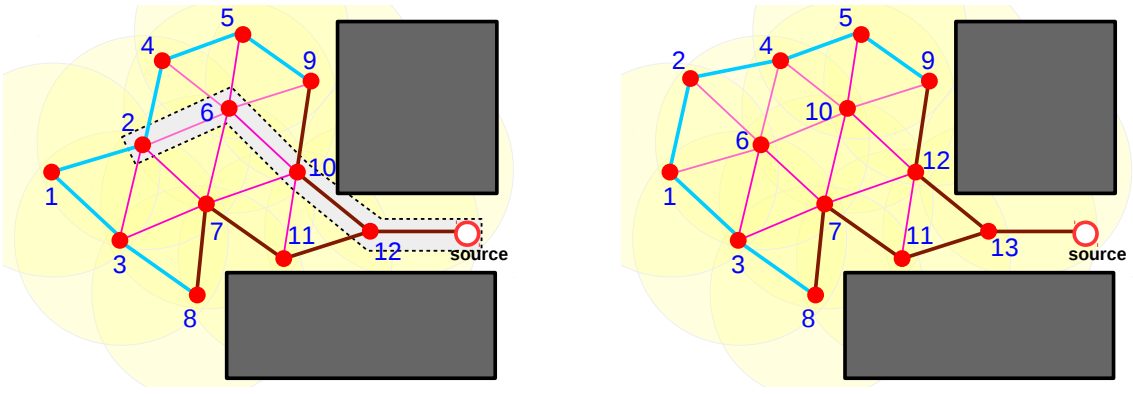
communicating each of their local information – the IDs of the neighbors that each see – to a central server via wireless communication), although the computation of the path can indeed be performed in a decentralized manner through peer-to-peer communication only (see [87] for a decentralized implementation of the Dijkstra's algorithm).

We consider the graph made out of the 1 and 0 simplices in $\mathcal{R}_{r_v}$. The frontier subcomplex, $\mathcal{K}$, computed in previous section (the 0-simplices in it) provides the list of robots which we need to potentially move to *expand* the frontier. We assign a cost of 1 to all the 1-simplices in the graph, except the 1-simplies in $\mathcal{O}$, to which we assign cost of $w_O > 1$ in order to avoid paths that pass through robots adjacent to obstacles, where navigation is more challenging. We use Dijkstra's search algorithm to find the shortest path from the source, which is the robot next to the base station (in case of multiple source, we can initiate the *open list* in Dijkstra's algorithm with the multiple sources as illustrated in [11]), to the closest vertex (0-simplex) in $\mathcal{K}$.

Robots are then "pushed" along this path where each robot on the path simply gets replaced by the one behind it on the path, while the robot that is on the frontier computes (as described next) and moves to a new location in the free/unexplored region. Since robots on the path get *replaced* by the robots behind them, this requires that we not only update the IDs in $\mathcal{R}_{r_v}$, but also the robot IDs in $\mathcal{K}$ and $\mathcal{O}$.

We use the visual homing controller described in Section 2.2.1 (Figure 3.9a). For a frontier robot, i, the desired bearings $\theta^i_{j,\text{des}}$ can be computed easily for the planned direction for deployment of the new robot (θ^j_{new} in the current coordinate frame of robot i), and assuming that the robots are separated by a distance of $r_v - \epsilon$. For every other robot, i', on the path through which robots are being "pushed", $\theta^{i'}_{j,\text{des}}$ are the current bearing values for the robot ahead of i' in the path (with correct ID re-orderings performed).

When robots are being "pushed" along a path, multiple robots move simultaneously, and for a robot moving on the path, some of its landmarks (*i.e.*, neighbors) are themselves moving. The bearing-based controller that we use, is in fact capable of dealing with moving landmarks, and give similar convergence properties. A few static landmarks (at least two



(a) Shortest path $12 \to 10 \to 6 \to 2$ identified from the source to a vertex in $\mathcal{K}$.

(b) Robots are "pushed" along the path. Notice how the new robot 13 appears near the source.

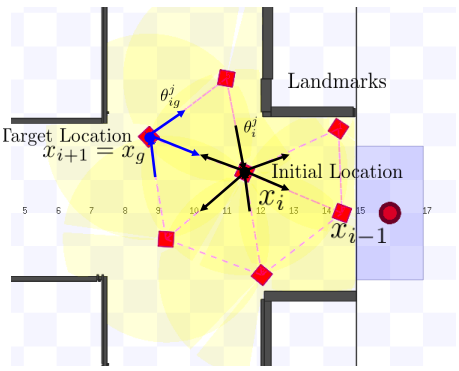
Figure 3.8: The complex $\mathcal{R}_{r_v}$, and the subcomplexes $\mathcal{K}$ (cyan) and $\mathcal{O}$ (brown). Path through the 1-skeleton illustrate *"pushing"*

in total) referenced by some of the moving robots on the path are sufficient in attaining convergence. In addition, the desired bearing is set for each robot for all of their surrounding neighbors. This allows robots to adaptively correct their trajectory while simultaneously gaining and loosing landmarks along their trajectory.

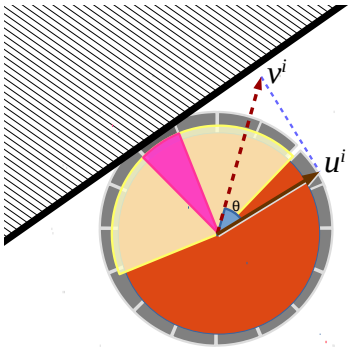
No robots reference the robot that is moving to a new (unexplored) location for expanding the frontier. This is because there are uncertainties about the unexplored region (*e.g.*, about presence of obstacles), and errors due to that should not propagate upstream. Furthermore, if a robot does not have more than one another robot to reference to as landmark, it employs an open-loop control to reach the desired location using odometry estimate, and drives back in case it loses the single visual link that it had. This is unavoidable when, for example, the robots move in a narrow passage in a single file.

Action on Touching an Obstacle

Upon touching an obstacle at a bearing of $\theta_o^i \pm \tau$ (τ being the resolution in the measurement of bearing to touch), the robot will not be able to progress in the direction between $(\theta_o^i - \frac{\pi}{2} + \tau, \theta_{jo}^i + \frac{\pi}{2} - \tau)$ (Figure 3.9b). Hence, if the command velocity, v^i, computed using the bearing-only controller "pushes" the robot inside an obstacle, we take the best projection of that velocity into the set of allowed velocities (u^i in the figure, falling inside the brown



(a) The bearing-based controller uses neighbors as landmarks and use the bearing angles to them to navigate to the desired location knowing the desired bearing angles.



(b) Upon touching an obstacle, the robot use the component of the computed velocity that is the projection in the valid/free sector (brown).

Figure 3.9: Components of the controller.

sector) such that using u^i as the velocity command the robot moves out toward the obstacle-free area freeing itself from the obstacle. Overall, this results in a behavior akin to sliding along the obstacle using the component of the velocity parallel to the obstacle.

Scale Correction

Since our controller is purely bearing-based, and although we attempt to create a hexagonal packing, small accumulation of the errors can decrease the average separation between the robots as we move further away from the source. To correct this, we perform a scale correction periodically, where we make a frontier robot, i, move forward keeping the reference robots behind it (opposite to a mean bearing to those robots), until it breaks visual link with at least one of those neighboring robots. Then we make the robot i drive back until it reestablishes the visual link with all its neighbors. This ensures that the average separation between the robots stay close to r_v.

3.3.3 Identification and Reallocation of Redundant Robots

Using the results from [24] as discussed in Section 2.1.2, we can identify redundant robots in the complex by computing a generator (non-trivial relative cycle) of the relative homology $H_2(\mathcal{R}_{r_v}, \mathcal{F})$, where $\mathcal{F} = \mathcal{K} \cup \mathcal{O}$ is the *fence* subcomplex.

We use the JavaPlex [90] library for the computation of the non-trivial relative cycle

using persistence algorithm. The required filtration over $\mathcal{R}_{r_v}$ is achieved by inserting the 0, 1 and 2 dimensional simplices in sequence. We add a disjoint 0-simplex, Q, as illustrated in Figure 2.4b, and construct the 1-simplices $\{i, Q\}$ and the 2-simplices $\{i, j, Q\}$, for every 0-simplex, i, in $\mathcal{F}$, and every 1-simplex, $\{i, j\}$, in $\mathcal{F}$. Call this new complex $(\mathcal{R}_{r_v}, \mathcal{F})$. Computation of persistent homology up to dimension 2 for this complex with $\mathbb{Z}_2$ coefficients using JavaPlex gives us a set of non-trivial generating 2-cycles in $(\mathcal{R}_{r_v}, \mathcal{F})$ which generate $H_2(\mathcal{R}_{r_v}, \mathcal{F})$. Any non-zero linear combination (in $\mathbb{Z}_2$ coefficients) of these cycles will also be a valid non-trivial 2-cycle which can be used to identify the robots that are sufficient for maintaining coverage. Thus we perform a greedy search for the *best* linear combination (a linear combination of cycles such that it contains the least number of 0-simplices) that also contain all the fence 0-simplices.

Thus, finally we have a set of 0-simplices which constitute robots that are sufficient to maintain the sensor coverage that is currently being maintained. All other robots are redundant and can be removed/reallocated. Once we have identified the redundant robots, in the next deployment cycle we use them, instead of deploying new ones from the source.

3.4 Results

3.4.1 Guarantees

Since throughout the deployment and covering process we keep the graph (1-skeleton of $\mathcal{R}_{r_v}$) of the already-covered region fixed (we only *"push"* robots along paths in the graph to the frontiers), we eliminate the possibility that the algorithm gets stuck in an infinite cycle in which the graph keeps cycling/switching between two configurations. Furthermore, by choosing to keep the graph structure fixed across deployment cycles, we eliminate the possibility that our control algorithm results in recession of a frontier or opens up a new hole in the already-covered region of the environment. If due to accumulation of errors we do open up a hole, and hence a new set of frontier 1-simplices appear, we send robots to those frontier 1-simplices to fill the hole.

Algorithm Termination: The algorithm, as described, will keep deploying robots to

frontier 1-simplices as long as they exist. In absence of obstacles nearby, a robot will be deployed for every frontier 1-simplex at an angle of 60° with the simplex into the uncovered region. This will always make the frontier progress (as illustrated in Figure 3.8). Although this may result in deployment of redundant robots (which are later identified and removed using the relative H_2 homology generator computation), the progress in the expansion of the frontier is always finite in obstacle-free regions. Nevertheless, when the expanding location lies inside an obstacle we need to consider, and thus avoid, the possibility that robots are deployed indefinitely to a region close to an obstacle because the unexplored region changing only infinitesimally at each deployment. This is however prevented by the design of our algorithm, as described in Section 3.3.1 item 'iv.', where we prevent the deployment of unnecessary robots near obstacles that make little to no progress in expanding the frontier. Thus, in a finite environment the algorithm will terminate with no more frontiers left for exploration.

Completeness: As described, when $\mathcal{K}$ is not empty, more robots will be deployed to close the frontier. When $\mathcal{K}$ is empty, then one can observe that for every 2-simplex $\{i, j, k\}$ in $\mathcal{R}_{r_v}$, the convex hull of the robots i, j and k will be covered by the disk of visibility for each of these robots. If $\{i, j\}$ is a 1-simplex in $\mathcal{O}$, then due to the way we introduce elements in $\mathcal{O}$ (Section 3.3.1), the region between the obstacle 1-simplices and the actual obstacles themselves will always remain covered by some robot's disk of visibility (except for non-convex features on the side of the obstacles that are smaller than r_v). Thus, when $\mathcal{K}$ is empty, we can guarantee sensor coverage of the entire environment.

Robustness to Robot Failure: The proposed algorithm can adapt to failure of robots. If a robot fails (and its neighbors can detect that), the swarm will ignore the presence of the failed robot. Thus a "hole" in the complex gets created, and hence $\mathcal{K}$ will have the frontier simplices surrounding that hole. This will result in new robot(s) being deployed to the newly open frontier, until $\mathcal{K}$ becomes empty once again.

Optimal Coverage: While our deployment algorithm itself does not guarantee optimality, the process of identifying redundant robots by computing the smallest non-trivial relative cy-

cle in $(\mathcal{R}_{r_v}, \mathcal{F})$ (described in section 3.3.3), and hence redistribution of the redundant robots, makes sure that we do not use more robots than required to cover the entire environment. While this still is not a guarantee of global optimality, this indeed is a local optimality in the sense that after redistribution we end up with a complex that is the optimum subcomplex of the original complex without any redundant robot.

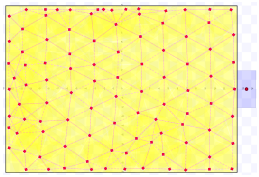
Limitations: Since the robots use the omnidirectional cameras only to obtain bearing to neighbors, and the only way they sense obstacles is through direct touch/contact, it is not possible to detect features that are smaller than r_v. Thus, presence of non-convex features on the surface of the obstacles that are smaller than r_v may result in some "blind-spots" inside the non-convex "grooves".

3.4.2 Simulations

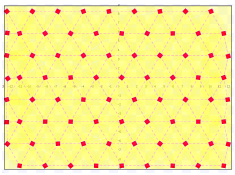
We demonstrate the performance of the proposed algorithm in simulation using integrated platform between Robot Operating System (ROS) [74], Stage Simulator [39] and JavaPlex [90]. We use ROS as a backbone that links all components together. Stage simulates the dynamics and sensors of the robots, while JavaPlex is used to compute the relative homology for identifying the redundant robot.

Comparison with Triangular Tessellation

To evaluate the performance of the proposed algorithms, we compare the number of robots used in covering an obstacle-free rectangular region using our algorithm and using the triangular tessellation, which we constructed manually by overlaying the environment on a triangular tessellation in a free space. In an obstacle-free environment, the performance of our algorithms is comparable to the triangular tessellation solution as illustrated in Figure 3.10. The majority of the robots deployed by our algorithm are in the triangular tessellation arrangement. However, due to accumulated errors and collision with the boundary, the arrangement gets distorted and the clutter of robots is higher near the boundary.

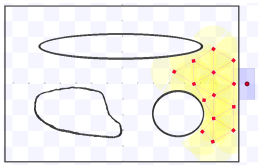


(a) Deployment using the proposed algorithms (robots deployed from the source on the right).

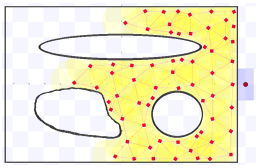


(b) Triangular tessellation using the same average separation between the robots, r_v.

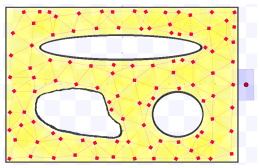
Figure 3.10: Our algorithm deployed 98 robots while the hexagonal packing requires approximately 78 robots.



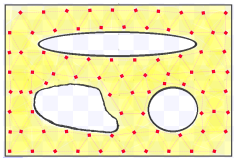
(a) $T = 14$



(b) $T = 64$



(c) $T = 106$



(d) Triangular Tessellation

Figure 3.11: Demonstration in a structured environment with obstacles. Figures (a)-(c) illustrate the progress of our sensor coverage algorithm at $14, 64$, and 106 deployment cycles respectively. Figure (d) is the "ideal" triangular tessellation in the environment for comparison, attained using 82 robots and using the same average separation between the robots.

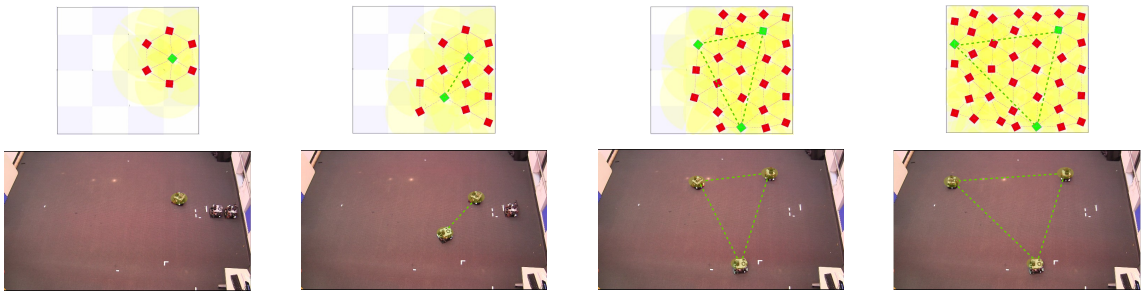
Structured environment

Our algorithm attains a similar performance in a structured environment with few obstacles as illustrated in Figure 3.11. Comparison between figures (c) and (d) illustrates that the performance of our algorithm is comparable to the hexagonal packing.

3.4.3 Experiment with Heterogeneous team of Live and Virtual Robots

We also tested our algorithm on a real experimental platform. Because of limited number of available physical robots, as well as to demonstrate a new paradigm in combining physical robots with virtual/simulated ones in a real-time experiment, we use a heterogeneous team of virtual and real robots to constitute the swarm in the experimental setup.

We used Scarab robot [67] as the physical robot platforms for some of the robots, and used Stage robot simulator to simulate the remaining robots in the swarm. The Scarab is a differential drive robot, while the virtual robots consisted of holonomic robots simulated in Stage. Each real robot in the environment was coupled with a robot in the simulation



(a) $t = 103$s. **(b)** $t = 276$s. **(c)** $t = 600$s. **(d)** $t = 938$s.

Figure 3.12: A heterogeneous team of live (green) and virtual (red) robots covering a simple, obstacle-free environment. The dashed green lines are drawn for comparison between the formation of the live robots and the simulated version of the live robots. The accompanying multimedia attachment shows the video of the simulation environment overlaid on the experiment for better comparison.

environment (*virtual models of real robots*), besides the remaining robots in the swarm being simulated as well (*simulated robots*).

In order to make the real robot work coherently with all virtual robots, its corresponding simulated peer needs to be synchronized through the feedback loop as illustrated in Figure 3.13. Localization of the real robots was done through the use of a motion capture system. The motion capture system would broadcast the pose information of the physical robots in the environment. These poses were used to update the poses of the simulated versions of the real robots using a proportional position control on the pose. This is used in conjunction with the pose of the other simulated robots to emulate bearing sensors for all the robots (real as well as simulated). The real robots would then utilize this sensor data to compute and execute bearing-based visual homing control. In summary, a bearing-based visual homing control was implemented in a heterogeneous team of real and virtual robots.

As a proof of concept, an obstacle free environment was selected for exploration by the heterogeneous team. The results are illustrated in Figure 3.12. The top row shows the simulated environment, with virtual robots colored in red and the simulated version of the live robots colored in green. The snapshots of the experiment in the lower row shows the live robots. The heterogeneous team is able to explore the environment with the limited sensing and communication capabilities. Testing in more complex environments and construction

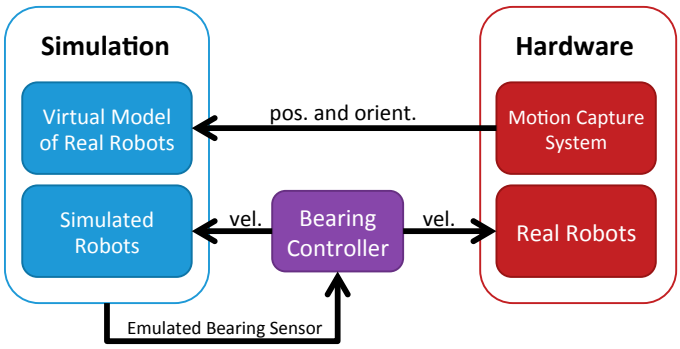


Figure 3.13: Block digram describing the communication and feedback between the simulation platform, experimental (real) robots and the motion capture system.

of the GVG is within the scope of future work.

3.5 Conclusion

This chapter propose an algorithm for the deployment of a swarm of resource-constrained, mobile robots in an unknown environment with the objective of attaining complete sensor coverage of the environment without using any metric information. The only sensors are a limited range omnidirectional camera that can detect bearing to neighboring robots and a touch sensor for detecting contact with obstacles and other robots. No global information is available. The proposed algorithm, which is derived from concepts in algebraic topology, is complete, terminates in finite environments, is robust to noise and robot failures, and is locally optimal. We demonstrated the proposed algorithm in a ROS-Stage simulation, as well as introduced a new paradigm in experimental demonstration involving a heterogeneous team of real and virtual robots.

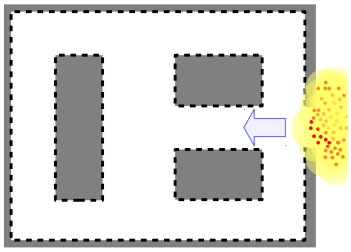
Chapter 4

Topological Mapping

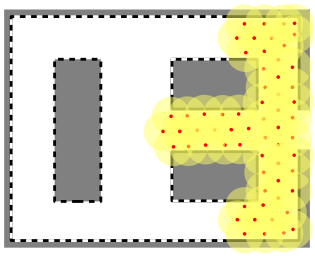
This chapter addresses the problem of building a topological map of an unknown 2-D environment using a swarm of resource-constrained robots. Although the algorithm in Chapter 3 also can provide an estimation of the environment, it requires a large number of robots to fully explore an environment. With limited number of available robots, this chapter proposes an algorithm to create a topological representation of the environment that can be used as a roadmap for navigation of other robots around the environment. The topological map that is of particular interest to us is the Generalized Voronoi Graph (GVG, also called a Generalized Voronoi Diagram or a GVD), which is the locus of points in the configuration space which have more than one distinct "closest" points on the boundary of the configuration space.

Similar to Chapter 3, each robot is only equipped with a limited-range bearing sensor that allows a robot to detect the bearing of its neighbors, and touch sensors that allow a robot to perform obstacle avoidance. As a result, the robot does not have access to range information and cannot localize themselves in any global coordinate system. Our algorithm allows the swarm of robots to construct an approximate GVG of the environment. The graph must be approximate since robots lack the ability to make range measurements. Further since all sensors are local, the graph must be constructed incrementally.

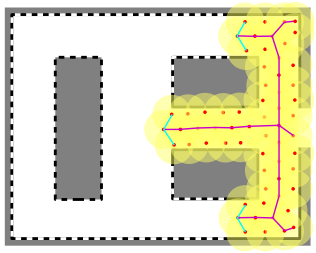
The research contained in this chapter was originally published in [78].



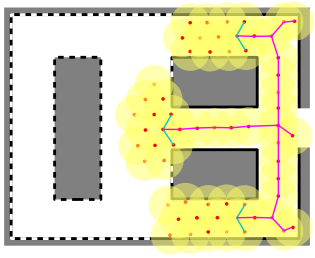
(a) A finite swarm of robots enters an unknown environment.



(b) The swarm "floods" part of the environment and attains hole-less sensor coverage.



(c) A topological map of the partially covered environment (APGVG) is computed – magenta curve).



(d) Robots not constituting APGVG are redeployed to the unexplored region and the process continues.

Figure 4.1: Illustration of the main steps in construction of a physical representation of a topological map using a finite swarm of robots with limited local sensing.

4.1 Overview

The essence of our approach is illustrated in Figure 4.1. The robot swarm enters a completely unknown area (Figure 4.1a). The robots navigate the environment, gather information, construct a simplicial complex representation (a Rips complex) of the environment with sensor coverage, and attain a *hole-less* sensor coverage using all the available robots. Since there are a finite number of robots available, we can only ensure coverage of a subset of the environment (Figure 4.1b). This can be accomplished with the algorithm described in Chapter 3. Once the hole-less sensor coverage of the partial environment is attained, we run a discrete graph-based GVG construction algorithm (similar to [95]) on the 1-skeleton of the Rips complex formed by the robot sensor footprints to identify the robots that can be removed and the ones that need to be kept in order to retain the approximate physical representation of the GVG of the partially covered environment (which we call an *approximate physical/partial GVG* or an APGVG for simplicity – Figure 4.1c). The robots that

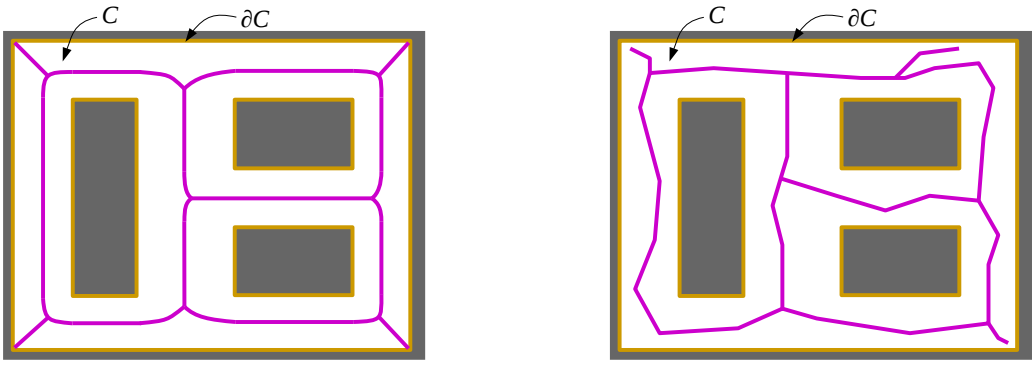
are not tagged to be part of the APGVG can now be redeployed beyond the frontier to the unknown part of the environment to attain hole-less sensor coverage of a new portion of the environment, while the robots stationed on the APGVG of the previously covered environment maintain their position (Figure 4.1d).

The overall algorithm involves interleaving robot deployment cycles with the construction of the APGVG. Every new deployment cycle results in a new APGVG which must be stitched together with the old APGVG so that the stitched result is an approximate GVG of the entire environment. This process results in a sparse *topological map* for the free environment and is a deformation retract of the environment.

We call each of these cycles (involving robot deployment, hole-less sensor coverage of part of the environment, and computation of APGVG for identifying robots for deployment in next cycle) a *APGVG computation cycle* or a APGVGCC for simplicity.

4.2 Multi-stage Construction and Stitching of APGVGs

4.2.1 Generalized Voronoi Graph and its Approximate Discrete Construction



(a) The exact GVG of the free space. **(b)** A topological map that *approximates* the GVG.

Figure 4.2: Comparison of the Genealized Voronoi Graph and its approximation.

The topological map of a configuration space, C, is a 1-dimensional subspace such that it is topologically equivalent to C. In particular, the topology of the topological map captures all the "*holes*" in the space C as *loops* in the topological map (Figure 4.2b). If C is a

subset of $\mathbb{R}^2$ (say, part of the plane with obstacles in it), the topological map can be more precisely described as a 1-dimensional *deformation retract* of the space, and can always be constructed [16, 18, 47]. The GVG is a special topological map with the property that each point on the GVG has two or more equidistant *closest points* on the obstacle boundary of the configuration space, ∂C (Figure 4.2a).

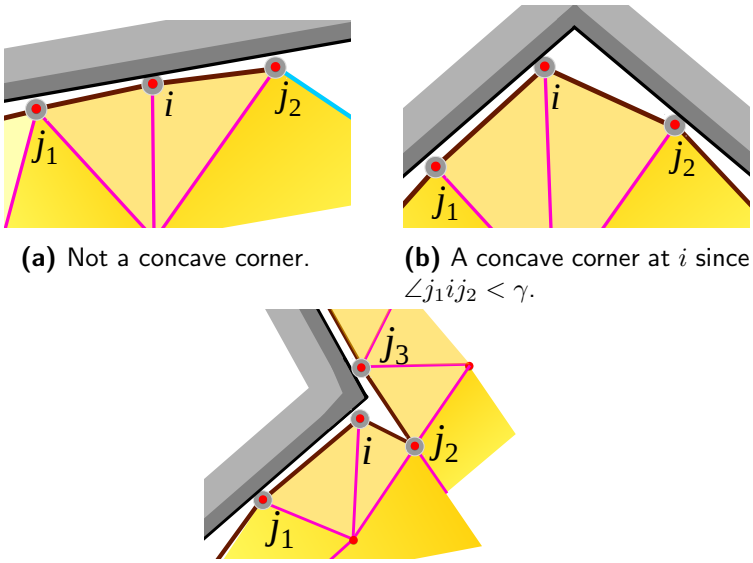
The 1-skeleton of the simplicial complex, $\mathcal{R}_{r_v}(X)$, formed by the robots as described in the previous section, can be considered as an approximate discrete graph representation of the covered subset of the workspace, with the robots being the vertices of the graph and the 1-simplices being the edges. Let's call this graph $\mathcal{G}_{r_v}(X)$ (or $\mathcal{G}_{r_v}$ for simplicity) for the set of robot positions, X. One can employ a *wave-front propagation* algorithm in $\mathcal{G}_{r_v}(X)$, as described in [11, 95], for identifying the vertices (the robots) in the graph which constitute an approximate GVG.

The overall idea is not very different from the continuous gradient flow method for the Computation of GVGs [18] – we employ a breadth-first search (Dijkstra's algorithm) with the initial *open list* containing all the vertices adjacent to the obstacles. Out of those initial vertices adjacent to obstacles, we mark the ones lying between two "*concave corners*" of an obstacle with the same label, so that the part of the wavefront originating from the vertices with the same label sweep a Voronoi cell of the approximate GVG. By the virtue of the Dijkstra's algorithm, the property of the wavefront is that at every instant all points on it are at an equal shortest distance from the closest obstacle. Thus, wherever the wavefronts with different labels *collide*, it ought to be a point on the GVG. The overall process is illustrated in Figure 4.4, and the pseudocode of the algorithm is provided later.

Segmentation of the Obstacle Subcomplex by Concave Corners

As described above, we need to segment the workspace boundary (boundary next to the obstacles) based on the presence of concave corners. However, since we do not have global knowledge of the environment, all that we can use to identify corners at the boundaries is the obstacle subcomplex, $\mathcal{O} \subseteq \mathcal{G}_{r_v}(X) \subseteq \mathcal{R}_{r_v}(X)$. This is achieved through communication between adjacent robots in the obstacle subcomplex. We choose the criteria on the angle at

a corner in the environment to be γ, which, for example in environments with only right-angle corners will be $\frac{\pi}{2}+\epsilon$ (where the factor ϵ accounts for mismatch of the robot placement with the corners, and for all the simulations we choose $\epsilon = \frac{\pi}{4}$).



(a) Not a concave corner.

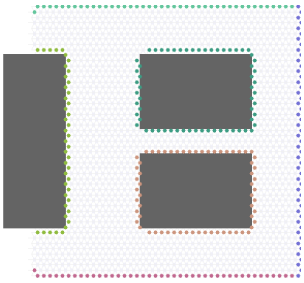
(b) A concave corner at i since $\angle j_1 i j_2 < \gamma$.

(c) Not a concave corner at i since $\angle i j_2 j_3$ does not satisfy normality condition.

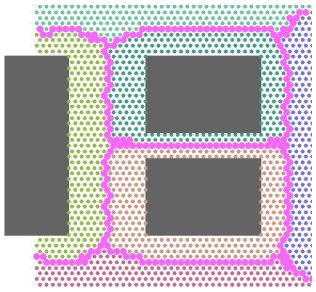
Figure 4.3: Detection of concave corners from the simplicial complex with $\gamma = \frac{\pi}{2} + \epsilon$.

Suppose robots$\{i, j_1\}$ and $\{i, j_2\}$ are 1-simplices in $\mathcal{O}$, and let $\angle j_1 i j_2$ be the angle made by the 1-simplices at i (which, in the local frame of i, is the relative bearing between j_1 and j_2 as seen by i) on the side opposite to the obstacles. We identify i as a concave corner if $\angle j_1 i j_2 < \gamma$ (Figure 4.3b).

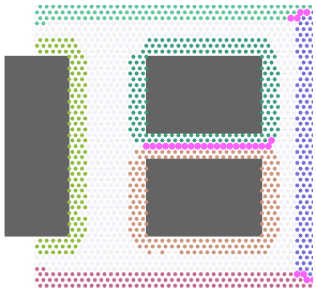
However, under some circumstances (as shown in Figure 4.3c, where we mark ij_2 and $j_2 j_3$ as obstacle 1-simplices due to Algorithm 3, it is not sufficient to consider only the angle made by two consecutive boundary 1-simplices at i. We also need to put some normality condition on angles made by the boundary 1-simplices at j_1 and j_2 in order to avoid too much spurious detection of concave corners. This method for corner detection is, nevertheless, mostly heuristic-based, given that we can only estimate the presence of the corners from relative bearings between the robots, and thus can lead to false positives or false negatives in concave corner detections. However, it is to be noted that the presence of concave corners

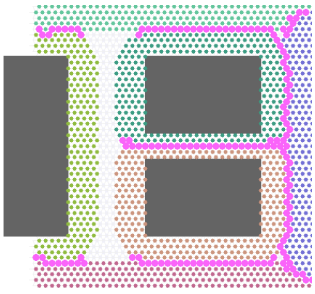


(a) The initial stage in the wavefront propagation, with different labels (marked by different colors) assigned to parts of the wavefront originating from segments of boundaries with no concave corner.



(c) Final approximate APGVG shown in magenta.





(b) The vertices at which the wavefronts with different labels (marked by different colors) "collide" constitute the approximate GVG (magenta disks) for the partially covered environment. Figures show the steps with 930 and 1430 vertices *expanded*.

Figure 4.4: Illustration of the progress of wavefront algorithm for construction of APGVG using a discrete graph representation of the partially covered environment. The vertices of the graph are marked by the small disks and are representative of the physical robots (not to scale, and dense, ideal placement for illustration

only effects the "*branchiness*" of the Voronoi graph, and does not effect its more fundamental property of being a deformation retract.

Once we identify the concave corner robots on the obstacle subcomplex, we assign identical labels to all robots between two concave corners, while the corner robot is assigned either of the labels of the obstacle robots on its two sides. This gives a segmentation of the obstacle subcomplex, $\mathcal{O} = \mathcal{O}_1 \cup \mathcal{O}_2 \cup \cdots \cup \mathcal{O}_\sigma$, where $\mathcal{O}_m$ contains all the robots that are assigned label m (Figure 4.5). Note that $\mathcal{O}_m$ is itself a subgraph of $\mathcal{O}$ (which in turn is a subgraph of $\mathcal{G}_{r_v}(X)$), and we denote the vertex and edge sets of this subgraph by $V(\mathcal{O}_m)$ and $E(\mathcal{O}_m)$, respectively.

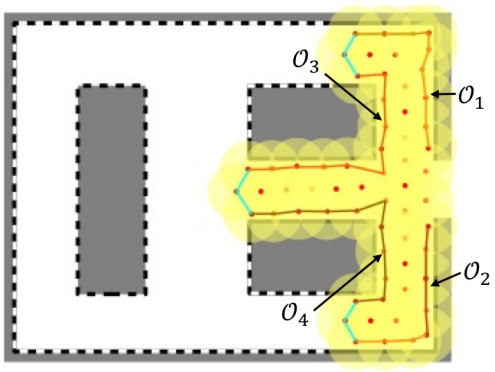


Figure 4.5: Obstacle subcomplex $\mathcal{O}$ is segmented by the corners. The segments $\mathcal{O}_1, \mathcal{O}_2, \ldots$ are the curves in different hues of brown. In cyan we show the frontier subcomplex $\mathcal{K}$

Wavefront Algorithm For Voronoi Graph Construction

Algorithm 5 presents the pseudocode of the wavefront algorithm for computing the GVG in a discrete graph setting as described earlier (also see Figure 4.4). The basic framework of the algorithm is that of Dijkstra's [20]. The algorithm takes as input the 1-skeleton of the Rips complex – the graph $\mathcal{G}_{r_v}$, and the segmented boundary subgraphs, $\mathcal{O}_1, \mathcal{O}_2, \cdots, \mathcal{O}_\sigma$. We assume that the *cost* of each edge in $\mathcal{G}_{r_v}$ is 1 (*i.e.*, "*distance*" is measured in hop count) since we do not have the inter-robot distances available. The algorithm outputs the set of vertices in the graph $\mathcal{G}_{r_v}$ which will constitute the discrete approximate GVG in the graph.

We initiate the *open list* in the search algorithm with all the vertices in the obstacle subcomplex, set their g-value to zero (*i.e.*, they are at a distance of zero from the obstacle subcomplex) and attach a label to them based on the segmentation of $\mathcal{O}$ (lines 5-10). The rest of the algorithm is the usual breadth-first search — at every iteration we choose the vertex, q, in the open list with smallest g-value (line 13), place it in the closed list (*i.e.*, "expand" the vertex – line 17) and update its *un-expanded* neighbors if they will have better g-values if reached via q (lines 22-26).

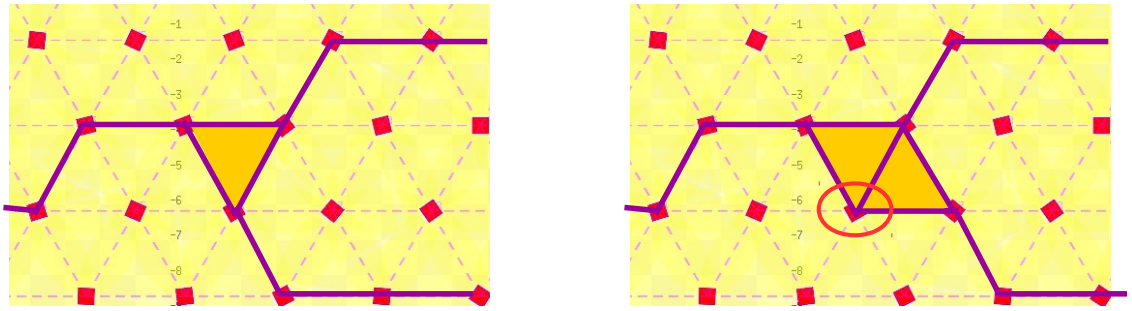
We determine whether the vertex q, which is being expanded, is part of the GVG by looking at its expanded neighbors that have a label different to q's label. Precisely, q is equidistant from two different segments of the obstacle subcomplex if its g-value would have

Algorithm 5 Compute Approximate Physical Generalized Voronoi Graph

Input: Graph $\mathcal{G}_{r_v}$, with vertex set $V(\mathcal{G}_{r_v})$ and edge set $E(\mathcal{G}_{r_v})$;
Segmented obstacle subgraphs, $\mathcal{O}_\alpha \subseteq \mathcal{O}$
Output: Vertex set constituting APGVG, $\mathcal{V} \subseteq V(\mathcal{G}_{r_v})$

1: **function** COMPUTEGVG($\mathcal{G}_{r_v}, \{\mathcal{O}_\alpha\}_{\alpha=1,2,\cdots,\sigma}$)
2: $g(\mathrm{v}) \leftarrow \infty, \forall \mathrm{v} \in V(\mathcal{G}_{r_v})$ ▷ Distances to obstacle
3: $l(\mathrm{v}) \leftarrow -1, \forall \mathrm{v} \in V(\mathcal{G}_{r_v})$ ▷ Labels
4: $\mathcal{V} \leftarrow \emptyset$
5: **for** $k = 1, 2, \cdots, \sigma$ **do**
6: **for each** $\mathrm{v} \in V(\mathcal{O}_k)$ **do**
7: $g(\mathrm{v}) \leftarrow 0$
8: $l(\mathrm{v}) \leftarrow k$ ▷ Assign label k to vertices in $\mathcal{O}_k$
9: **end for**
10: **end for**
11: $Q \leftarrow V(\mathcal{G}_{r_v})$ ▷ Set of un-expanded vertices
12: **while** $Q \neq \emptyset$ **do**
13: $\mathrm{q} \leftarrow \mathrm{argmin}_{\mathrm{q}' \in Q} g(\mathrm{q}')$ ▷ Maintained by a *heap*
14: **if** $g(\mathrm{q}) == \infty$ **then** ▷ Open list is empty
15: **break**
16: **end if**
17: $Q \leftarrow Q - \mathrm{q}$ ▷ Remove q from Q
▷ Look at expanded neighbors with a different label
18: $\mathrm{u} \leftarrow \mathrm{argmin}_{\mathrm{u}' \in \mathcal{N}_{\mathcal{G}_{r_v}}(\mathrm{q})} \{g(\mathrm{u}') \mid \mathrm{u}' \notin Q,\ l(\mathrm{u}') \neq l(q)\}$
19: **if** $g(u) + 1 == g(q)$ OR $g(u) == g(q)$ **then**
20: Insert q into $\mathcal{V}$ ▷ It's a GVG vertex!
21: **end if**
▷ un-expanded neighbors that need updating:
22: $W \leftarrow \{\mathrm{w} \in \mathcal{N}_{\mathcal{G}_{r_v}}(\mathrm{q}) \mid \mathrm{w} \in Q,\ g(\mathrm{w}) > g(\mathrm{q}) + 1\}$
23: **for** $\mathrm{w} \in W$ **do**
24: $g(\mathrm{w}) \leftarrow g(\mathrm{q}) + 1$ ▷ Update to lower g-value
25: $l(\mathrm{w}) \leftarrow l(\mathrm{q})$ ▷ Copy label to neighbor
26: **end for**
27: **end while**
28: **return** $\mathcal{V}$
29: **end function**

been the same (or one more) had it been expanded via a differently labeled vertex, and hence placed in $\mathcal{V}$ (lines 18-21). However, this process may end up including some redundant vertices (and corresponding 2-simplices from $\mathcal{R}_v$) in the GVG, which are not essential in maintaining the connectivity of the GVG (Figure 4.6b). We remove such vertices from the GVG through a simple post-check of the number of 2 and 1-simplices connected to a vertex belonging to the GVG. In particular, if a vertex in the GVG is connected to n edges (1-simplices) that are part of the GVG, which in turn form a boundary of at least $n-1$ counts of 2-simplices, and also is part of the GVG, then the vertex is redundant in the GVG and



(a) None of the robots in the marked GVG can be removed.

(b) Robot marked by red ellipse in the GVG is redundant.

Figure 4.6: Identifying robots that can be removed from the GVG computed by ComputeGVG.

can be removed (an explicit deformation retract can be constructed).

4.2.2 Robot Redeployment and Stitching the APGVGs

As illustrated in Section 4.1, we construct the partial GVGs in the discrete setting (the APGVGs) at every APGVGCC.

For easy reference, for the i^{th} APGVGCC we will use superscripts of i to indicate the different objects described so far (*e.g.*, X^i will be the set of robot positions constituting the hole-less coverage of the partial environment at the i^{th} APGVGCC, $\mathcal{R}^i_{r_v}$ its Rips complex, $\mathcal{G}^i_{r_v}$ its 1-skeleton, $\mathcal{K}^i$ the frontier subcomplex, APGVG^i the GVG computed on $\mathcal{G}^i_{r_v}$ using segmented obstacle subcomplex $\mathcal{O}^i = \mathcal{O}^i_1 \cup \mathcal{O}^i_2 \cup \cdots \cup \mathcal{O}^i_{\sigma^i}$, etc.). Thus, we have APGVG^i = ComputeGVG$(\mathcal{G}^i_{r_v}, \{\mathrm{O}^i_\alpha\}_{\alpha=1,2,\cdots,\sigma^i})$).

By the virtue of its construction, APGVG^i and APGVG^{i+1} will be connected to $\mathcal{K}^i$ (Figure 4.7(a)). Following the computation of APGVG^{i+1}, we consider each connected component of $\mathcal{K}^i$, and identify the subcomplex, $\mathcal{S}^i$, necessary to keep the branches of APGVG^i and APGVG^{i+1} emanating from that component of $\mathcal{K}^i$ connected to each other (this essentially boils down to eliminating the robots at the trailing ends of the connected component of $\mathcal{K}^i$ – Figure 4.7(b)). We exclude the robots in $\mathcal{S}^i$ from the set of robots, Λ^{i+1}, for re-deployment in the $(i+1)^{th}$ APGVGCC (Figure 4.7(b)).

Thus we identify the set of robots that can be redeployed for the APGVGCC^{i+1} as

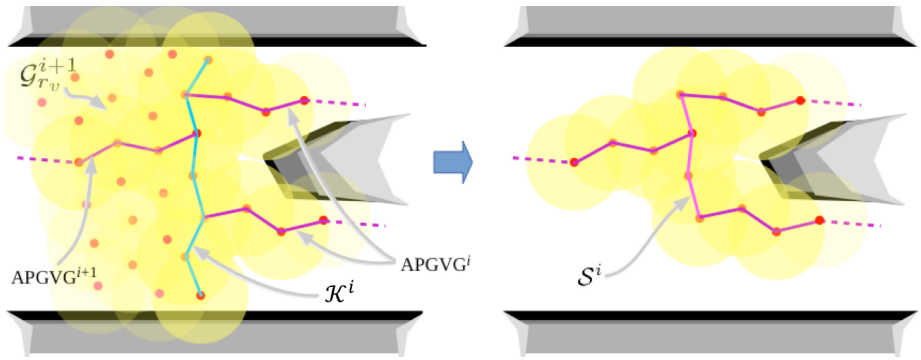


Figure 4.7: Stitching APGVG^i and APGVG^{i+1} by considering each connected component of $\mathcal{K}^i$. The set of robots re-deployed/removed in going from (a) to (b) is $\Lambda^{i+1} = \mathcal{G}_{r_v}^{i+1} - (\text{APGVG}^{i+1} \cup \mathcal{K}^{i+1} \cup \mathcal{S}^i)$

$\Lambda^{i+1} = \mathcal{G}_{r_v}^{i+1} - (\text{APGVG}^{i+1} \cup \mathcal{K}^{i+1} \cup \mathcal{S}^i)$ (*i.e.*, we leave the robots on the partial GVG just computed, as well as those on the current frontier subcomplex and the subcomplex of the past frontier, $\mathcal{S}^i$).

In general, we use the "push" strategy through the 1-skeleton as described in 3.3.2 for finding paths to transport the robots one at a time to explore new locations outside the frontier $\mathcal{K}^{i+1}$. We then use the local bearing-based control described in Section 2.2.1 to move the robots. However, in some cases the path from a re-deployable robot to a frontier may contain parts of the approximate GVG which are not surrounded by neighboring robots any more. The "push" strategy does not work well under such circumstances due to lack of a sufficient number of landmark robots for the bearing-based controller. For that we need to use a separate controller, which we are in the process of implementing, that physically navigates the re-deployable robot along the single-robot chain constituting the GVG. In the simulations presented in this paper such a situation does not arise since we use sufficient number of robots in the swarm.

The overall algorithm for the multi-stage approximate GVG construction can thus be summarized as follows:

Algorithm 6 Multi-stage approximate GVG construction using a finite robot swarm

$\Lambda^0 :=$ the set of all robots
$\mathcal{K}^0 :=$ the initial frontier at the entrance
$i := 0$
while $\mathcal{K}^i \neq \emptyset$ **and** $\Lambda^i \neq \emptyset$ **do**
▷ APGVGCC^{i+1}
    Deploy robots in Λ^i to unexplored region outside $\mathcal{K}^i$ for hole-less coverage and construct the final $\mathcal{R}_{r_v}^{i+1}$ (which includes $\mathcal{K}^i$), with its 1-skeleton $\mathcal{G}_{r_v}^{i+1}$.
    Compute the obstacle and frontier subcomplexes, $\mathcal{O}^{i+1}, \mathcal{K}^{i+1} \subseteq \mathcal{R}_{r_v}^{i+1}$
    APGVG^{i+1} = ComputeGVG($\mathcal{G}_{r_v}^{i+1}, \{\mathrm{O}_\alpha^{i+1}\}_{\alpha=1,\cdots,\sigma^{i+1}}$)
    Identify robots $\mathcal{S}^i \subseteq \mathcal{K}^i$ to keep for proper of APGVG^i and APGVG^{i+1} stitching
    $\Lambda^{i+1} = \mathcal{G}_{r_v}^i - ($APGVG$^{i+1} \cup \mathcal{K}^{i+1} \cup \mathcal{S}^i$
    $i := i + 1$
end while

4.2.3 Estimation of the Number of Robots Required

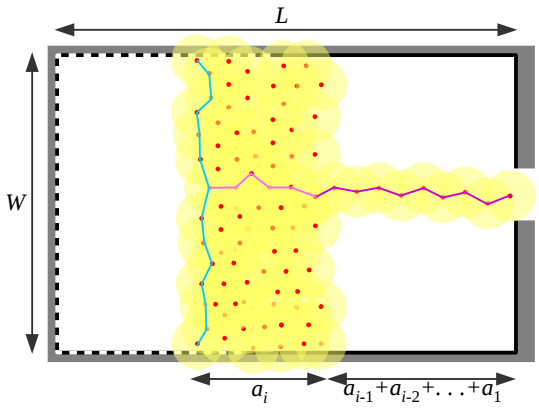


Figure 4.8: Illustration of the number of robots required for an obstacle-free environment with $\beta = 1$.

The number of robots required for being able to completely construct the approximate GVG is generally highly dependent on the precise structure of the environment. However, under some assumptions of the environment, a very rough and informal estimate can be worked out. In this section, we provide an extremely simplified estimate of the number of robots that will be required for constructing the complete approximate GVG using the algorithm described in this paper. We consider the dimensions of the environment described in terms of number of average robot separations along X or Y directions (which, we approximately assume to be uniform, and equal to κr_v for some constant κ). Suppose the width

of an environment (the dimension orthogonal to the main flow direction of the robots) is W times the average robot separation, and the length (dimension in the direction of robot flow) is L times the average robot separation (Figure 4.8 shows the obstacle-free case). Furthermore, in the presence of obstacles, let the average number of "branches" that the final GVG will have in the vertical direction be β.

Say we start with $n = n_1$ robots. In APGVGCC^1, those robots will be able to progress a distance of $a_1 = \frac{n_1}{W}$ average robot separations along the width of the environment. This will also be equal to the approximate number of robots that will constitute APGVG^1 with βa_1 robots. Thus, the remaining robots, $n_2 \simeq n_1 - \beta a_1 = n_1 \frac{W-\beta}{W}$ can be deployed for APGVGCC^2. In general, using an inductive argument, at the beginning of the i^{th} APGVGCC, the number of robots available will be $n_k \simeq n_{k-1}\rho \simeq n_1 \rho^{k-1}$, where $\rho = \frac{W-\beta}{W}$. However, if the algorithm terminates at the m^{th} APGVGCC, we should at least have the final free robots span the entire width of the environment (so that there are enough robots to have the complete obstacle subcomplex and empty frontier subcomplex, for being able to compute the final APGVG effectively). This gives us

$$n_1 \rho^{m-1} \simeq W \tag{4.1}$$

Furthermore, we should be able to span the entire length of the environment using the APGVGs of length $a_1, a_2, \cdots, a_{m-1}$. Thus,

$$\frac{1}{W}\left(n_1 + n_2 + \cdots + n_{m-1}\right) \simeq L \tag{4.2}$$

$$\Rightarrow \qquad \frac{1-\rho^{m-1}}{1-\rho} \simeq \frac{LW}{n_1} \tag{4.3}$$

Combining the above equations, and eliminating m, one gets $n_1 \simeq W + \beta L$. Thus, this simplified estimate puts the required number of robots at $W + \beta L$.

In practice we would surely like to keep a margin for safety and have more robots than what is presented in this simple estimate. For instance, the estimation for the environment in Figure 4.9 is about 60 robots, while we used about 100 robots in the simulation.

4.3 Results

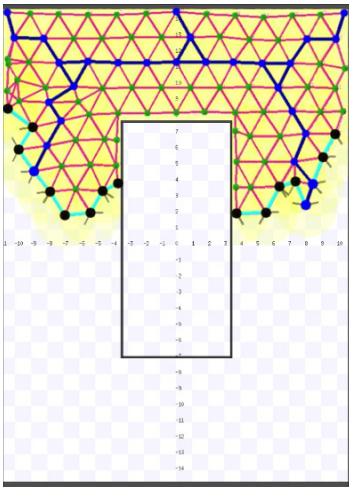
We implemented the proposed algorithm on the Robot Operating System (ROS) [74] platform with the kinematic robots and the on-board sensors simulated by Stage robot simulator [39]. All the APGVG related computations are performed and kept by the server node, while each robot sends the locally-sensed data to the server and listens to commands from the server. Obstacle avoidance is performed individually by each robot.

Figure 4.9 shows a simple environment with an entrance at the top. A team of 100 robots construct a topological map (an approximate GVG) using the proposed algorithm in four APGVGCCs, with a total of 259 deployment iterations, which is the approximate number of robots required to cover the entire environment.

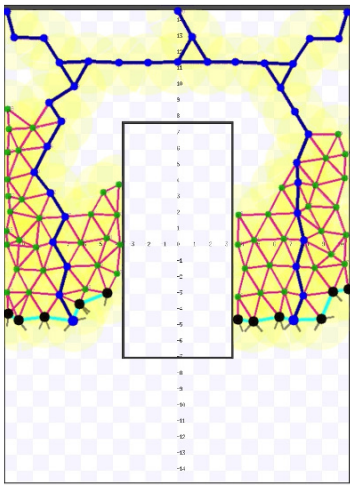
Figure 4.10 shows a simulation in a more complex indoor environment (a part of the 4^{th} floor plan of the Levine building at the University of Pennsylvania). We construct the approximate GVG of the environment with a swarm of 270 robots that is not sufficient to *fill* the entire environment (Figure 4.10a). The experiment was completed in three APGVGCCs with 535 deployment iterations.

4.4 Conclusion

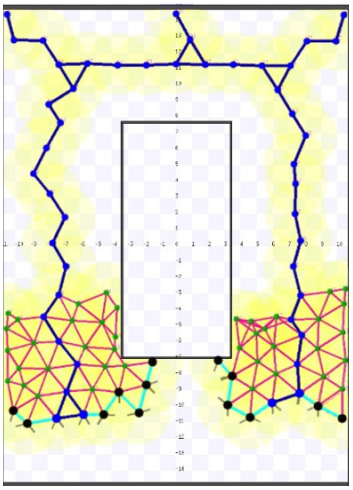
This chapter presents the basic theory and algorithms that allow a swarm of resource-constrained robots to automatically create a topological map, specifically a Generalized Voronoi Graph, of indoor environments without requiring metric information. This method involves covering part of the free space of an environment prior to constructing a Generalized Voronoi Graph from the covered space. The excess robots are then used to extend the covered space and further construct a GVG of the environment until a full topological representation is completed. We demonstrate the proposed algorithm in a ROS-Stage simulation on a simple and complex office-like environment. The constructed topological can then be used for fast and efficient transportation of other robots and resources from one region to another in unknown, GPS-denied environments.



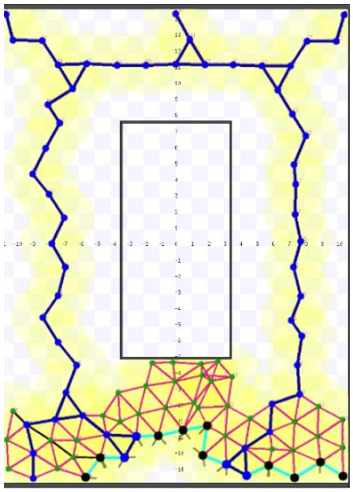
(a) The Rips complex, $\mathcal{G}^1_{r_v}$, with frontier subcomplex, $\mathcal{F}^1$, marked in cyan and black, and the discrete GVG, APGVG^1, computed in $\mathcal{G}^1_{r_v}$ shown in blue.



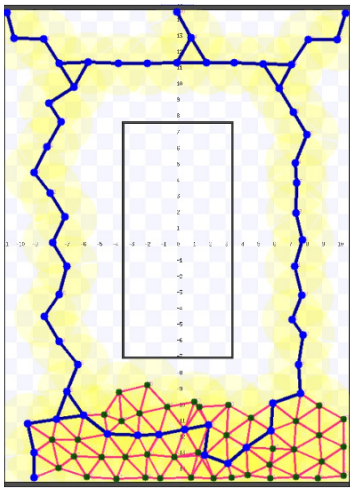
(b) Robots are re-deployed to construct $\mathcal{G}^2_{r_v}$. The new discrete Voronoi graph, APGVG^2, is also shown in blue, stitched with the earlier APGVG^1.



(c) The end of APGVGCC^3, showing the three subsequent APGVGs stitched together.

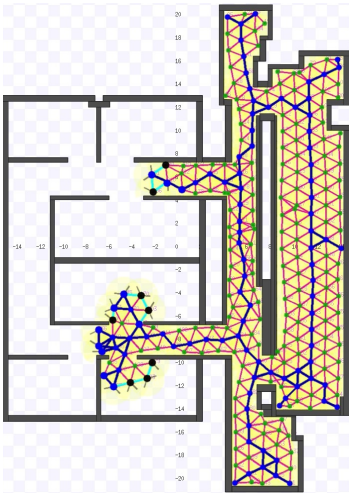


(d) The conclusion of APGVGCC^4.

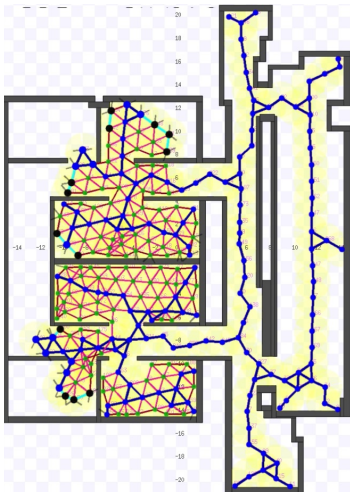


(e) The algorithm terminates since there are no more unexplored frontiers ($\mathcal{F}^5 = \emptyset$).

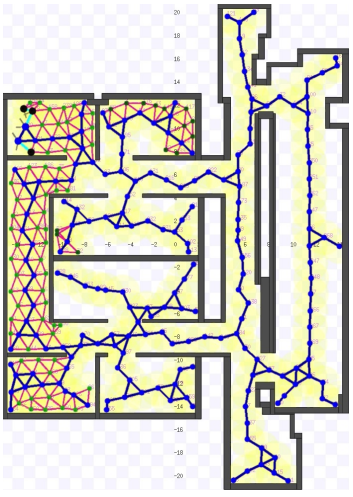
Figure 4.9: Demonstration of the proposed algorithm in ROS simulation using a simple environment.



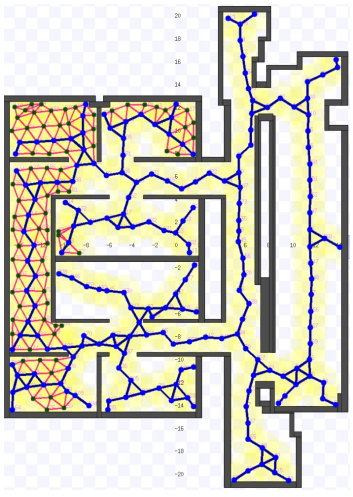
(a) APGVG[1], computed in $\mathcal{G}^1_{r_v}$.



(b) At the end of the computation of APGVG[2].



(c) The end of APGVGCC[3].



(d) Algorithm terminates since there are no more unexplored frontiers.

Figure 4.10: Simulation in the complex (office structure) environment.

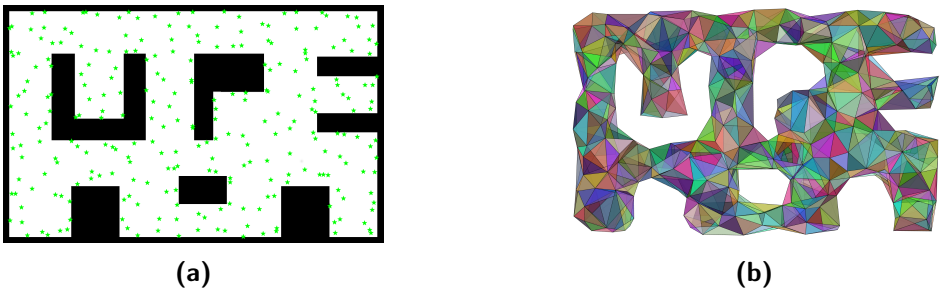
Chapter 5

Landmark-based Exploration

Chapter 3 and 4 focus on robot swarms with inter-robot sensing only. Although these robots are capable of exploring and gaining the global information of the unknown environment with few assumptions regarding its geometry, the number of robots required is proportional to the dimension of the environment. This drawback leads us to consider a little more sophisticate sensing model where each robot is equipped with an omni-directional, limited-range sensor that can identify landmarks in its neighborhood. These landmarks can be obtained from computer vision tools such as feature detections or object recognitions. Nevertheless, there is no global information regarding the robot's position or any range measurements available and thus the robot needs to use the bearing angles to the landmarks for local navigation.

In this chapter, we assume that the detection of landmarks is trivial, *i.e.*, the robot always detects any landmarks within its disk of visibility and identify them with unique IDs. Given a collection of identifiable landmarks, a landmark complex, a simplicial complex constructed from observations of landmarks as discussed in Section 2.1.3, can then be cumulatively constructed to encapsulate the topological information of the environment. Under the assumption that there are sufficiently dense landmarks, we propose exploration and exploitation strategies that guide a swarm of robots to explore an environment using only bearing measurements without any global information or explicit range measurement. These strategies are built on top of the coordinated exploration technique discussed in Sec-

tion 2.3.3.



(a) (b)

Figure 5.1: Illustration of the geometric realization of the *landmark complex* (b) that represents the environment (a). The map is constructed from a collections of observations of the distinguishable landmarks (green stars) by the robots during exploration.

Given the environment filled with distinguishable landmarks, we first present the strategy to actively guide the robot swarms to explore the environment and construct a landmark complex, as illustrated in Figure 5.1. The basic strategy directly constructs a navigation roadmap from the dual of the landmark complex and utilizes an existing bearing-based controller to guide the robots around the environment. We then establish the necessary and sufficient conditions that guarantee the completeness of this metric-free exploration strategy. In simulation, we extensively evaluate the performance of the proposed method in both idealized and more realistic scenarios.

In the last section, we propose the solutions to address some of the limitations of our landmark-based exploration strategy. First, we propose an alternative control method that utilizes the relative distances toward the landmarks, which can obtained from consecutive bearing measurements, for exploration and exploitation. The advantages of this new control policy are threefold: lowering the requirement on the density of landmarks, not relying on the global compass direction, and yielding a collision-free path during navigation. Second, for specific scenarios, we redefine the notion of landmarks so that the landmark complex can detect some of the small features in the environment. Lastly, we discuss the method to handle the misidentification of landmarks.

Part of the research contained in this chapter was originally published in [75].

5.1 Preliminaries

5.1.1 Notations

We denote by $\mathcal{W} \subset \mathbb{R}^2$ the obstacle free region where the swarm of robots can be deployed to explore the environment. Our objective is to deploy a swarm of N robots, denoted by $\{R_i\}_{i=1}^{N}$, to explore $\mathcal{W}$ and construct a sparse map that encapsulates all topological features of the environment. The position of R_i is denoted as $x_i \in \mathcal{W}$, and we represent the set of the positions of all the robots as $X = \{x_1, x_2, ..., x_n\}$.

Given a collection of m identifiable, stationary landmarks, $\{L_i\}_{i=0}^{m}$, we represent the position of L_i as $y_i \in \mathcal{W}$, and represent the set of the positions of all the landmarks as $Y = \{y_1, y_2, ..., y_m\}$. A robot, R_i, equipped with an omni-directional camera, can measure the bearing toward the landmarks within its disk-shaped sensing footprint of radius r. Thus, $S_i = \{L_j \mid \|x_i - y_j\| \leq r, \text{ and } \forall \alpha \in [0,1], (1-\alpha)x_i + \alpha y_j \in \mathcal{W}\}$ denotes the set of landmarks detected by R_i. The bearing measurement (relative to a fixed reference frame) to landmark $L_j \in S_i$ is denoted by $\theta_j^i \in [-\pi, \pi)$. The relative bearing between landmarks $L_j, L_k \in S_i$ is defined as $\theta_{jk}^i = (\theta_k^i - \theta_j^i) \mod 2\pi$, which is between 0 and 2π. Given an observation S_i by robot R_i, let $\boldsymbol{\theta}_i = \{\theta_j^i | L_j \in S_i\}$ denotes the snapshot of bearing measurement by R_i. Figure 5.2 illustrates the sensing model of two robots with four landmarks. All landmarks are within the distance of r from R_1. However, $S_1 = \{L_1, L_2, L_4\}$ because L_3 is occluded from the line-of-sight of R_1.

We assume a very simple obstacle detection model: A robot can detect and identify the general direction of an obstacle only upon contact. In practice, computer vision techniques such as stereo vision system can be used for collision avoidance in local navigation [34]. Additionally, we assume that the robots are holonomic, hence the configuration of each robot is simply the position, $x \in \mathcal{W} \subseteq \mathbb{R}^2$.

5.1.2 Dispersion

We borrow the notion of *dispersion* from sampling theory [59] to numerically quantify the sparsity of landmark distribution. The dispersion of a finite set P of samples in a metric

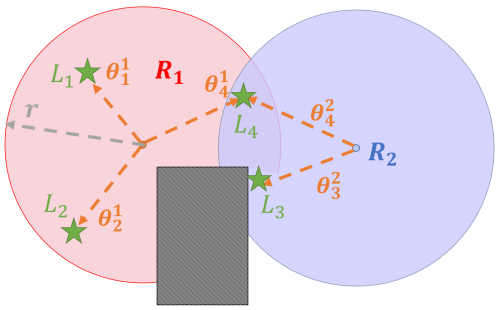


Figure 5.2: Observation: $S_1 = \{L_1, L_2, L_4\}$, and $S_2 = \{L_3, L_4\}$. The measurement for robot R_1 are $\theta_1^1, \theta_2^1, \theta_4^1$, , which be used to calculate relative bearing $\theta_{12}^1, \theta_{24}^1, \theta_{41}^1$. Although L_3 is within radius r from R_1, it is occluded by obstacle and hence not a member of S_1.

space $(\mathcal{X}, \rho)$ is

$$\delta(P) = \sup_{x \in \mathcal{X}} \{\min_{p \in P} \{\rho(x, p)\}\}$$

Using L_2 metric, the L_2 dispersion in $\mathbb{R}^2$ can viewed as the radius of the largest ball that does not touch any sample points p.

The dispersion of the landmarks $\delta(\{L_i\})$ or simply δ over L_2 metric is defined as

$$\delta = \sup_{x \in W} \{\min_{y_j \in Y} \{\|x - y_j\|\}\}$$

Hence, the upper bound of the dispersion is r so that every point in the environment can observe at least one landmark.

5.1.3 Multi-robot Exploration

To demonstrate the application of landmark complex in exploration task, we adapt the frontier-based coordinated strategy from [13], which has an assumption of centralized coordination. The frontier assignment and the mapping occur in a centralized manner, while each robot executes their task individually. In summary, the exploration process can be divided into three general steps as the following.

1. Identify all unexplored frontiers.

2. Calculate the cost-utility function for each frontier for all robots.

3. Update the map as robots navigate to the frontiers

Generally, the cost-utility function is the estimated difference between profits (utility) and expenses (cost), which can be defined as the following.

Cost: The cost function is the distance between current location and the frontier.

Utility: The utility function is the expected information gain, which can be defined as the expected area that a robot will explore.

5.2 Algorithmic Design

This section presents the basic strategy where we assume that the global compass direction is available, *i.e.*, the robots can determine their own orientations either by using magnetometer or by sharing inter-robot bearing measurement when crossing other robots. The robots are locally controlled using the bearing-only navigation system discussed in Section 2.2.2.

The outline of an algorithm is presented in Algorithm 7. The exploration process occurs in an iterative manner until all frontiers are explored. In line 2, the process begins with identifying the set of frontiers – a collection of unexplored areas as defined in Section 5.2.1. As the robots *explore* beyond the frontiers, they construct the landmark complex $\mathcal{L}$, and thus *exploit* it for navigation by constructing its dual navigation graph $\mathcal{G}$ in line 3. Since landmark complex is a topological representation of the entire environment, it is constructed cumulatively (see Algorithm 8), while the navigation graph $\mathcal{G}$ is computed for each instance of $\mathcal{L}$ as described in Section 5.2.2.

Using the navigation graph $\mathcal{G}$ and the list of frontiers $\mathcal{F}$, we compute the cost-utility for all pairs of free robots R_i and frontiers $\langle a, b\rangle \in \mathcal{F}$ in line 5. We then choose the pair of the frontier $\langle a, b\rangle$ and free robot R_i such that the cost-utility function is maximized in line 6. The assignment table, A, is used to keep track of the current assignment, where $A[i] = \{a, b\}$ denotes frontier $\langle a, b\rangle$ is assigned to R_i.

The robot-to-frontier assignment can be done using a greedy algorithm for simplicity or the Hungarian algorithm for the optimal assignment. However, since the task assigned to one robot affects the utility function of other robots, it is not trivial to apply the Hungar-

ian algorithm. Furthermore, although all robots begin as free, each robot will execute its assigned task in parallel and may not finish at the same time. The task assignment will often occur in asynchronous manner and thus the benefit of Hungarian algorithm would be diminished.

After receiving an assignment, each robot individually navigates to its assigned frontier using the bearing-based controller and the navigation graph $\mathcal{G}$. As the swarms navigate and explore the environment, their observations are used for cumulatively updating the landmark complex $\mathcal{L}$ and frontier list $\mathcal{F}$.

Algorithm 7 Swarm Exploration Overview

```
do
    Identify the set of frontiers F
    Construct/update the landmark complex L and compute the navigation graph G.
    while exist robot with no frontier assignment do
        Compute the cost-utility for each free robot R_i with each frontier ⟨a, b⟩ ∈ F.
        Assign frontier to free robot with maximal cost-utility function .
    end while
while F ≠ ∅
```

5.2.1 Frontier Identification

In frontier-based exploration, the boundaries of the unexplored regions constitute frontiers. In our case, a frontier constitutes of an ordered pair of landmarks with the order describing the orientation of the frontier region.

With m landmarks, there are $m(m-1)$ combinations of possible frontiers. At the beginning, all frontiers' statuses are unknown. As robots explore the environment, the frontiers are identified using relative bearing measurements. Given an observation S_i, every pair of landmarks and orientations become potential frontiers. The potential frontier is then either identified as *clear* or *unexplored*. Once the frontier is marked as clear, it will remain unchanged throughout the exploration. The objective of the exploration process is then to clear all unexplored frontiers.

A frontier can be cleared in two ways. Firstly, the frontier is observed as part of the

inner diagonal of the explored polygon, where an explored polygon is the convex hull of the observed landmarks. Without the range measurement to landmarks, we use the relative bearing to determine whether the observation occur inside or outside the explored polygon. The largest relative bearing between any two adjacent landmarks can be larger than π if and only if the observation occurs outside the explored polygon. This is checked by sorting the bearing in the polar coordinate system and calculating the different between the adjacent landmarks on the list. Figure 5.3 illustrates the examples of observation that occurs inside and outside the explored polygon during exploration process. With limited information, either half of the potential frontiers are cleared or all of them are marked as unexplored for each observation.

Secondly, the frontier is cleared if it is an obstacle boundary, i.e., a boundary that is adjacent to the obstacles. For actual obstacle boundary, the robot will most likely run into obstacles which can then be detected. On the other hand, it is difficult to determine that the frontier is not the boundary without any filtering process as illustrated by Figure 5.4. There are two potential solutions to resolve this issue.

The first solution involves tracing the $1-$skeleton of the landmark complex after the exploration is completed. During the task execution, the frontier that is undetermined after randomly investigating for a certain amount of time is then marked as a potential obstacle boundary. After the exploration process is completed, we calculate the $1-$skeleton of the landmark complex to trace of all obstacle boundaries and re-explored potential boundaries that remain ambiguous, i.e. there is an inconsistency between the obstacle boundary from the landmark complex and the actual boundary marked by the exploration process. Nevertheless, this approach does not guarantee any completeness as the frontier exploration relies on the random walk process.

On the other hand, we can decrease the unpredictability of the random walk during frontier exploration by decreasing the dispersion of the landmarks, i.e., making them denser. As dispersion decreases, the landmark becomes closer causing the overlapped regions to be larger. Hence, the robot is more likely to observe simplices in the landmark complex.

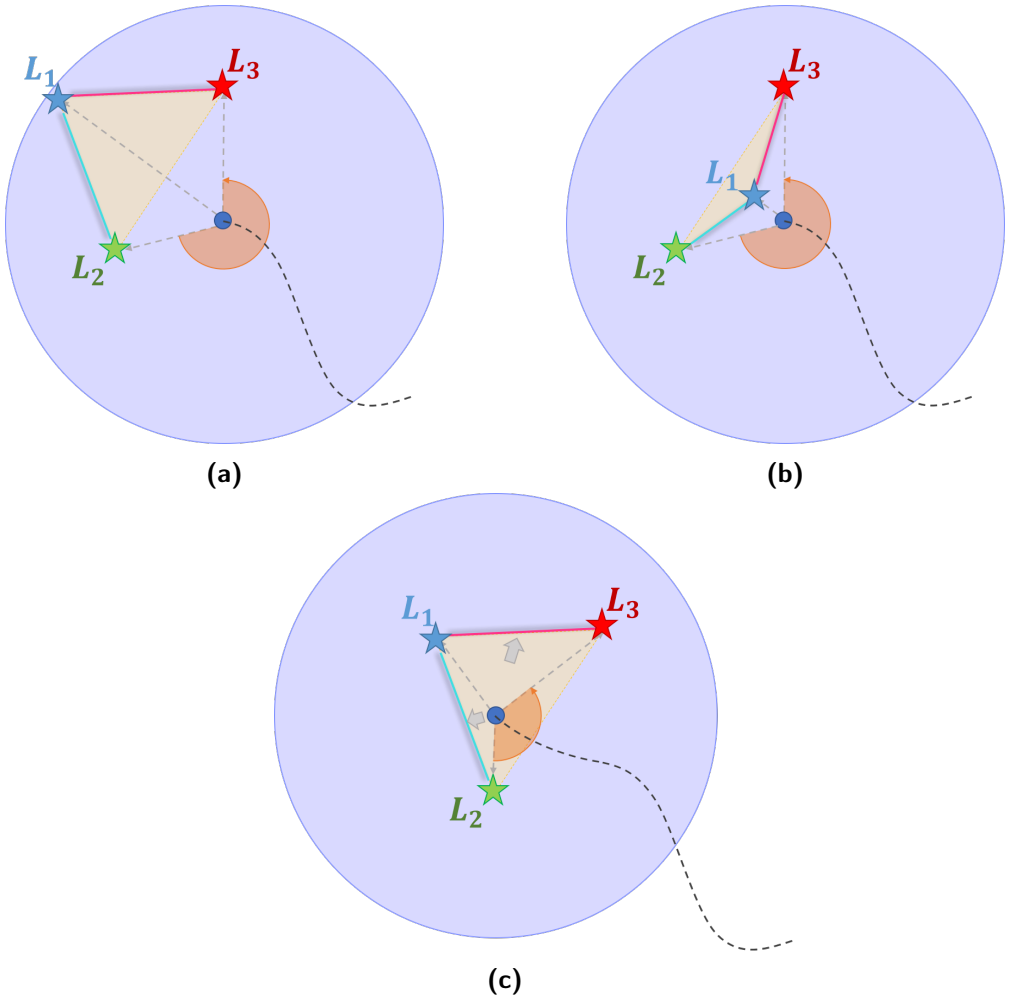


Figure 5.3: Frontier Identification: A method to determine the frontiers during exploration process. The robot is assigned to explore frontier $\langle 2, 3\rangle$ with the current observation includes three landmarks $\{L_1, L_2, L_3\}$. The explored polygon is shaded in yellow and the largest relative bearing is shaded in orange. Before getting inside the explored polygon (a-b), the largest relative bearing is larger than π and none of the frontiers is cleared since the robot cannot determine the placement of landmarks with bearing measurement alone. For instance, the robot observes the exact same bearing measurement for both (a) and (b) and hence none of the frontiers can be cleared. However, after moving inside the explored polygon (c), the bearing becomes smaller than π and three of the potential frontiers, including the assigned frontier, $\langle 1, 2\rangle, \langle 2, 3\rangle, \langle 3, 1\rangle$ are cleared.

Ideally, the entire configuration along the edge between two landmarks should be contained inside the overlapped region so that crossing the boundary would suffice to clear the frontier as illustrated in Fig 5.5a. In other words, the landmark complex constructed from observations of every boundary would suffice to capture all topological features of the envi-

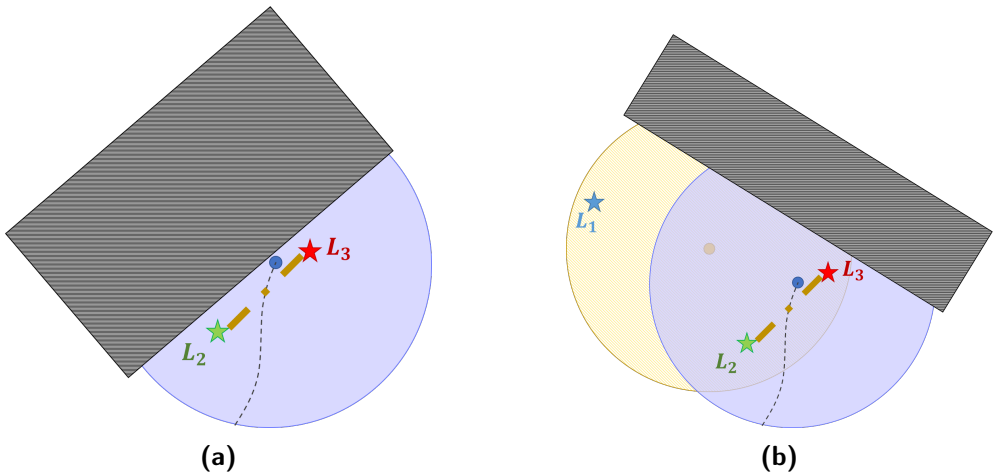


Figure 5.4: Obstacle Boundary: A frontier constituting of the boundary of the obstacle. In the first case (a), the robot hit the obstacle right after crossing the edge and cannot continue the exploring $\langle 2, 3\rangle$, so $\langle 2, 3\rangle$ can be masked as obstacle boundary. On the other hand (b), the robot can continue exploration for a while after crossing the boundary into $\langle 2, 3\rangle$. Thus, the status of $\langle 2, 3\rangle$ is uncertain. Although $\langle 2, 3\rangle$ is not an obstacle boundary, the desired observation occurs at the position denoted by the yellow striped disk, where the robot may or may not reach.

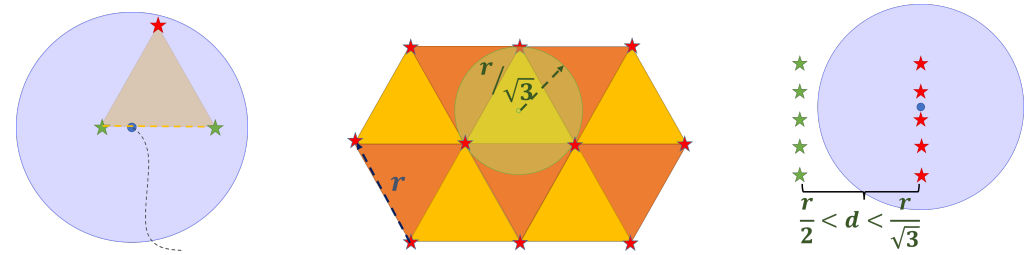
ronment. We consider the optimal and worst case to determine the upper and lower bound of δ.

Proposition 1. The density of landmarks, $\delta, \in [\frac{r}{2} \frac{r}{\sqrt{3}}]$ is necessary and sufficient.

The optimal landmark distribution occurs when all landmarks have the distance of r from each other. This forms a uniform triangular tessellation where each equilateral triangle has the side length of r (Fig 5.5b). Hence, the radius of the circumscribed of equilateral triangle provide the upper bound of the dispersion, $\delta \leq \frac{r}{\sqrt{3}}$. Nevertheless, it is not sufficient as one of the counter example illustrated in Figure 5.5c.

For sufficient condition, we consider the worst case scenario when the landmarks form the lines with the maximal separation as illustrate in the counter example in Figure 5.5c. Hence, without minimal distance between landmarks, the sufficient condition needs to constrain the separation between lines to r. As a result, the lower bound of the dispersion is $\frac{r}{2} \leq \delta$.

In practice, the necessary condition would be good enough since intersected regions is large enough that the robot will hardly miss it. Additionally, with the dispersion of $\frac{r}{\sqrt{3}}$, it will satisfy the necessary landmarks required for navigation in all scenarios as the landmarks



(a) Ideal Distribution **(b)** Triangular Tessellation **(c)** Poor Distribution

Figure 5.5: In ideal scenario (a), a robot can see other adjacent landmarks from any point along the boundary frontier. A robot can detect red landmark from any point along between two green landmarks. The optimal landmark distribution forms a triangular tessellation with side length of r and the circumscribed of equilateral triangle is $\frac{r}{\sqrt{3}}$ (b). However, in poor distribution (c), the robot may not detect other landmarks from the boundary even if $\delta \leq \frac{r}{\sqrt{3}}$.

become approximately three times denser. Note that additional landmarks may be required near the obstacle to satisfy the necessary condition for navigation.

5.2.2 Landmark Complex and Navigation Graph Construction

The landmark complex $\mathcal{L}$ is cumulatively constructed as the robots explore an environment. The update function is defined in Algorithm 8. For any given observation, the landmark complex is updated by adding simplices constructed from all possible combination of detected landmarks as defined in line 5. $\mathbb{P}(S_i)$ denotes the power set of S_i. Additionally, a bearing lookup table, BLT, is bookkeeping a snapshot of observed bearing for guidance during navigation. The landmark complex only need to be updated when changes occur in the set of detected landmarks and each robot may choose any instance of bearing measurement as the reference stored in BLT. Due to noisy measurement, one may use the average of multiple measurements to reduce the noise.

Algorithm 8 Update Landmark Complex

Input: List of landmarks S_i and the bearing measurement $\boldsymbol{\theta}_i$
Output: Landmark complex $\mathcal{L}$ and bearing lookup table BLT.

1: **function** UPDATELC($S_i, \boldsymbol{\theta}_i$)
2: **if** $\mathcal{L}$ and BLT not initialize **then**
3: $\mathcal{L} \leftarrow \emptyset, BLT = []$
4: **end if**
5: $\mathcal{L} \leftarrow \mathcal{L} \cup (\mathbb{P}(S_i) \setminus \emptyset)$
6: $BLT[S_i] = \boldsymbol{\theta}_i$
7: **end function**

Figure 5.6: Navigation Graph (b) is a partial $1-$skeleton of the *observation complex*, a dual complex of landmark complex (a). A vertex on navigation graph corresponds to the free simplex in the observation complex, while an edge between them corresponds to two set of observations that shares common landmarks. The number on the edges represents the dimension of sub-simplex which is the number of common landmarks subtract by one. Note that we don't include all simplices because the free simplex already encapsulates the observation of its faces in the navigation graph. This also significantly reduces the size of navigation graph.

We then construct the navigation graph $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$ which is a $1-$skeleton of the observation complex for exploiting the landmark complex. This graph can be viewed as a road map where each vertex represents a unique observation. To minimize the size of navigation graph, the vertex set $V_\mathcal{G}$ only consists of the *interesting* observations. An observation is interesting if it contains the unique combination of landmarks that cannot be observed elsewhere. We denote an interesting observation as *free simplex* in the landmark complex. The free simplex is a simplex that is not the face (or simply subset) of any other simplex.

$$V_\mathcal{G} = \{S_i \in \mathcal{L} |\ \forall S_j \in \mathcal{L} \setminus \{S_i\}, S_i \cap S_j \neq S_i\}$$

The edges are added for every pair of free simplices that share sufficient dimension of sub-simplex, n_l, required for navigation.

$$E_\mathcal{G} = \{(S_i, S_j) |\ S_i, S_j \in V_\mathcal{G}, S_i \neq S_j, |S_i \cap S_j| \geq n_l\},$$

where $n_l = 2$ for our holonomic robots.

Since we are only interested in the free simplices when constructing a navigation graph,

it is sufficient to add only the simplex representing current observation to the landmark complex in line 5 of Algorithm 8.

5.2.3 Cost-Utility Function

The cost function $C(S_i, \langle a, b\rangle)$ is defined as the minimum distance from a vertex representing the current configuration, S_i, to one of the vertices adjacent to the frontier, $\{S_j \in V_{\mathcal{G}} |\ \{L_a, L_b\} \subseteq S_j\}$. Since there is no actual metric information that can measure the real distance needed to traverse an edge, we choose the dimension of overlapping sub-simplex to estimate the difficulty of traversing it, i.e. the number of common landmarks between two observations subtract by one. Hence, the weight can be defined as:

$$\forall (S_i, S_j) \in E_{\mathcal{G}}, W(S_i, S_j) = \frac{1}{|S_i \cap S_j| - 1}. \tag{5.1}$$

Given the weighted graph, the cost function can then be calculated using any graph-based searching method such as Dijkstra's or A*. The output of the graph-based searching method is cost to reach each frontier and the best sequences of observations that can guide the robot there.

The utility function $U(\langle a, b\rangle)$ is defined as the potential information gain from exploring that frontier. One potential indicator is the number of unexplored frontiers that share common landmarks with it. Additionally, the robot should avoid exploring the frontier in the same area as other robots. Hence, we define the utility function as:

$$U(\langle a, b\rangle) = \frac{|\{\langle c, d\rangle \in \mathcal{F} |\ \{L_a, L_b\} \cap \{L_c, L_d\} \neq \emptyset\}|}{|\{R_i |\ A[i] \cap \{a, b\} \neq \emptyset\}|} \tag{5.2}$$

Hence, the cost-utility function is defined as

$$CU(S_i, \langle a, b\rangle) = U(\langle a, b\rangle) - \beta \cdot C(S_i, \langle a, b\rangle), \tag{5.3}$$

where $\beta \geq 0$ determines the relative importance between utility and cost function.

5.2.4 Task Execution

The task assignment outputs the target frontier $\langle a, b\rangle$ and the sequence of observations, $S_{i_0}, S_{i_1}, ..., S_{i_k}$, that guides a robot to the assigned frontier, where S_{i_0} corresponds to the current observation and S_{i_k} corresponds to the observation adjacent to the assigned frontier. The execution consists of two phases: navigating to S_{i_k} and then explore $\langle a, b\rangle$.

In phase one, the robot uses bearing-based controller to go through the sequence of observations where the desired bearing is recorded in the lookup table BLT. Note that the sequence of observations is simply the guidance. For each target observation S_{i_j}, the robot does not require to reach the exact observation. It can skip to the next target as soon as it observes enough landmarks to reach the next one. Once the robot reaches a configuration where it can observe both landmarks L_a, L_b, it continues to the next phase.

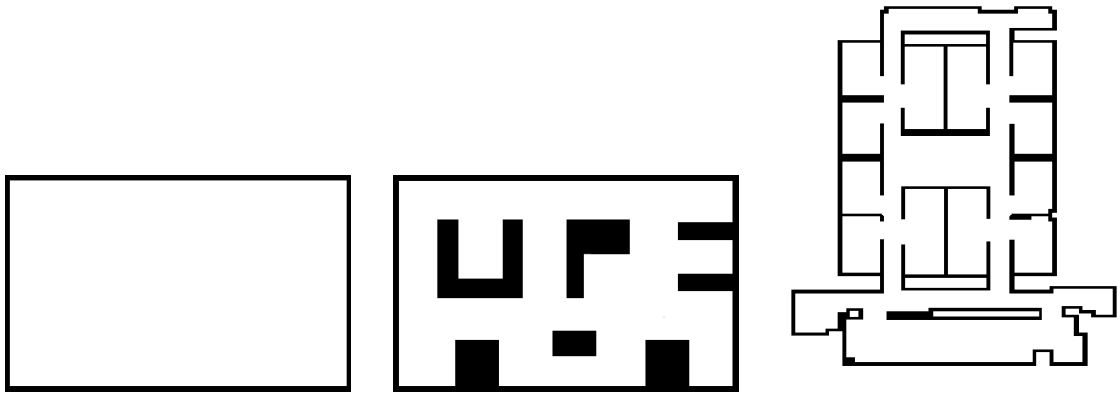
In phase two, the robot begins by approaching the regions that constitute the frontier, i.e. driving toward areas between two corresponding landmarks if it is on the opposite side. After arriving at the right region, the robot begins exploring the frontier by searching for other landmarks that can be observed from this region. This can be done using biased random walk to search locally. Each robot carries out phase two until either the frontier is identified or a time limit is reached, where time limit depends on the dispersion of the landmarks.

5.3 Statistical Analysis

In this section, we evaluated the performance of the landmark-based exploration through simulations on various setups using MATLAB. First, we demonstrate the performance of the proposed method in idealized scenarios where landmarks can be found anywhere in the environment and they are always sufficiently dense. Then, we consider the more realistic scenarios where landmarks are limited to the corners of the obstacles.

5.3.1 Idealized Scenario

In the idealized scenarios, we assume that the ideal landmark complex, i.e. a landmark complex constructed from a collection of all possible observations, can correctly capture all



(a) Obstacle-free Environment **(b)** Simple Environment **(c)** Complex Environment

Figure 5.7: Testing Environments: The topological exploration are evaluated on three different environments with teams of up to 30 robots.

topological features of the environment and the landmarks are sufficiently dense.

We first demonstrate the performance of the proposed method with various size of swarms in various environments. We then show that the landmark complex constructed by the proposed method can be used for navigating a robot to a vertex in the landmark complex. Lastly, we compare the performance of the proposed method with a random walk algorithm.

In all studies, we assume that the bearing measurement has the Gaussian noise $\mathcal{N}(0, \sigma)$ with $\sigma = \frac{\pi}{72}$, which means that the errors are within 5 degrees 95% of the time.

Exploration

We evaluate our proposed method on three different environments: obstacle-free, simple structure, and complex structure, based on two criteria: completion rate and execution time. In each environment, the landmarks are randomly generated with $\delta = \frac{r}{\sqrt{3}}$, averaging in $130, 260$, and 800 landmarks, respectively.

The completion rate is measured as the percentage of the coverage area corresponding to the simplices that have been observed, while the execution time is measured by the number of iterations, where each iteration represents one movement and one measurement for each robot.

With the simulation of 30 trials for each environment with random initial positions for the robots, the results are as the following. The completion rates are consistent across the

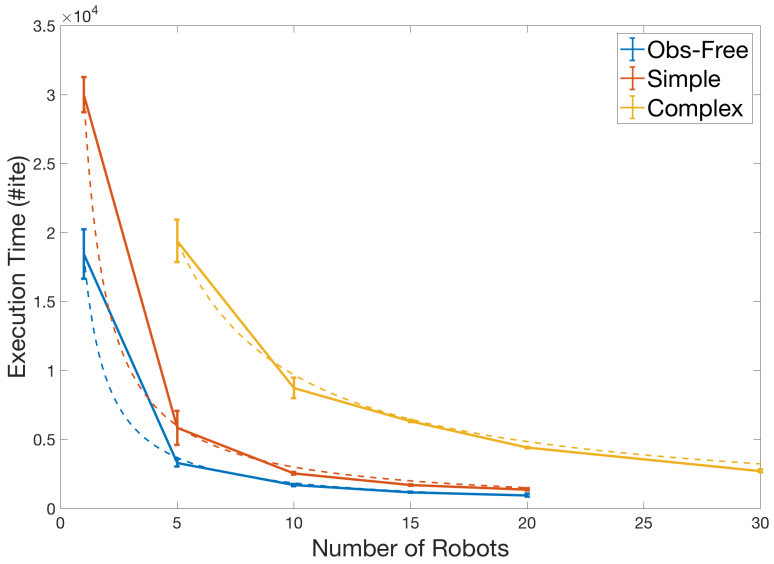


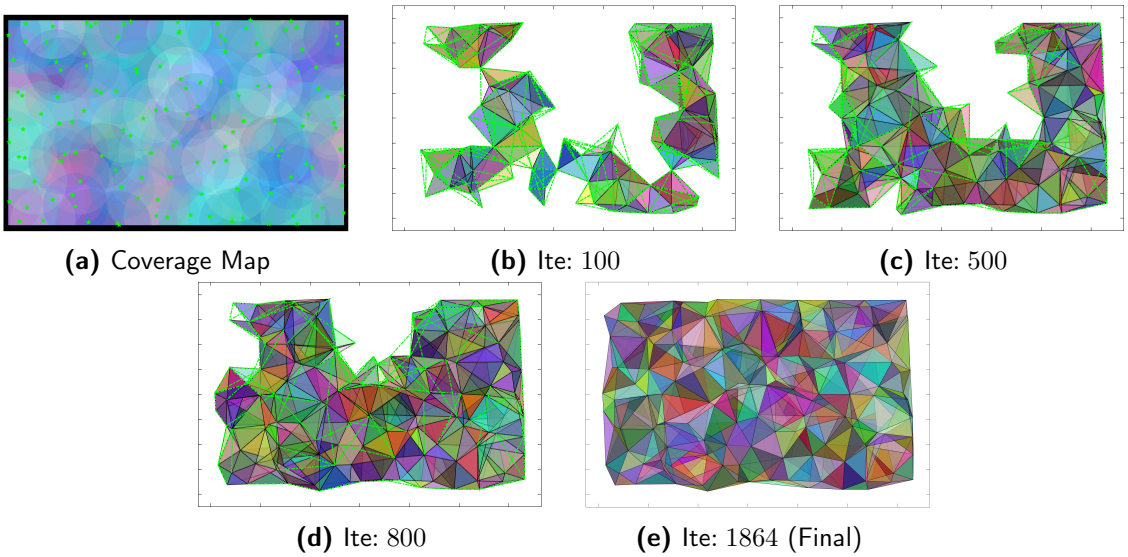
Figure 5.8: Execution Time vs. Number of Robots: The dashed-lines represent the expected execution time which is approximated by a rectangular hyperbola curve (number of robots × execution time = constant total workload).

size of robot swarms with an average of 98% on an obstacle-free environment, 95% on a simple structure, and 91.5% on a complex structure. On the other hand, the execution times signify the efficiency of exploration with team of robots as illustrated in Figure 5.8. The outcomes beat the expectation in most cases, where the expected execution time is approximated by a rectangular hyperbola curve (number of robots × execution time = constant total workload). The total workload is determined by the total execution times used by the smallest swarm size on each environment.

Nevertheless, we observed that the efficiency seems to be diminished beyond a certain size of the swarm which could have resulted from limited amount of traversal space or poor coordination between robots.

Figures 5.9, 5.10, and 5.11 illustrate the snapshots of geometric realization of the landmark complex constructed during exploration on all three environments. The geometric realization uses the approximated position of landmarks for the purpose of visualization only. The first top-left of each figure illustrates the coverage map of the landmarks, following by the updated landmark complex at various iterations. The coverage map is generated

by overlaying the sensing disk from each landmark with different color. The progress are smaller and smaller toward the end as robots need to traverse larger distance to explores different frontiers including the *false* ones, which do not contribute any new information.



(a) Coverage Map **(b)** Ite: 100 **(c)** Ite: 500

(d) Ite: 800 **(e)** Ite: 1864 (Final)

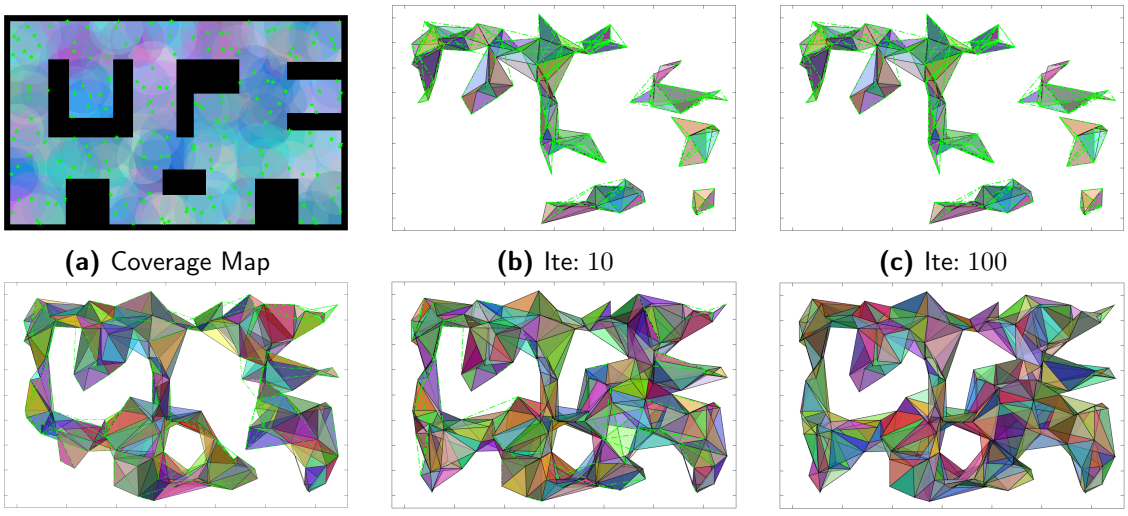
Figure 5.9: Obstacle-Free Environment: The algorithm finishes exploring the trivial environment filled with 133 landmarks using teams of 10 robots in 1864 iterations. Figure (a) illustrates the coverage map of all landmarks. Figures (b-e) show the geometric realization of the constructed landmark complex at different iteration, where green-dashed lines denote the unexplored frontiers. At the beginning, the landmark complex belongs to different homotopy class as it contains many connected components and holes. The disconnected components are linked and the holes are mended as more simplices are observed.

Exploitation

In this simulation, we demonstrate the exploitation of observation complex constructed by other robots in the past. The targets are given as the set of observation and we use the navigation graph to guide the robot there.

Comparison with Random Walk

In this section, we compare the performance of our proposed method with a random walk algorithm in the simple environment with 167 landmarks. In each trial, we first simulate our proposed method until termination and then use the final completion rate as the stopping criterion for the random walk algorithm. The simulated result over 30 trials has been shown



(a) Coverage Map **(b)** Ite: 10 **(c)** Ite: 100

(d) Ite: 500 **(e)** Ite: 1000 **(f)** Ite: 1483 (Final)

Figure 5.10: Simple structure: The constructed landmark complex successfully captures all topological features of the environment with non-trivial topology, i.e. the number of holes is equal to the number of obstacles. The exploration with teams of 20 robots is completed in 1483 iterations.

in Figure 5.13, where the task progress is determined by percentage of current completion rate over the final one. Up until 50% task completion, the performance of both methods are comparable. However, the proposed method is approximately three time more efficient when consider the full exploration task.

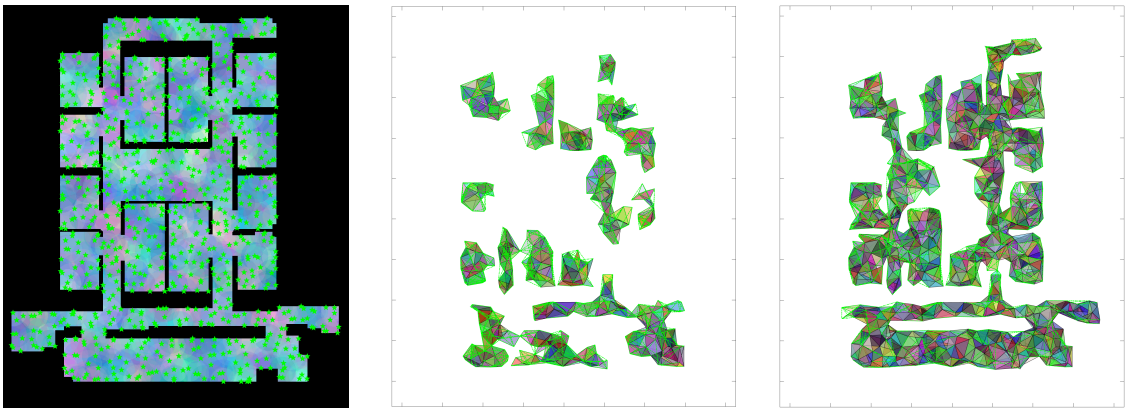
5.3.2 Benchmarking

We consider a more realistic scenario where the landmarks are the corners of the obstacles which can be detected by the existing methods in computer vision [46, 82, 85]. We then evaluate the proposed methods based on various characteristics of the environment. To simplify the experimental setup, each obstacle is represented by a rectangle and the environment is defined using the following parameters:

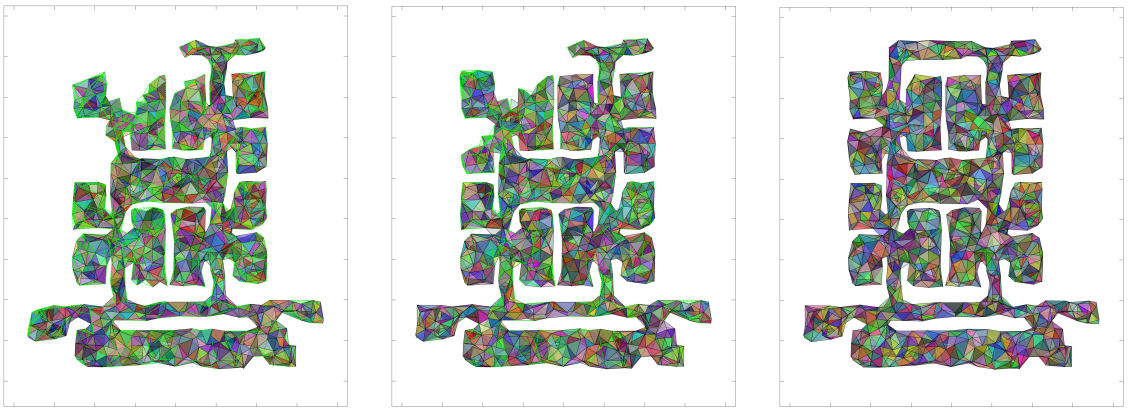
d : the average diameter of the obstacles

o : the ratio of the occupied spaces to the entire environment

r : the sensing radius of the robots



(a) Coverage Map **(b)** Ite: 100 **(c)** Ite: 500



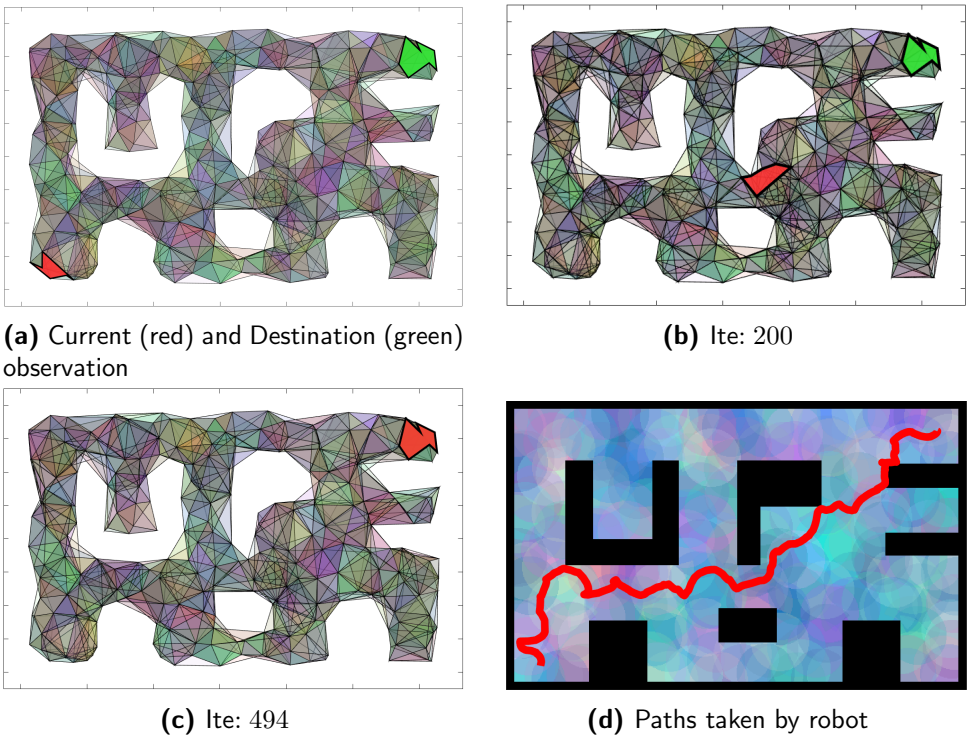
(d) Ite: 1000 **(e)** Ite: 1500 **(f)** Ite: 2373 (Final)

Figure 5.11: Complex Environment: The proposed method is evaluated on the large map resembling the building floor plan filled with 869 landmarks using team of 30 robots. The exploration is completed in 2373 iterations

We simulate the experiment on three different setups where each of the parameters become an independent variable and then evaluate them based on the accuracy of the constructed landmark complex. Additionally, the control policy is modified such that each robot will terminate its mission if it does not observe any landmark in order to limit the noises from random behavior.

Accuracy of Landmark Complex

In realistic scenario, the landmark complex may not be able to correctly capture the topology of the environment. Hence, we define the accuracy of the constructed landmark complex as the function of the coverage rate (previously defined as the completion rate), which is



(a) Current (red) and Destination (green) observation

(b) Ite: 200

(c) Ite: 494

(d) Paths taken by robot

Figure 5.12: Navigation Example: Robot exploits the landmark complex constructed by other robots for navigation from initial simplex to goal simplex.

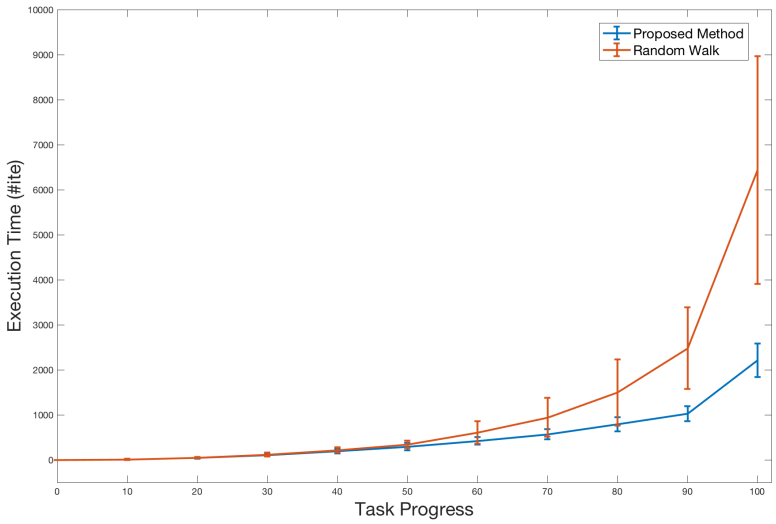


Figure 5.13: The comparison between the execution time of the exploration task between our proposed method and the random walk algorithm. Up until 50% task completion, the performance of both methods are comparable. However, the proposed method is approximately three time more efficient when consider the full exploration task.

the proportion of free space covered by the geometric realization of the landmark complex, and the percentage of holes/obstacles correctly captured by the landmark complex. Let ρ denote the completion rate and κ denote the ratio of the holes in the landmark complex (the first Betti number) to the number of disconnected obstacles in the environment then the accuracy can be defined as

$$Acc = \exp(-a(\kappa - 1)^2)\rho.$$

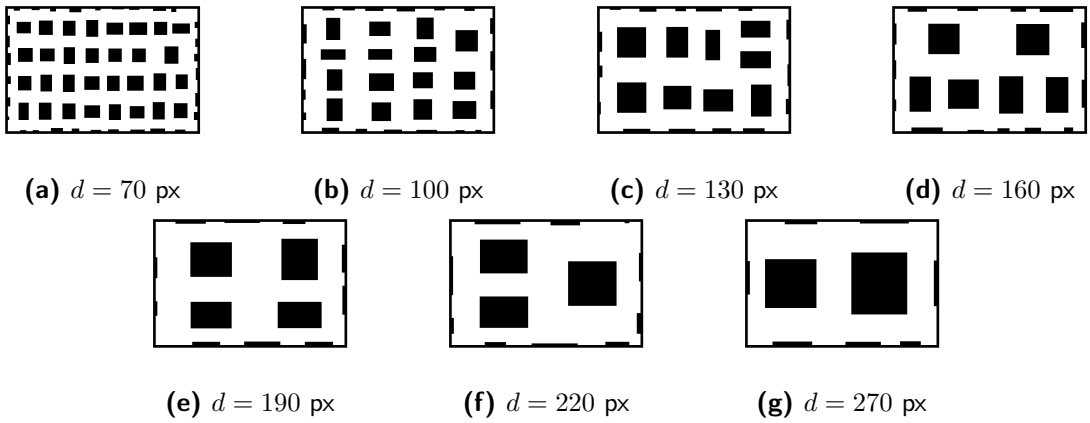
The exponential term has a value between 0 and 1 where a determines the penalty rate of the missing holes and the extra holes created by the landmark complex. Hence, $Acc = 1$ if and only if $\rho = \kappa = 1$, i.e. the landmark complex correctly captures all the obstacles in the environment and its geometric realization covers the entire free space.

In all simulations, the dimension of the environment is set to be 400×640 pixels^2. The result is collected from 50 randomly generated maps for each set of parameters. For the evaluation, a is set to be 5 which will deduce the accuracy rate to $50\%, 25\%$, and 5% if the landmark complex misses 35%, 50%, and 80% of the holes, respectively.

Exp-1: Diameter of Obstacles

In the first setup, we fix the occupancy ratio to 0.3 and the sensing radius to 80 pixels while set the average diameter of obstacles to various values between 70 to 270 pixels as illustrated in Figure 5.14.

According to the simulation results, both the coverage rate and the accuracy of landmark complex are low in the environments filled with large obstacles since there are not enough landmarks for navigation and mapping. The coverage rate increases as the diameter of obstacles decreases as more obstacles lead to larger number of landmarks. However, the accuracy peaks at the diameter of 130 pixels, which is slightly lower than twice the sensing radius, and then sharply drops afterward due to the misdetection of small obstacles. In general, the landmark complex may misdetect any obstacle that is smaller than twice the sensing radius. However, due to the geometrical features of the rectangular obstacle, the



(a) $d = 70$ px **(b)** $d = 100$ px **(c)** $d = 130$ px **(d)** $d = 160$ px

(e) $d = 190$ px **(f)** $d = 220$ px **(g)** $d = 270$ px

Figure 5.14: Examples of test environments with various diameters (d) of obstacles in pixels (px).

landmark complex will only misidentify it if a robot can observe any three corners simultaneously. Thus, any obstacle with diameter larger than $\sqrt{2}$ times the sensing radius should be detected. As a result, the landmark complex only detects some of the obstacles with the diameter of 100 pixels and none of the obstacles with the diameter of 70 pixels.

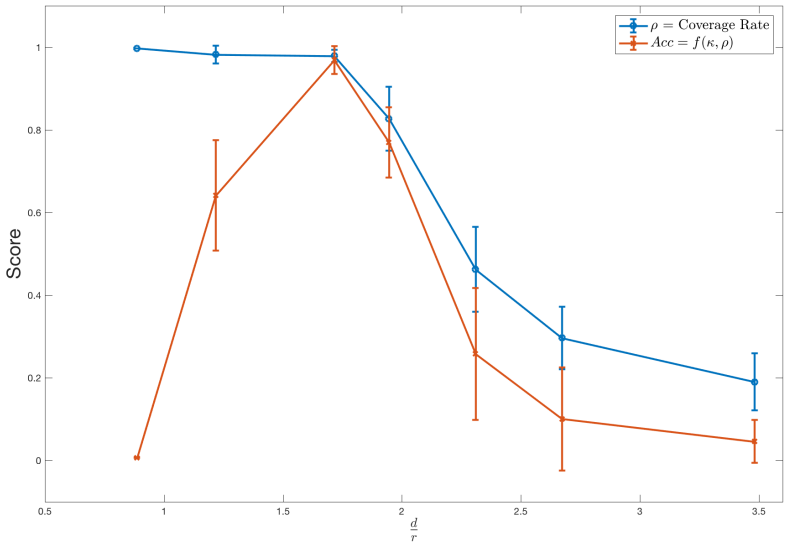
Exp-2: Ratio of Occupied Space

In the second setup, we fix the diameter of obstacles to 130 pixels and the sensing radius to 80 pixels while set the ratio of occupied space to various values between 0.1 to 0.5 as illustrated in Figure 5.16.

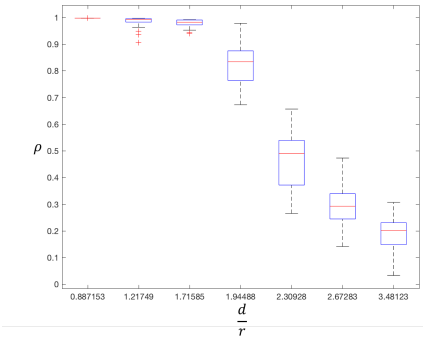
With low occupancy rate, there are not enough obstacles (and landmarks) for navigation and mapping, resulting in the low score in both the coverage rate and the accuracy. As the ratio of occupied spaces increases, there are more obstacles and landmarks in the environment, leading to a better coverage rate. Similarly, the score in the accuracy goes up as the landmark complex gain more coverage as most of the obstacles are correctly detected ($d > \sqrt{2}r$). Note that the accuracy slightly goes down for cluttered environment due to some narrow corridors that the robots fail to discover.

Exp-3: Sensing Radius

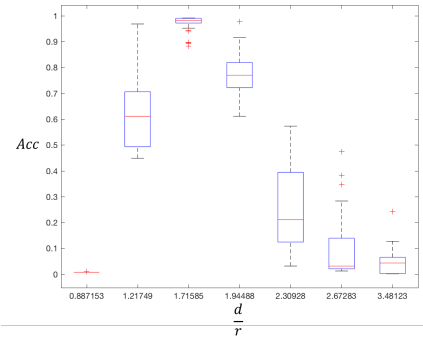
In the last setup, we use the third set of maps from the first setup where the diameter of obstacles is around 130 pixels and the ratio of occupied space is 0.3 and set the sensing



(a) Performance VS Diameter of Obstacles



(b) Box plot of ρ



(c) Box plot of Acc

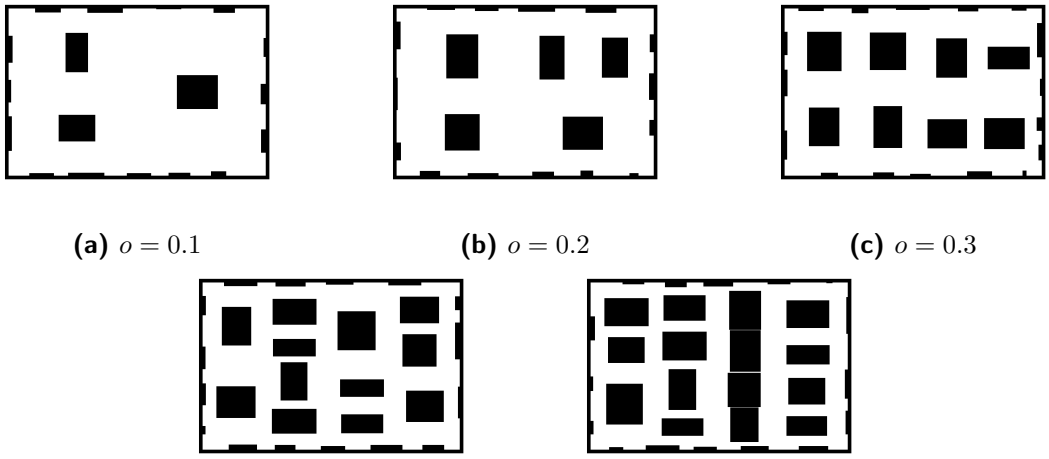
Figure 5.15: The performance of the proposed method on test set 1.

radius to various values between 40 to 200 pixels.

The coverage rate increases along with the sensing radius while the accuracy sharply drops after the sensing radius grows pass 80 pixels. As the sensing radius increases, the obstacles become relatively small and are hence misdetected by the landmark complex.

5.4 Alternative Control Strategies in Presence of Coarse Range Measurement

In this section, we discuss the adjustments that could address some of limitations in our proposed method.



(a) $o = 0.1$ **(b)** $o = 0.2$ **(c)** $o = 0.3$

(d) $o = 0.4$ **(e)** $o = 0.5$

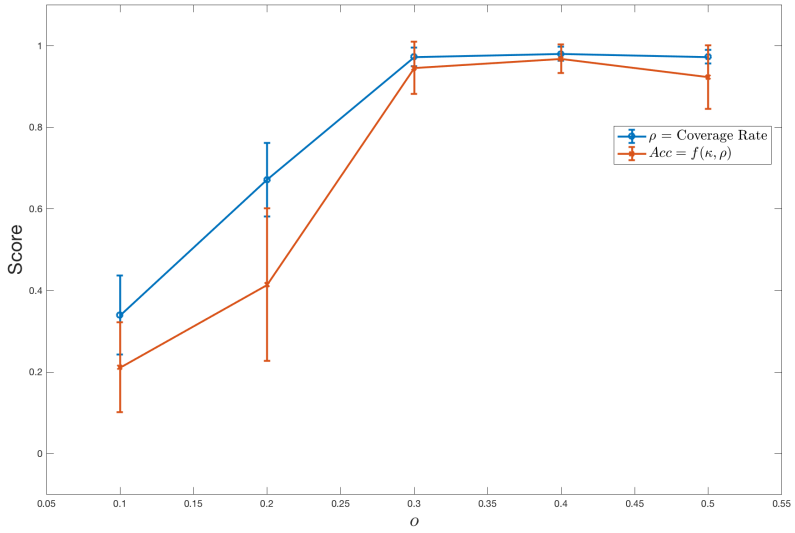
Figure 5.16: Examples of test environments with various ratios of occupied spaces.

5.4.1 Unscaled Distance Exploration Strategy

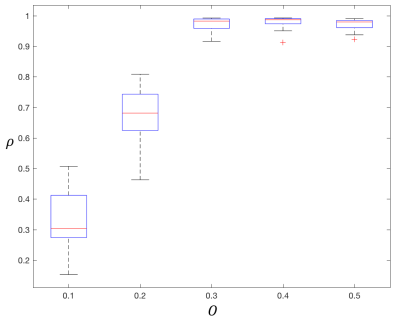
Although the basic strategy performs reasonably well with the bearing-only controllers, there are some downsides that can be avoided with the additional information derived from consecutive bearing measurements such as the minimum density of landmarks and the requirement for global compass direction.

Unscaled Distance Estimation

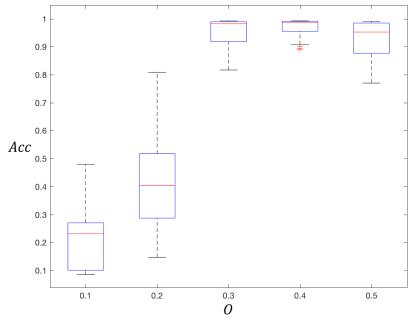
The distance toward landmark i can be estimated from two consecutive bearing measurements and the heading direction using the law of Sines as illustrated in Figure 5.19. By utilizing this information, we propose a revised exploration strategy that guides a robot to follow the sequence of line segments that are either equidistant to each pair of landmarks or aligned with the obstacles. Coincidentally, the traversal path of the robot is similar to the bounded Voronoi graph (BVG), a boundary of the bounded Voronoi diagram (BVD) as illustrated in Figure 5.20. The bounded Voronoi diagram is a Voronoi diagram where the visibility of each generator is constrained by the obstacles [35]. Before going into the details of the exploration strategy, we first introduce the controller that guides to robot along the equidistant line between a pair of landmarks.



(a) Performance VS Ratio of Occupied Space



(b) Box plot of ρ



(c) Box plot of Acc

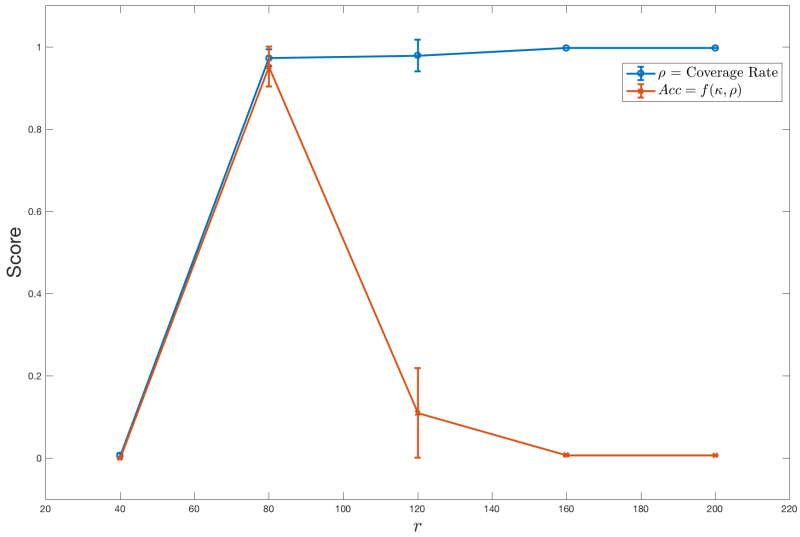
Figure 5.17: The performance of the proposed method on test set 2.

Controller

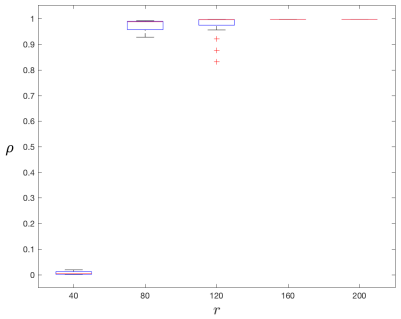
Given L_a and L_b as the pair of reference landmarks, the unscaled vector toward landmark L_i is defined as

$$\vec{i} = \frac{y_i - x}{C}, \quad i \in \{a, b\}$$

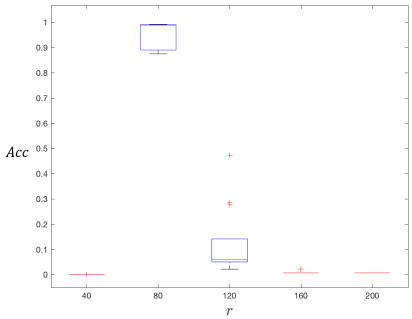
where y_i denote the position of landmark L_i, x denote the position of the robot, and C denote the unknown scaling factor due to unknown travel distance. Note that $\vec{i}$ can be estimated directly as previously described and C is constant across landmarks for each observation.



(a) Performance VS Sensing Radius



(b) Box plot of ρ



(c) Box plot of Acc

Figure 5.18: The performance of the proposed method on test set 3.

The control input

$$u_0 = \frac{\vec{a}\|\vec{a}\| + \vec{b}\|\vec{b}\|}{\|\vec{a}\|^2 + \|\vec{b}\|^2}$$

will drive the robot toward the center point between two landmarks as illustrated by the plot of vector field in Figure 5.21a.

Let

$$\hat{c} = \frac{\vec{b} - \vec{a}}{\|\vec{b} - \vec{a}\|}$$

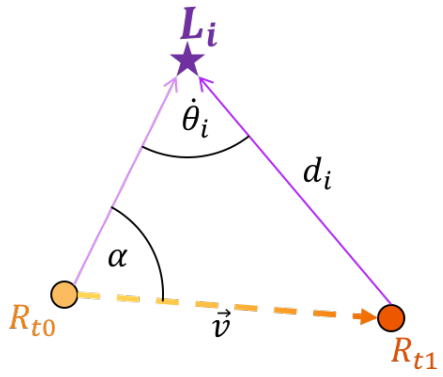
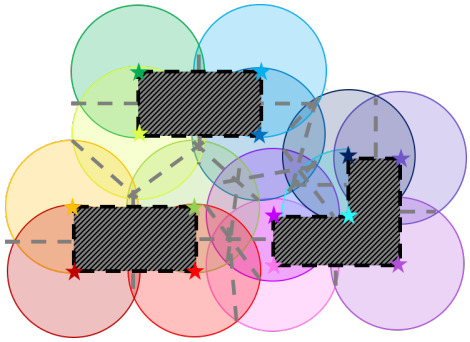
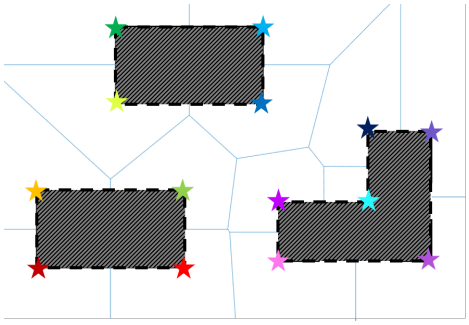


Figure 5.19: Assuming that a robot is moving in a straight line, an unscaled distance toward the landmarks can be estimated from consecutive bearing measurements using the law of Sines, *i.e.*, $\tilde{d}_i = \frac{d_i}{|\vec{v}|} = \frac{\sin(\alpha)}{\sin(\dot{\theta}_i)}$. Since the distance traveled $|\vec{v}|$ is unknown, we cannot calculate the exact the distance toward landmark d_i.



(a) The gray dashed lines represent the bisectors of the overlapping regions corresponding to the 1-simplices.



(b) The bounded Voronoi graph, a boundary of the bounded Voronoi diagram that is constructed with landmarks at the corner obstacles, is a subset of the bisectors of overlapping regions in 1-simplices.

Figure 5.20: The environment can be explored by traversing a sequence of line segments equivalent to bounded Voronoi graph and the boundary of the obstacles.

denote the unit vector from L_a to L_b and

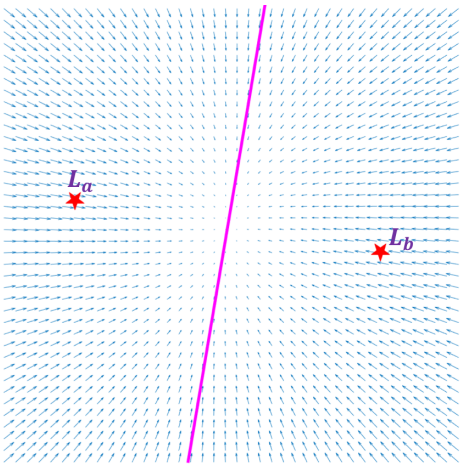
$$\hat{c}^{\perp} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \hat{c}$$

denote the perpendicular vector of $\hat{c}$ that is aligned with the desired heading direction. By projecting a vector u_0 onto $\hat{c}$ and $\hat{c}^{\perp}$, we can manipulate the output to get the desired

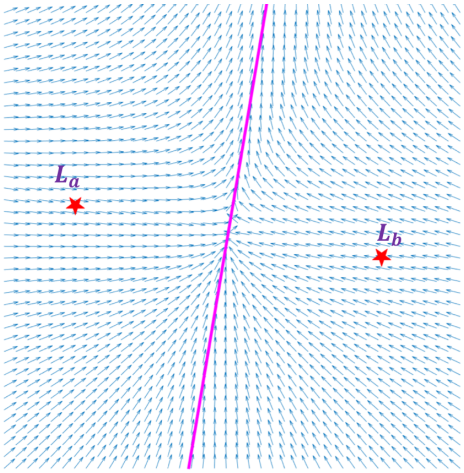
behavior such as the following control input

$$u = k(u_0 \cdot \hat{c})\hat{c} + |u_0 \cdot \hat{c}^{\perp}|\hat{c}^{\perp}.$$

The first term will drive the robot toward the equidistant line between two landmarks where k determines the convergence rate, while the second term will force the robots to move along the line segment in the desired direction. Note that u becomes $\vec{0}$ as $u_0 = \vec{0}$ at the midpoint between L_a and L_b, which is not desirable. However, we can simply set u to be $\hat{c}^{\perp}$ when $u_0 = \vec{0}$ to fix this issue. Additionally, a control input u can be treated as a unit directional vector since a robot does not have any odometry information, as illustrated in Figure 5.21b.



(a) $u_0 = \frac{\vec{a}\|\vec{a}\| + \vec{b}\|\vec{b}\|}{\|\vec{a}\|^2 + \|\vec{b}\|^2}$



(b) Unit vector of $u = k(u_0 \cdot \hat{c})\hat{c} + |u_0 \cdot \hat{c}^{\perp}|\hat{c}^{\perp}$ with $k = 1$

Figure 5.21: The plots of vector field: (a) u_0 guides a robot to the center between two landmarks ,denoted by two red starts; (b) u guides a robot toward the equidistant line, denoted by a magenta line, while also heading toward the desired direction (upward).

Revised Exploration Strategy

In the revised strategy, our goal is to construct our landmark complex by traversing an bounded Voronoi graph (BVG). Hence, we assume that the corners of the obstacles will be detected as landmarks in order to satisfy the condition for constructing BVG. We will mainly consider the scenario where the landmarks are at the corner of the obstacles. However, the

same strategy can handle the environment with additional landmarks in the free space as well since adding more landmarks will increase the connectivity of the graph and generally make it easier to traverse.

- *Frontier Identification:* Since our goal is to traverse the BVG, we can ignore all other edges that are not part of the BVG using the relative distance information. Given the pair of successive bearing measurements, we first embed the landmarks that have been observed consecutively into a local coordinate frame and then determine whether the current position is on the BVG. To be part of BVG, the robot, which locates at the origin, should form an isosceles triangle with a pair of the nearest landmarks. This condition can be relaxed to a pair of the nearby landmarks, i.e. within certain distance ratio of the closet landmark, to allow room for error in bearing measurement. Let $\vec{a}, \vec{b}$ be the vectors toward a pair of nearest landmarks in the local coordinate frame. Then $\triangle OAB$ is formed by connecting $\vec{a}$, $\vec{b}$ and $\vec{o} = \vec{b} - \vec{a}$, where I is the angle opposite to $\vec{i}, i \in \{o, a, b\}$. $\triangle OAB$ is an isosceles triangle if and only if $A = B$. Hence, we define the evaluation function $\varphi : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}_{\geq 0}$ as

$$\varphi(\vec{a}, \vec{b}) = \exp(|A - B|) + \frac{\|\vec{a}\| + \|\vec{b}\|}{2\delta} - 2,$$

  where δ is the distance toward the nearest landmark in the current observation. This φ function measures how close the triangle is to the ideal isosceles one. The first term penalizes the triangle with unbalanced base angles while the second term penalizes the pair of landmarks that are not the nearest ones. Note that the term δ is used for normalizing the unscaled distance measurement. We can then determine whether a pair of landmarks should be explored using the threshold on φ. For instance, the color map in Figure 5.22 displays the minimal value of φ among all pairs of visible landmarks at each location (with the upper limit at 5 in the color map). It can be seen that the regions with minimal values form a graph similar to BVG that can be utilized in exploring the environment.

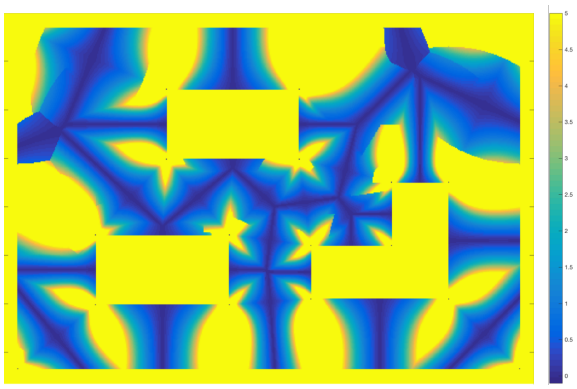


Figure 5.22: The color map displays the minimal value of φ among all pairs of visible landmarks at each location.

- *Control Policy:* The proposed controller allows the robot to move along the bisector of the overlapping region between two landmarks in either direction which is sufficient to traverse the majority of the BVG as illustrated in Figure 5.20. Nevertheless, the BVG may not be fully connected in the region with sparse landmarks such as those near the long obstacle/wall. Hence, the robot may need to traverse along the boundary of the obstacles to ensure that the environment is fully explored when the BVG intersects with the obstacles. There are three scenarios where the BVG intersects with the obstacles as illustrated in Figure 5.23. The robot needs to explore the obstacles in the directions where the corners have not been observed yet. Additionally, during initialization (and also exploration if the bearing measurement is too sparse), the robot may need to reorient itself with the BVG if it only observes one landmark or situates too far from the BVG. For instance, the robot may circle around the landmarks outwardly if it only observes one landmark or increase the value of k in the control input u to push the robot toward BVG if it can observe more than two landmarks.

- *Navigation Graph:* One of the downsides of this strategy is that the navigation graph cannot be constructed directly from the landmark complex since it does not contain any directional information. Additionally, some of the 1-simplices are not parted of the BVG. Hence, the navigation graph has to be constructed iteratively during the exploration process. Let the navigation graph be a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where

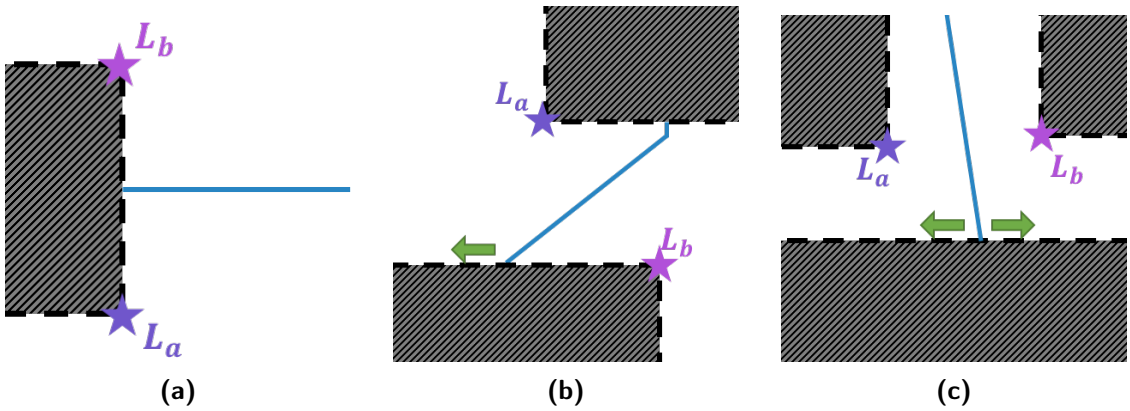
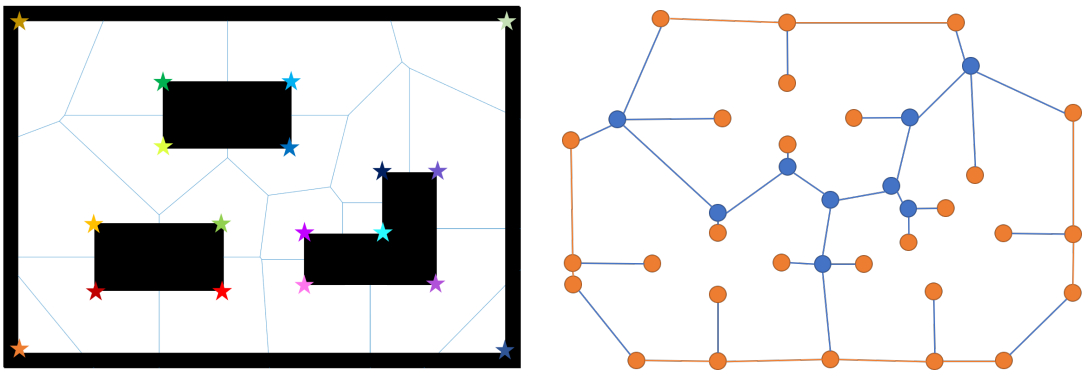


Figure 5.23: The are three scenarios where BVG intersects with obstacles. In scenario (a), both landmarks are the corner of the intersecting obstacle, so the robot does not need to explore any further. In scenario (b), one of the landmark is the corner of the intersecting obstacle, so the robot needs to explore along the obstacle in the direction opposite to observed corner. In scenario (c), both landmarks do not belong to the intersecting obstacle, hence the robot needs to explore in both direction.

the vertex ,$v \in \mathcal{V}$ corresponds to the junction of paths in the BVG or the intersection of BVG with the obstacles and the edge in $e \in \mathcal{E}$ corresponds to the path along the BVG or the boundary of the obstacles, as illustrated in Figure 5.24. The navigation graph of the BVG consists of the vertices, where the blue ones correspond to the junctions of paths on BVG and the orange ones correspond to the intersection with obstacles, and the edges, where the blue lines correspond to the movement along the BVG and the orange lines correspond to the movement along the boundary of the obstacles. The number of blue nodes and edges in the navigation graph might vary depending on the threshold we use for φ. Additionally, we can set the weight of the orange edges to be much higher to avoid the movement along the obstacles while maintaining the connectivity of the graph.

Discussion

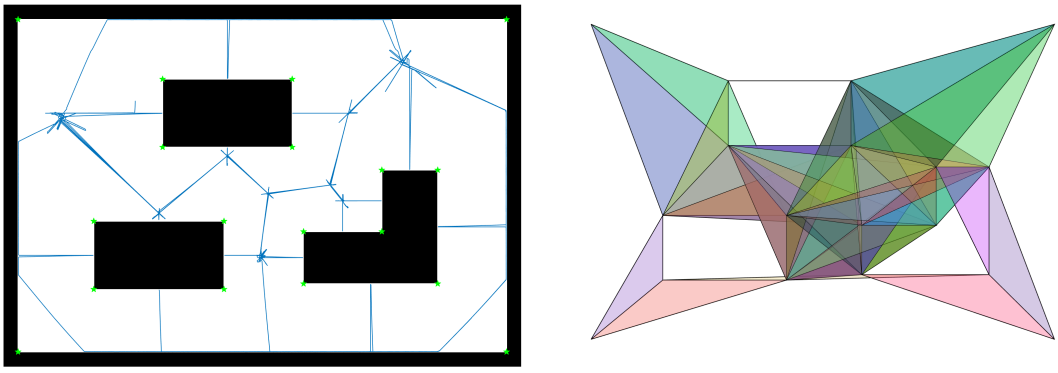
Using a simple greedy algorithm, we simulate the exploration with the revised strategy. The result shows that the robot can explore and construct the topological map of the environment by following the paths that either belong to the BVG or the boundary of the obstacles as illustrated in Figure 5.25.



(a) The BVG of the environment.

(b) The navigation graph of the BVG.

Figure 5.24: The navigation graph of the BVG consists of the vertices, where the blue ones correspond to the junctions of paths on BVG and the orange ones correspond to the intersection with obstacles, and the edges, where the blue lines correspond to the movement along the BVG and the orange lines correspond to the movement along the boundary of the obstacles.



(a) The paths traversed by the robot during the exploration.

(b) The landmark complex constructed from observations along the traversed paths.

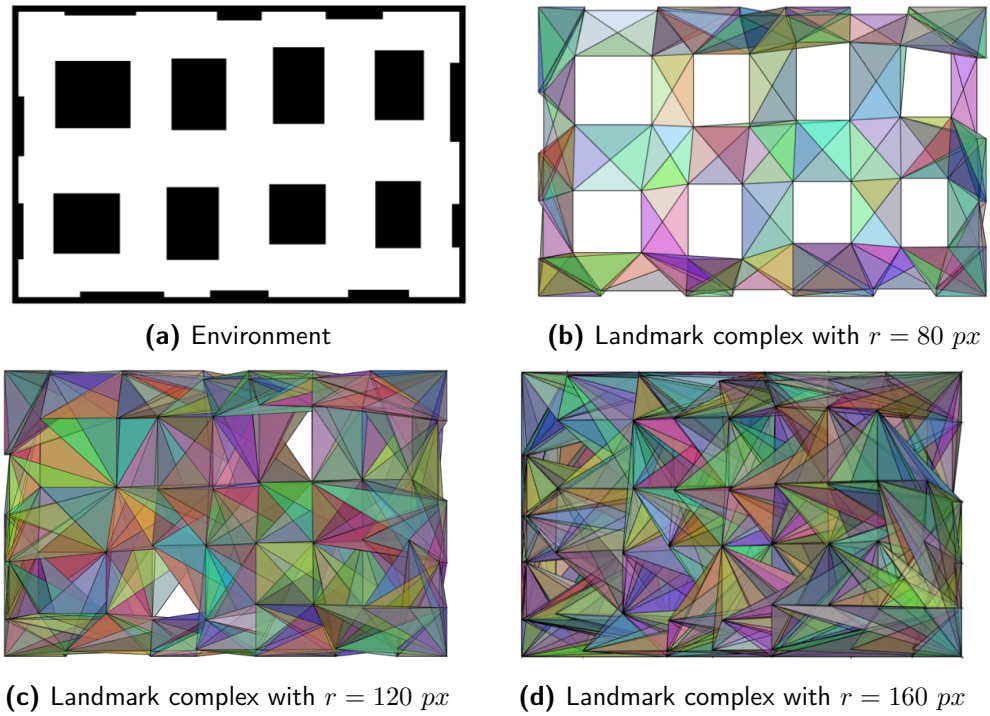
Figure 5.25: Using the revised strategy, the robot successfully explores the environment and constructs the right landmark complex.

There are three advantages of the revised strategy. First, it has much higher success rate compared to the basic strategy when operating in the environment filled with sparse landmarks since the revised strategy uses little to no random walk process during the exploration. Second, there is no need for global compass direction and the bearing lookup table since the robot can use the direction from the navigation graph with proposed controller to navigate the environment. Additionally, the motion along the BVG is guarantee to be collision-free. Lastly, the execution time is significantly reduced since the robot no longer

needs to explore all possible frontiers, where some of them are very difficult to observe. Nevertheless, the use of relative distance information for both exploration and exploitation causes the revised strategy to be more sensitive to noise.

5.4.2 Misdetection of Holes/Obstacles

Until now, we have been focusing on addressing the limitations of the landmark-based navigation. In this section, we consider an adjustment on the definition of landmark complex to address some of its limitation.



(a) Environment **(b)** Landmark complex with $r = 80\ px$

(c) Landmark complex with $r = 120\ px$ **(d)** Landmark complex with $r = 160\ px$

Figure 5.26: Using the corners of the obstacle as landmarks, the landmark complex correctly captures all the obstacles with sensing radius of 80 pixels but fails to identify the obstacles as the sensing radius increases to 120 and 160 pixels.

In the realistic scenario, we observe that the constructed landmark complex only accurately determine the topology of the environment in a very limited ranges of parameters as illustrated in Figure 5.26. For the scenario with insufficient landmarks, the accuracy of the landmark complex is low since it cannot cover an entire environment and thus it is the physical limitation on the sensing capability. On the other hand, for the scenario with plenty of landmarks, the accuracy of the landmark complex still sharply fells when the obstacles be-

come relatively small compared to the sensing radius. In Figure 5.26, the landmark complex nicely represents the environment with the sensing radius of 80 pixels but fails to identify most and all of them at the sensing radius of 120 and 160 pixels respectively. The issue lies in the non-convexity of the domain of visibility of each landmark. For rectangular obstacle, the hole, corresponding to the obstacle, will not be detected by the landmark complex if there exists an observation that can see its antipodal corners as illustrated in Figure 5.27.

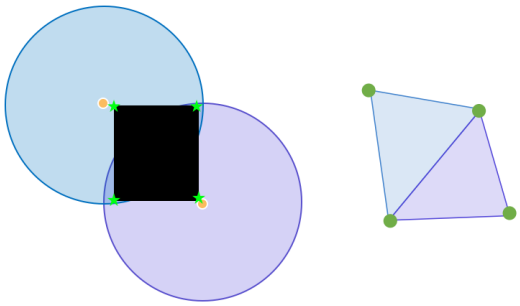


Figure 5.27: The hole, corresponding to the obstacle, will not be detected by the landmark complex if there exists an observation that can see its antipodal corners.

According to the definition in [43], the landmark complex is the nerve of the cover of the collection of observations that see at least one landmark. By Dowker duality, the landmark complex is homotopy equivalent to the observation complex, which is the nerve of the cover of the landmarks. Hence, the landmark complex is homotopy equivalent to the free-space as long as the nerve of the cover of the landmarks is. Nevertheless, the nerve lemma, which proves the homotopy equivalent relation, only applies to weakly convex sets in Euclidean space.

One may consider the cover of landmark l to be the regions in the free space that can observed l, i.e. its domain of visibility. Due to the occlusion of obstacle, the domain of visibility at each corner of the obstacle is non-convex and hence the underlying nerve cover may not be homotopy equivalent to the free-space.

In this section, we propose an alternative definition of the landmark. Instead of using corners as the landmarks, we redefine the landmark to be an edge of the obstacle, where a pair of corners that constitute the edge and the regions between them are fully visible as shown in Figure 5.28. This is simply an edge detection, which can be done with the existing

techniques in computer vision. With an edge of the obstacle as a landmark, the domain of visibility for this new landmark becomes a half plane, which by itself is convex.

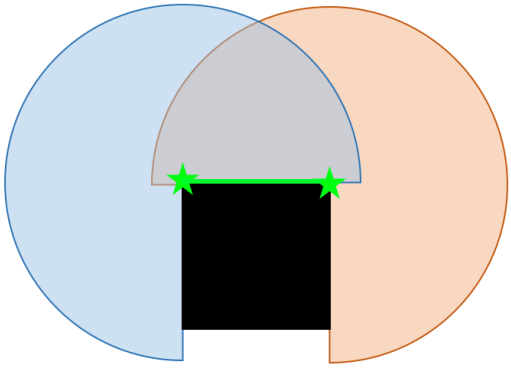


Figure 5.28: The domain of visibility of an edge is the intersection of the domain of visibility of the corners which becomes a half plan for infinite sensing radius.

Nevertheless, the occlusion of other obstacles can still cause the domain of visibility to be non-convex. As a result, some of the smaller obstacles can still be misdetected. For instance, if there exists a location where an edge is completely visible along with another fixed unoccluded edge (of a different obstacle) for every edge of an obstacle, then that obstacle will not be detected. Figure 5.29 illustrates such example where $\forall b \in \{b_1, ...b_4\}$, there exist a location such that b is visible along with g_1. Thus, g_1 will cone out the blue obstacle. Nevertheless, in practice, this limitation could actually be useful in filtering out the small objects in the environment such as chair or table.

Using the result from data set 3, we recompute the accuracy of the landmark complex using edges as the landmarks. The accuracy significantly improves for a larger sensing radius, as illustrated in Figure 5.30. However, this method also increases the minimum sensing radius as it requires a longer-range sensor to observe an entire edge.

An alternative solution would be to redefine the landmarks as the set of edges that can be simultaneously observed and belong to different obstacles. For instance, given an environment with two obstacles in Figure 5.29, the set of landmarks would be all unique pairs of edges from each obstacle. The domain of visibility of this new landmark is the intersections of all half planes without any occlusion since the occluded edge must be part of the landmark and hence convex. However, this method can quickly become intractable

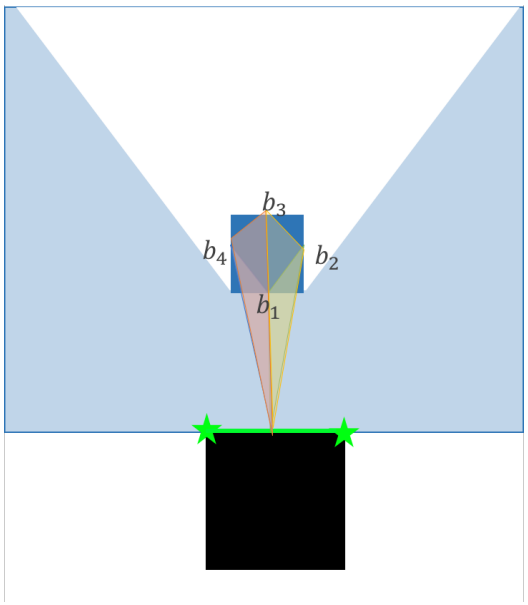


Figure 5.29: $\forall b \in \{b_1, ...b_4\}$, there exist a location such that b is visible along with g_1. Thus, g_1 will cone out the blue obstacle.

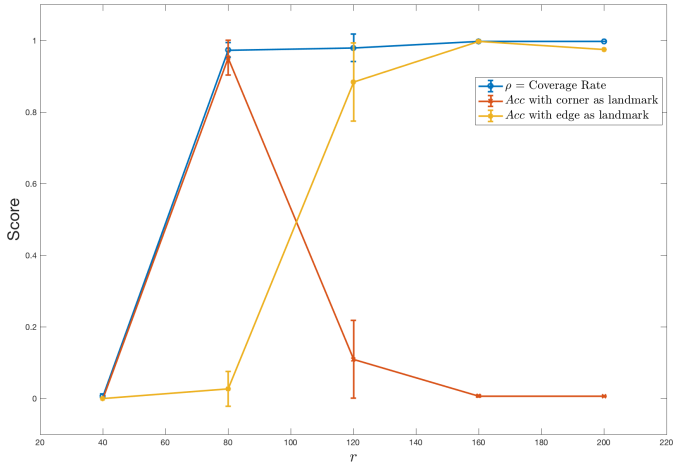


Figure 5.30: By replacing the corner with the edge as landmark, the accuracy significantly improves for a larger sensing radius. However, this also increase the minimum sensing radius as an edge requires a longer sensing range to observe.

as the sensing radius increases since the number of simplices grows exponentially with the number of obstacles.

5.5 Conclusion

This chapter first presents the metric-free exploration algorithm for the swarms of limited sensing capabilities using a topological representation of an environment. Each robot is equipped with an omni-directional, limited range sensor that can uniquely identify landmarks in its neighborhood and measure their bearing angles for local navigation. The landmark complex, a simplicial complex that encapsulates the topological information of the environment, is constructed based on the observation of identifiable landmarks. With sufficiently dense landmarks, we demonstrate the performance of our proposed method on three environments with different structures in simulations. Additionally, the constructed landmark complex can be further exploited for future navigation. We then present an alternative exploration strategy in the presence of coarse distance measurement. Using the coarse distance information, the robot can follow the paths that either belong to the bounded Voronoi graph or align with the obstacles to completely explore an environment. Lastly, we discuss some solutions to address the misdetection of the obstacles by the landmark complex.

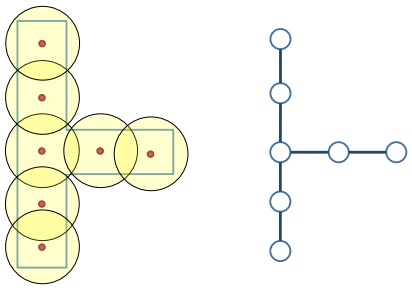
Although our strategies are implemented in a centralized manner, similar results can be achieved in a decentralized implementation if all robots can maintain a communication link with each other. Since the map representation is simply a set of landmarks' ids, which is sparse and compact, the communication bandwidth is not an issue. Of course, resource-constrained robots will also have limited range radios and may not be able to form a connected graph. The study of decentralized exploration algorithms and creating the landmark complex would be an interesting direction to explore.

Chapter 6

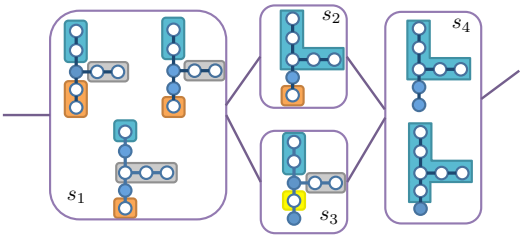
Pursuit-Evasion

There are many variations of the pursuit-evasion as described in 2.3.4. This chapter addresses the pursuit-evasion where a team of coordinated pursuers needs to search a given environment for an unknown number of evaders or targets. Pursuit-evasion formulation can be used to represent many useful applications such as surveillance or search and rescue. For instance, consider the problem of deploying a team of mobile sensing robots to patrol a military base in order to detect any intruders breaking into the base, or to search for survivors after a disaster. In such scenarios, pursuers must take into account the fact that evaders are mobile and may avoid being detected; they may also know the location of all pursuers at all times and may move faster than the pursuers. As a result, simply checking all areas is not sufficient and one needs to generate sophisticated pursuit strategies. Without any assumptions regarding the number, the speed, or the maneuverability of evaders, we design a general algorithm for automatically computing the strategy that pursuers should follow for detecting all evaders.

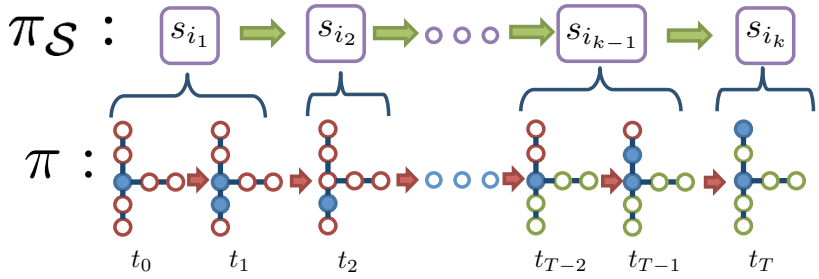
In this chapter, we propose an alternative approach for solving the worst-case adversarial pursuit-evasion problems where multiple pursuers equipped with limited-range sensors are used to detect and capture all possible mobile evaders in a given environment. Our main objective is to design a general algorithm for automatically computing the strategy that pursuers should follow for detecting all evaders. First, we demonstrate the proposed frame-



(a) A graph representation is constructed for the given environment and the pursuers' sensor model



(b) The entire of configuration space of $N = 2$ pursuers is partitioned into a set of abstraction states (purple boxes), where subset of them are being shown s_1, s_2, s_3, s_4 above. This abstraction is described in section 6.2. An edge occurs between two abstraction states s_1 and s_2 because it is possible to go in one move of a pursuer, from a concrete member state of s_1 to a concrete member state of s_2.



(c) The hierchical planner synthesizes a strategy as a sequence of abstraction actions (π_S) and then refine π_S into a sequence of actions (π) that can be executed by the pursuers for $N = 2$.

Figure 6.1: Illustration of the main steps for synthesize the solution strategy for pursuit-evasion problem with our framework.

work on the metric map and then explain how to apply similar framework on the landmark complex constructed in Chapter 5.

Part of the research contained in this chapter was originally published in [77].

Overview

Given inputs as a map of the environment and sensor models for the pursuers, we obtain a graph representation of an environment using the Čech Complex. Even with such representations, the configuration space grows exponentially with the number of pursuers. In order to address this challenge, we propose an abstraction framework to partition the configuration space into sets of topologically similar configurations that preserve the space of possible evader locations.

The essence of our approach is illustrated in Figure 6.1. Our algorithm takes as input a map representing the environment, a sensor model of the pursuers, and the number of pursuers. Using the sensor model of the pursuers, we first construct a graph representation of the environment as shown in Figure 6.1a. Next, we formulate the configuration space of the pursuers using the graph representation and the number of pursuers N and partition it into the set of abstract states (Figure 6.1b). Finally, we synthesize the strategy as a sequence of abstract actions and then perform the refinement step to map the abstract strategy into the solution strategy in the configuration space of the pursuers as illustrated in Figure 6.1c.

6.1 Preliminaries

6.1.1 Problem Description

We considers the problem of pursuit-evasion (PE) for worst-case adversarial targets with N pursuers, where the number of evaders is unknown and the evader is capable of moving arbitrary fast.

A map is defined as a free space, $\mathcal{W}$, in an n-dimensional Euclidean space, where n is typically 2 or 3. The position of the ith pursuer is specified by $p_i \in \mathcal{W}$, which can be applied to the sensor model O to get the sensor footprint, a set of points in $\mathcal{W}$ that can be observed from position $p_i, O(p_i) \subseteq \mathcal{W}$. An *evader space* $\mathcal{E}$ is defined as a set of points not being observed by any pursuers,

$$\mathcal{E} = \mathcal{W} \setminus \bigcup_i O(p_i).$$

Each point in the map can be *clear* or *contaminated*. The point is contaminated if an evader could be present in it, otherwise it is clear. The map is said to be clear when all points in $\mathcal{W}$ are clear. The point can be clear by being observed by any pursuer. However, the clear point $p \in \mathcal{E}$ can become contaminated again if there exists a *path* in an evader space from p to another contaminated point $q \in \mathcal{E}$, where the path from p to q is defined as a continuous function $\tau_{pq} : [0, T] \to \mathcal{E}$ such that $\tau_{pq}(0) = p, \tau_{pq}(T) = q$.

The process of clearing and contaminating the map as the pursuer move around is illustrated in Figure 6.2. Initially, evader can be in any unobserved points, so they are all

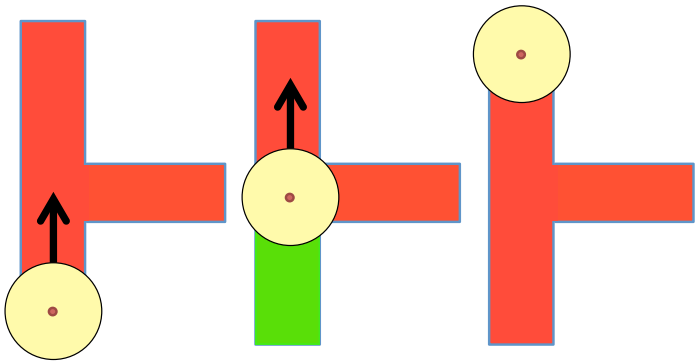


Figure 6.2: Illustration of the contaminated regions, shaded in red, and cleared regions, shaded in green, as the pursuer moves around in the free space. Initially, evader can be in any regions, so they are all contaminated. The pursuer then moves forward and clear the regions along the path. However, as the pursuer moves further and the cleared regions are connected to the contaminated ones, they become contaminated again.

contaminated. The pursuer then moves forward and clear the points along the path. However, as the pursuer moves further and the clear points are exposed to the contaminated ones, they become contaminated again. The objective of PE is then to compute trajectories for each pursuer for clearing all regions in the evader space, where a trajectory of ith pursuer is defined as a continuous function of time, $p_i(t)$ for $t \in [0, T]$.

Definition 4 (Strategy on map)**.** Let $\mathcal{W}$ be a free space representing a map. A strategy (π) is a collection of trajectories for all pursuers, $\pi_{\mathcal{W}} : [0, T] \rightarrow \mathcal{W}^N$, i.e. $\pi_{\mathcal{W}}(t) = [p_1(t), p_2(t), ..., p_N(t)]$.

Definition 5 (PE problem on a map)**.** Given the map $\mathcal{W}$ with N pursuers with sensor model O, determine a strategy π that clears the map $\mathcal{W}$.

Synthesizing a solution in continuous space can quickly become intractable, especially when multiple pursuers are required. As a result, we choose to reduce PE problem on a map to *PE problem on a graph* using Čech complex construction. We will first describe how to construct a graph and then formally introduce PE problem on a graph.

One of the main step in graph construction is choosing a set of representative points such that every point in $\mathcal{W}$ can be observed from at least one of the samples. Ideally, we also want to minimize the size of the representative set. However, this is essentially a

minimum set cover problem, one of the well-known NP-complete problems and hence we use the sampling-based method. First, we uniformly distribute the points to cover the convex hull of $\mathcal{W}$ based on the sensor model O. We then keep the sampling points that lie within $\mathcal{W}$ and set aside the rest. Next, we iterate through the points in $\mathcal{W}$ that are not within the sensor footprints of any chosen positions and choose the point in $\mathcal{W}$ closest to the nearest samples from the discarded points.

Assuming that O is convex and the pursuer can move holonomically, we then construct the Čech complex over the sampling points. For Čech complex, a 0-simplex exists for each sampling point; a 1-simplex exists between two 0-simplices whose their corresponding points have a non-empty intersected sensor footprints; and a 2-simplex exists for every 3-tuple of points whose sensor footprints have a non-empty intersection. To assert that we attain the hole-less coverage of the free space, we want the 2-simplices to cover all the points that are *sufficiently* faraway from the obstacle. The points are sufficiently faraway if they cannot be observed from the closest boundary of the obstacles.

Definition 6 (Graph representation)**.** $G = (V, E)$, where V is the set of 0-simplices and E is the set of 1-simplices.

Similar to the map, each vertex on G can be either clear or contaminated. The vertex v is clear when pursuer visits. However, v can be recontaminated if there exists a sequence of unobserved vertices to another point u, i.e. $(w_1, ..., w_k)$, where $w_1 = v, w_k = u, (w_i, w_{i+1}) \in E$ and all w_i's are unobserved. G is clear when all vertices are clear.

The trajectory on a graph is then defined as a sequence of vertices, $(v^0, v^1, ..., v^T), v^i \in V$ such that $(v^i, v^{i+1}) \in E$. For simplicity, we will discretize the movement of pursuer into time step of 1. In addition, we assume that multiple pursuers can occupied same vertex.

Definition 7 (Strategy on a graph)**.** Let $G = (V, E)$ be a graph. A strategy on graph (π_G or π) is a collection of trajectories on graph, $\pi : \{0, 1, ..., T\} \to V^N$, i.e. $\pi(t) = [v_1^t, v_2^t, ..., v_N^t]$.

Definition 8 (PE on graph)**.** Let $G = (V, E)$ be a graph. Determine a strategy π that clears G.

Furthermore, the strategies computed on the graph can be translated back into executable trajectories on the map. For any $(u, v) \in E$, a path between u, v is defined as a continuous function $\tau_{uv} : [0, 1] \to \mathcal{W}$ such that $\tau(0) = u$ and $\tau(1) = v$. Additionally, to prevent v from immediately contaminate u during execution, τ_{uv} must satisfy the following property:

$$\bigcap_{t\in[0,1]} O(\tau_{uv}(t)) = O(u) \cap O(v).$$

Since the every path from a point in $O(v)$ to a point in $O(u)$ while remaining inside $O(v) \cup O(u)$ must go through $O(v) \cap O(u)$, this property ensures that the intersection is always observed and hence no path in evader space from points in $O(v)$ to $O(u)$ (without going through other vertices).

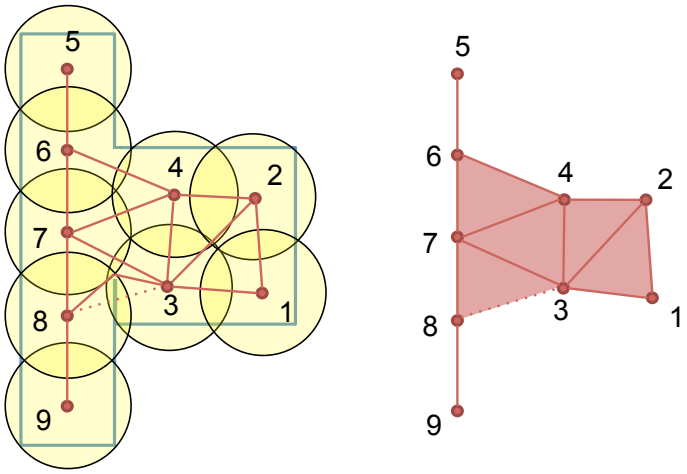


Figure 6.3: Example of graph representation in 2D environment with holomomic pursuer equipped with circular sensor footprint (left) and its Čech Complex (right). The path between vertices are denoted by solid lines, which are either a straight line or a pair of lines through an intermediate point in the presence of obstacles.

We will focus on the holonomic pursuer with sensor model of a ball with radius r.

We demonstrate the construction of graph representation using Čech complex on a map with circular sensor model in Figure 6.3. The path between any vertices will either be a straight line or a pair of straight lines through the point inside the intersection of their sensor footprints due to the presence of obstacles. In both cases, these paths satisfy the property for τ that prevents the immediate contamination during execution. For instance,

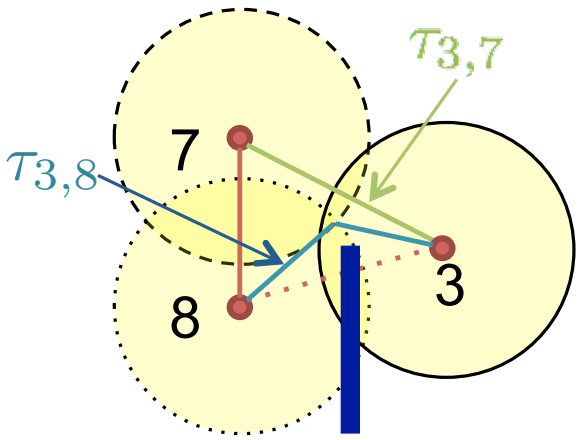


Figure 6.4: A valid path exist for any edges in G. For instance, any points along $\tau_{3,7}$ and $\tau_{3,8}$ remain observing $O(3) \cap O(7)$ and $O(3) \cap O(8)$ respectively.

$\tau_{3,7}$, a straight line from vertex 3 to 7, and $\tau_{3,8}$, a pair of straight lines from vertex 3 to 8, always cover their corresponding intersection as shown in Figure 6.4.

6.1.2 Pursuit-Evasion on Landmark Complex

This section derives the problem of pursuit evasion on the landmark complex, constructed in Chapter 5, as PE on graph. Since the landmark complex is identical to the Čech complex when the landmarks are sufficiently dense, the navigation graph described in Section 5.2.2 can be used as a graph representation of the landmark complex. Thus, all evaders in the environment will be captured by clearing the navigation graph of the landmark complex under the following assumptions.

1. The geometric realization of the landmark complex fully covers of the environment. This can be satisfied if the landmark is sufficiently dense and there exists landmark at every corner of the obstacles. This assumption ensures that clearing all of the free simplices of the landmark complex will capture all possible evaders in the environment.

2. The pursuer must maintain the visibility of all shared landmarks when transitioning between free simplices. This assumption ensures that the transition between free simplices won't trigger the re-contamination.

6.1.3 Solving PE as a Partially Observable Planning Problem

Since the positions of the evaders are unknown, we could not fully observe the state during planning. Hence, we introduce the notion of *belief state* which is a unique situation that may occur during execution.

The belief state, denoted by x, consists of the configuration/position of all pursuers, denoted by $\mathbf{p}$, and the possible positions of the evaders, which will be referred as the *contaminated regions* denoted by $\mathbf{c}$. The collection of all belief states is referred as the *belief space* (X) , while the *configuration space* (P) is spanned by the position of the pursuers. The span of contaminated regions will be referred as the *contamination space*, (C).
Thus we have that

$$x = (\mathbf{p}, \mathbf{c}) \in X, \quad \text{with } \mathbf{p} \in P, \mathbf{c} \in C.$$

On the graph representation G, the configuration space, P, is spanned by the pursuer positions, $\mathbf{p} = \{p^1, ..., p^N\}$, where $p^i \in V$. On the other hand, the contamination space, C, can be defined as a set of vertices that the evaders could be present in. Hence, it is a subset of a power of set of V, $\mathbf{c} \in C \subseteq \mathbb{P}(V)$.

The update step occurs when the pursuers take action, i.e. move along an edge in G. With the time discretization on graph, the action can simply be written as the next configuration of the pursuers, $\mathbf{p}'$, and hence the update function can be defined as

$$Update(x_t, \mathbf{p}'):$$
$$x_{t+1} = (\mathbf{p}', UpdateContaminate(\mathbf{c}_t, \mathbf{p}'))$$

$UpdateContaminate$ updates the contaminated vertices based on the current contamination status and next configuration of the pursuers by computing a set of reachable vertices on $G' = (V \setminus \mathbf{p}', E \setminus \mathbf{p}')$ beginning at $\mathbf{c}_t \setminus \mathbf{p}'$. In addition to all edges that contain occupied vertices, the edge subtraction may require removing some additional edges. The additional edge removal will be explained in section 6.2.1.

The solution strategy is then a sequence of actions in the belief space such that the

contaminated regions becomes an empty set, i.e. $\pi = \{(\mathbf{p}_0, \mathbf{c}_0), (\mathbf{p}_1, \mathbf{c}_1), ..., (\mathbf{p}_T, \emptyset)\}$, where $(\mathbf{p}_0, \mathbf{c}_0)$ is the initial state. Solving this as a partially observable planning problem requires search in an intractably large space of belief states, which is exponential in the number of joint pursuer-evader configurations. To address this challenge, we will use a novel abstraction technique that is described in the next section.

6.2 Abstraction Framework

6.2.1 Abstraction State Space

To cope with the exponential growth of the belief space, we propose the novel method to partition the configuration space into *abstraction state space*, denoted by $\mathcal{S}$, by utilizing the topological invariants of the evader space.

Although the contamination space might appear to be exponential of $|V|$, not all combinations of the contaminated regions are reachable. Since the evader can move arbitrary fast, any adjacent regions in the evader space will both be either contaminated or cleared.

Utilizing this fact, we define the *connected component (CC)* function which returns the sets of adjacent vertices in the evader space of G based on the assignment of pursuers, $\mathbf{p}$, denoted by $CC(\mathbf{p}, G)$ or simply $CC(\mathbf{p})$ when G is obvious. The connected component function can be computed by projecting the sensor footprints of the pursuers onto the free space, and then reconstructing the graph representation of the evader space, $\mathcal{E}$, as illustrated in Figure 6.5.

The connected component function can also be computed by subtracting the vertices (and their associated edges) occupied by the pursuers from G. Nevertheless, in the present of obstacles, the evader space might remain connected in G while become disconnected in $\mathcal{W}$ as illustrated in Figure 6.6. These exceptions lead to additional edges removal from G. For 2D environment, there are only two possible scenarios. The first scenario occurs when the intersection of two sensor footprints is completely contained inside the sensor footprint of another vertex (Figure 6.6a). The other scenario occurs when there is a 4-way intersection of sensor footprints, resulting in edge intersection in G (Figure 6.6b).

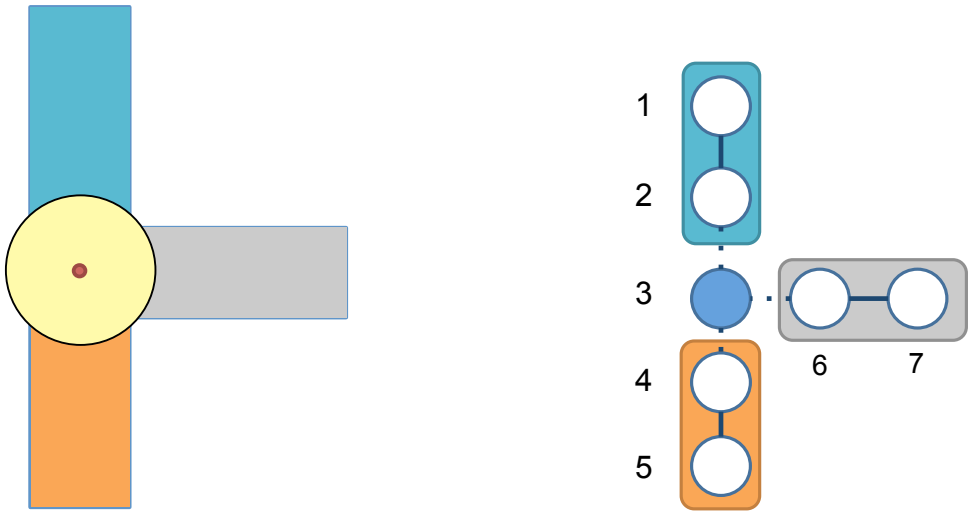
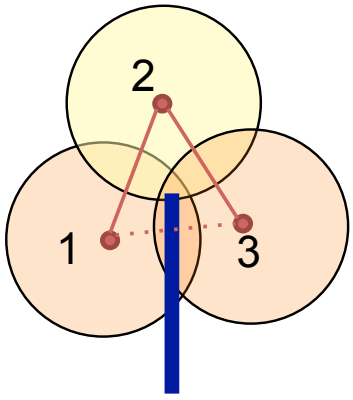
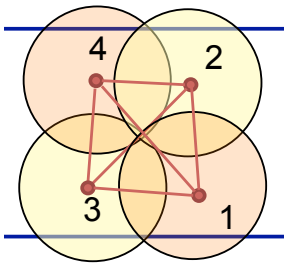


Figure 6.5: Illustration of an connected component function on G with pursuers at $\mathbf{p} = \langle 3, 3 \rangle$. (a) Sensor footprint is projected onto $\mathcal{W}$ which then separate the evader space into multiple connected components (b) The graph is reconstructed on the evader space where the position of pursuers are depicted by blue-shaded vertices, while the evader space are grouped by shaded boxes for each connected component. $CC(\langle 3, 3 \rangle, G) = \{\{1, 2\}, \{4, 5\}, \{6, 7\}\}$



(a) With pursuer at $\langle 2 \rangle$, 1 and 3 remain connected with removal of 2 in G but actually get disconnected in $\mathcal{W}$.



(b) Similarly, with pursuers at $\langle 2, 3 \rangle$, 1 and 4 remain connected with removal of 2 and 3 in G but actually get disconnected in $\mathcal{W}$.

Figure 6.6: Additional edges removal is required when computing connected component on G on these cases.

During graph construction, we can keep track of the intersection between sensor footprints to handle the first scenario, while edge intersection can be easily computed. Hence, the CC function can be computed on G for 2D environment. For higher dimension, the CC function can be computed on G only if all exceptions are tractable. Otherwise, completeness is not guarantee.

Using the results of the CC function, we want to partition the configuration space into *abstract states* in a way that preserves the topology of the evader space, which is equivalent to the contamination status of each connected component remains unchanged. Using abstract state $\mathcal{S}_1$ in Figure 6.7 as an example, $\langle 3,3\rangle \sim \langle 3,4\rangle$ and there exists a one-to-one mapping between $CC(\langle 3,3\rangle)$ and $CC(\langle 3,4\rangle)$ which preserves the contamination status of each connected component. On the other hand, the edge between two abstract states denotes the transition that does not preserve the topology of an evader space, which could then lead to the changes in contamination status of the evader space. For instance, the edge between $\mathcal{S}_1$ and $\mathcal{S}_2$ represents the transition between $\langle 3,4\rangle$ and $\langle 4,4\rangle$ which is resulted in two connected components of $CC(\langle 3,4\rangle)$ merging and could potentially changes their contamination statuses. We first introduce a relation between two adjacent configurations and then formally define an equivalence relation for partitioning the configuration space as follow.

Let $\mathbf{p} \to \mathbf{q}$ denotes two adjacent configurations $\mathbf{p}, \mathbf{q} \in P$ s.t. $(p^k, q^k) \in E, \forall k \in \{1, ..., N\}$. Given two adjacent configurations we can define a relation, which we call *transition relation*, as follows.

Definition 9 (Transition Relation)**.** Let $\mathbf{p}, \mathbf{q} \in P$ s.t. $\mathbf{p} \to \mathbf{q}$. The transition relation $\rho_{\mathbf{p},\mathbf{q}}$ between connected components of $\mathbf{p}$ and $\mathbf{q}$ is defined as

$$(cc_i^{\mathbf{p}}, cc_j^{\mathbf{q}}) \in \rho_{\mathbf{p},\mathbf{q}} \quad \Leftrightarrow \quad cc_i^{\mathbf{p}} \cap cc_j^{\mathbf{q}} \neq \emptyset\,,$$

where $cc_i^{\mathbf{p}} \in CC(\mathbf{p}), cc_j^{\mathbf{q}} \in CC(\mathbf{q})$.

Given the previous definition we can now formally introduce an equivalence relation

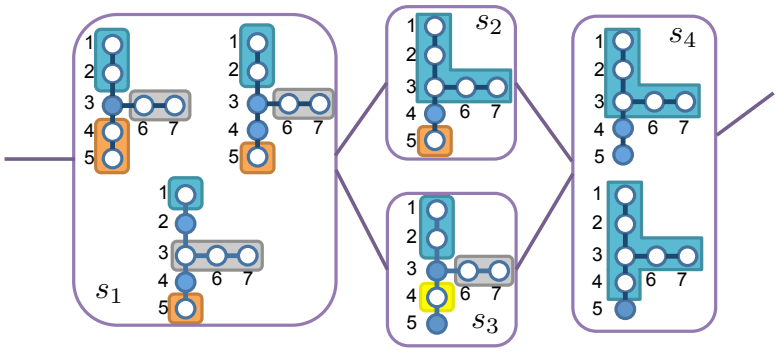


Figure 6.7: Simple configuration space of two pursuers is partitioned into abstraction state space using equivalence relation.

between states, which we will use to define a state abstraction.

Definition 10 (Equivalence Relation)**.** For all $\mathbf{p}, \mathbf{q} \in P$ we say that $\mathbf{p}$ is equivalent to $\mathbf{q}$, or $\mathbf{p} \sim \mathbf{q}$, if and only if there exists a finite sequence $\{\mathbf{z}^i\}_0^T \in P^{T+1}$ with ρ such that

1. $\mathbf{z}^0 = \mathbf{p}$, and $\mathbf{z}^T = \mathbf{q}$;
2. $\mathbf{z}^i \rightarrow \mathbf{z}^{i+1}$, with $i \in \{0, ..., T-1\}$;
3. $\rho_{\mathbf{z}^i, \mathbf{z}^{i+1}}$ is a bijection.

Hence, the abstraction state space can be defined as $\mathcal{S} = P/\sim$, where each abstraction state $s_i \in \mathcal{S}$ is a collection of equivalent configurations.

As a result, the contamination status of each connected component could only be changed upon transition between abstraction states. Hence, we can synthesize the solution strategy on the abstraction state space instead of synthesize the strategy directly in the configuration space.

In next section, we will describe the algorithm to incrementally construct the abstraction state space $\mathcal{S}$, the function mapping P to $\mathcal{S}$, denoted by (γ), and the adjacency matrix of abstraction states, denoted by $\mathcal{M}$.

6.2.2 Partition Algorithm

Algorithm 9 outlines an incrementally construction of the abstraction state space and other components required for synthesizing a strategy. The concept is to perform a forward search

Algorithm 9 Partition algorithm

```
1: S ← ∅, M ← ∅
2: Q.Insert(p_I) for some arbitrary p_I
3: while Q ≠ ∅ do
4:     p ← Q.GetFirst(),    mark p as visited
5:     γ(p) ← null,    Adjacent_S ← ∅
6:     for p' ∈ Adjacent(p) do
7:         if p' is visited then
8:             if CC(p) ∼ CC(p') then
9:                 if γ(p) is null then
10:                    γ(p) ← γ(p')
11:                else
12:                    Resolve conflict if needed
13:                end if
14:            else
15:                Adjacent_S.Insert(γ(p'))
16:            end if
17:        else if p' is unvisited then
18:            Q.Insert(p'),    mark p' as alive
19:        end if
20:    end for
21:    if γ(p) is null then
22:        S.Insert(Abstract(p)),    γ(p) ← Abstract(p)
23:    end if
24:    for a ∈ Adjacent_S do
25:        Update M(γ(p), a)
26:    end for
27: end while
```

over P beginning at an arbitrary state $\mathbf{p}_I$ and partition them into the abstraction state, $a \in \mathcal{S}$, based on the output of $CC(\mathbf{p})$.

Following standard forward search algorithm (line 2-6, 17-18), each state in P begins as *unvisited* and will be marked *alive* or *visited* upon inserting to or removing from Q respectively. The set of alive states is stored the in list Q and the search is completed when the list Q is empty. The function $Adjacent(\cdot)$ in line 6 returns the set of adjacent states by moving the pursuers along graph G. In this step, we will restrict the adjacent states to one pursuer movement only.

The partitioning occurs between line 7-15 and 21-25, where we compare the connected components of the current state to the visited adjacent states and either assign the current

state to the new abstraction state or append it to the existing one.

In general, comparing the connected component between two arbitrary configurations is nontrivial. Nevertheless, comparing those of the adjacent configurations is much simpler. In line 8, we compute the transition relation, $\rho_{\mathbf{p},\mathbf{p}'}$, as defined in Definition **??** and check whether it is bijective. This transition relation is also used for updating the contaminated regions when transiting between abstraction states.

If the graph consists of *cycles*, a conflict might occur when two similar configurations get assigned into two different abstraction states. This will be resolved in line 12 where two abstraction states will be combined.

Furthermore, the adjacency matrix, $\mathcal{M}$, is updated based on the connectivity of the corresponding configuration states. In line 15, $Adjacent_A$ keeps track of the adjacent abstraction states which will then update $\mathcal{M}$ in line 25. The adjacency matrix also store the transition relation(s) between two abstraction states and the corresponding configurations. The transition relation might not be unique if G consists of cycles.

As a result, the computational complexity of the partition algorithm is approximately $O(dN|V|^{N+1})$, which consists of $O(dN|V|^N)$ from the forward search algorithm over the configuration space of size $|V|^N$ where each configuration has $O(dN)$ adjacent states, d denotes the average degrees of the vertices, and $O(|V|)$ from comparison of evader space in line 8. Although the number might seems large, it is much smaller comparing to searching over original belief space because the partition algorithm is only exponential with respect to the number of pursuers, which is commonly known as *curse of dimensionality* in multi-robot motion planning problem. In the next section, we will explain how to use the output of the partition algorithm to synthesize the solution strategy.

6.3 Hierarchical algorithm

To synthesize the strategy using the abstraction framework, we first search for the strategy in the abstraction state space and then refine the strategy into the configuration space. If the number of pursuers, N is given, planning in abstraction framework would either return the

strategy or indicate that no solution exists for the given N. The search for strategy in the abstraction state space can be done using existing techniques for graph-based searching such as Dijkstra's algorithm. We will describe the abstraction believe space for planning in the abstraction state space in section 6.3.1, and then discuss the refinement step in section 6.3.2.

6.3.1 Planning in the abstraction state space

The abstraction believe state for the abstraction state space, denoted by $x_{\mathcal{S}}$, becomes a pair of the abstraction state (s) and the list of contaminated regions (L), where each region represents a set of adjacent vertices of the evader space.

$$x_{\mathcal{S}} = (s, L), \quad L = \{s^j\}.$$

The update step during planning will keep track of the contaminated regions using the information stored in the adjacency matrix $\mathcal{M}$. Since the transition relation ρ may not be unique, the update step with input s_k has to be called for each ρ stored in $\mathcal{M}(s_t, s_k)$.

$$Update(x_{\mathcal{S},t}, s_k, \rho): \qquad L_{t+1} = \{s_k^j \mid \exists s_t^i \in L_t, (s_t^i, s_k^j) \in \rho\} \tag{6.1}$$

$$x_{\mathcal{S},t+1} = (s_k, L_{t+1}) \tag{6.2}$$

The solution strategy is a sequence of abstract belief states such that the list of contaminated regions eventually becomes empty, denoted by $\pi_{\mathcal{S}} = (a_{i_1}, \{a_{i_1}^j\}), (a_{i_2}, \{a_{i_2}^j\}), ..., (a_{i_k}, \emptyset)$. We will then describe how to map the solution strategy into the strategy on a graph with the refinement process on the following section.

6.3.2 Refinement

The information stored in $\mathcal{M}$ provides the boundary configurations representing the transition between abstraction states. Nevertheless, given the sequence of abstraction states, the entering configuration might not be adjacent to the leaving one. For instance, in Figure 6.8, the incoming and outgoing configuration of abstraction state s_j are not adjacent. Thus, the refinement step is required to find the trajectory from the incoming to outgoing

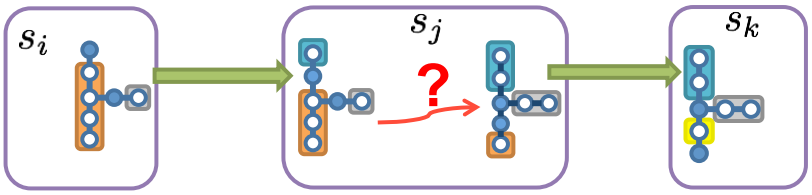


Figure 6.8: The transition between abstraction states represents the action between boundary configurations, however, the incoming configuration need not be adjacent to the outgoing one. Hence, the refinement step is necessary to find the trajectories between them.

configuration such that all intermediate configurations are the member of s_j.

Using mapping function γ, one method for refinement is to perform the forward search inside each abstraction state to find the trajectory from the incoming to outgoing configurations. Nevertheless, this method might be inefficient since we already expand the full configuration space while executing partition algorithm.

An alternative method is to store information of the spanning trees of each abstraction state during the partition algorithm and then search for the trajectory on the spanning trees during refinement. The downside of this method is that the result is usually suboptimal compared with one from forward search method.

Given the strategy $\pi_{\mathcal{A}} = \{s_0, s_1, s_2, ..., s_k\}$ where $\gamma(\mathbf{p}_I) = s_0$, the refinement then first search for a trajectory from $\mathbf{p}_I$ to the boundary configuration connecting to s_1 while remaining within s_0. Then, we continue refine the solution inside s_1 from the incoming configuration from s_0 to the outgoing configuration connecting to s_2 while remaining within s_1. The same process continue until we reach the final abstraction state s_k.

6.3.3 Minimizing the number of pursuers

The full algorithm for synthesize the solution strategy is given in algorithm 10 when the number of pursuers, N, required is unknown. Note that incrementing N by one at each iteration is more efficient that doing binary search because the computational complexity is exponential with respect to N.

Algorithm 10 Hierarchical Strategy Synthesis

1: Construct G of free space $\mathcal{W}$ with sensor model $O(\cdot)$
2: $N \leftarrow 1, \pi_{\mathcal{S}} \leftarrow [\,]$
3: **while** $\pi_{\mathcal{S}}$ is empty **do**
4: Partition G^N into abstraction state space $\mathcal{S}$
5: $\pi_{\mathcal{S}} \leftarrow Planning(\mathcal{S}, \mathbf{p}_I)$
6: Increment N
7: **end while**
8: $\pi \leftarrow Refine(\pi_{\mathcal{S}})$

6.4 Results

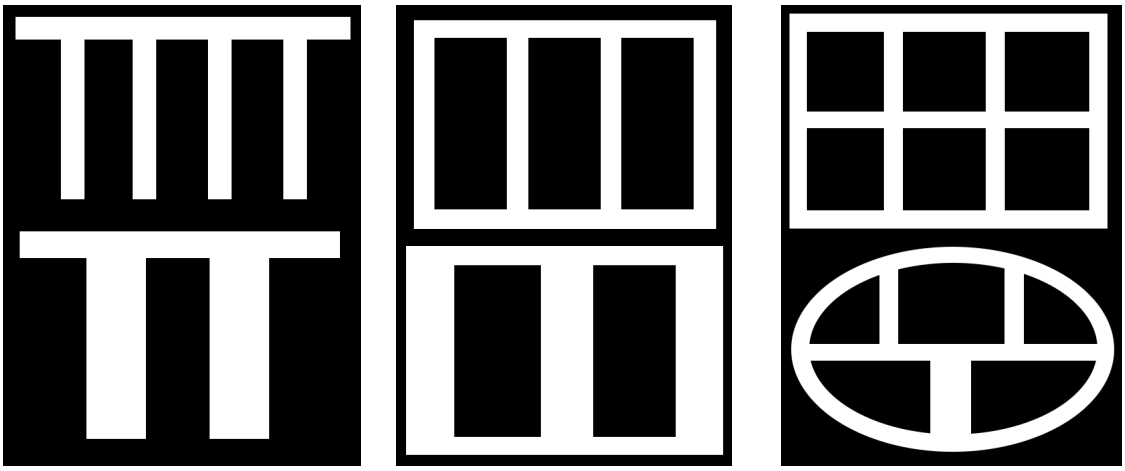
The construction of graph representation is implemented in MATLAB, whereas the remaining components are implemented in C++. We validate the proposed method in simulations with environments of varying topologies and using different number of pursuers as discussed in section 6.4.1. Then, we compare our results with the graph-based searching over the full belief space on simple environments in section 6.4.2.

6.4.1 Simulation Results

We evaluated the performance in simulation on environments with three different topologies as show in Figure 6.9. Each pursuer has a disc sensor footprint with a radius of one meter. Due to various structures of the environments, we represent their dimensions with the number of vertices in the graph representation. Figure 6.9c illustrates the graph representation of the testing environments and the sensor footprint of the pursuer. The number of pursuers required in each environment is computed by iterating from $N = 1$.

Tree Structure

We evaluated on the tree structures with varying number k and width w of branches (vertical corridors). The graph representations contain $8 - 48$ vertices. It requires 2 pursuers to clear the map for $w = 1$ and 3 pursuers to clear the map for $w = 2$. The execution times of each component are illustrated in Figure 6.10.



(a) Tree structure **(b)** Ladder Structure **(c)** Random Loops Structure

Figure 6.9: Testing Environments

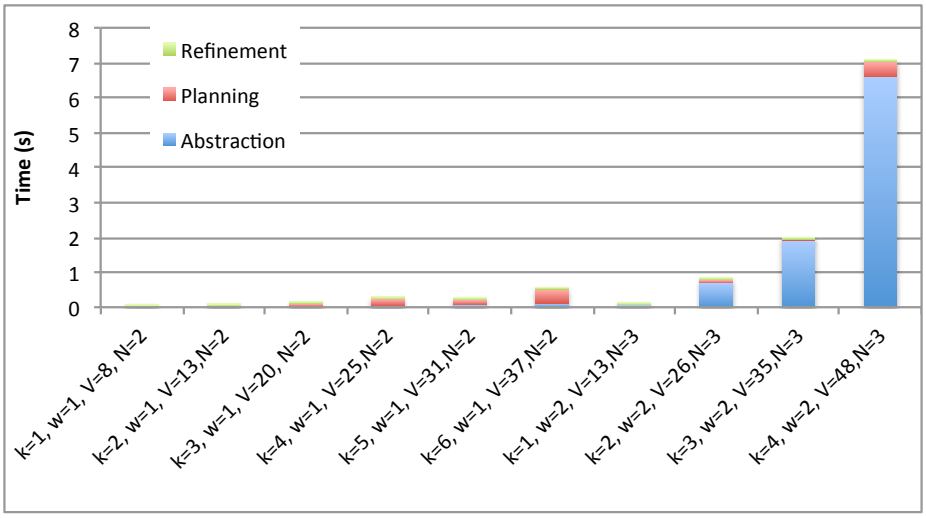


Figure 6.10: The execution time of the proposed method on tree structure with k branches of width w using N pursuers on graph with V vertices.

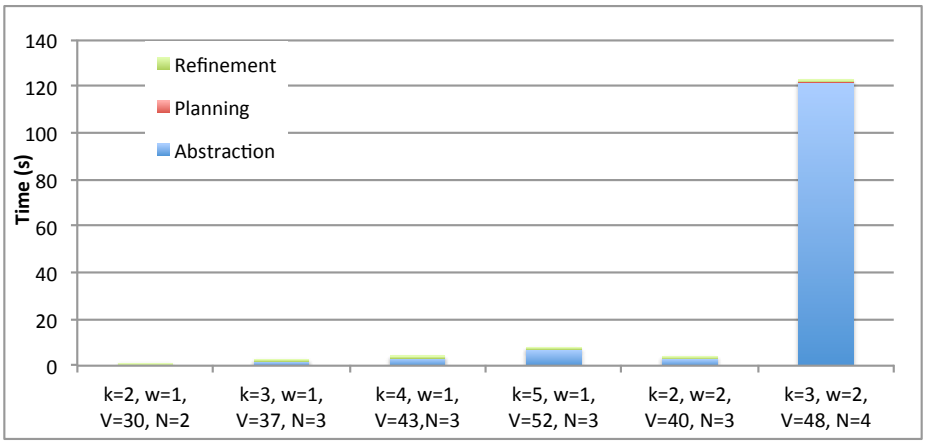


Figure 6.11: The execution time of proposed method on ladder structure with k steps of width w using N pursuers on graph with V vertices.

Ladder Structure

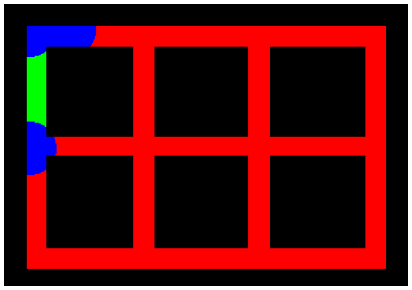
We evaluated on the ladder structures with varying number k and width, w, of steps (vertical corridors). The graph representations contain $30 - 52$ vertices. For ladder with single loop $k = 2$, it requires 2 pursuers to clear the map with $w = 1$ and 3 pursuers to clear the map with $w = 2$. If the ladder with multiple loops $k > 2$, it requires 3 pursuers to clear the map with $w = 1$ and 4 pursuers to clear the map with $w = 2$. The execution times of each component are illustrated in Figure 6.11.

Random Loops Structure

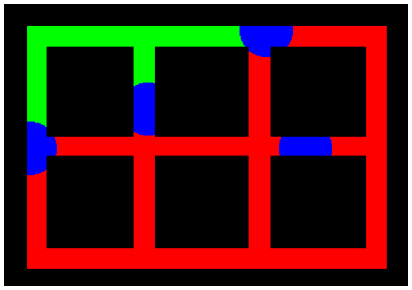
We evaluated the proposed methods on two maps shown in 6.9c using 4 pursuers for the top one, which consists of 46 vertices, and 5 pursuers for the bottom map, which consists of 53 vertices. The total execution times are 105.59 and 4076.32 seconds, in which abstraction framework are accounted for 103.25 and 4074.44 seconds respectively. The intermediate steps during the clearing process are shown in Figure 6.12 and Figure 6.12.

As explained in section 6.3.2, one the main disadvantages of refinement using spanning tree is the lengthy strategy as indicated by a high number of iteration in both examples. This strategy can be further improved; however, this is out the scope of this thesis.

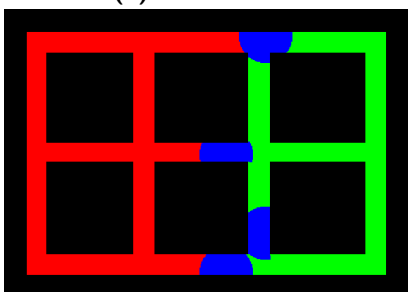
The simulation results show that the abstraction framework is responsible for the major-



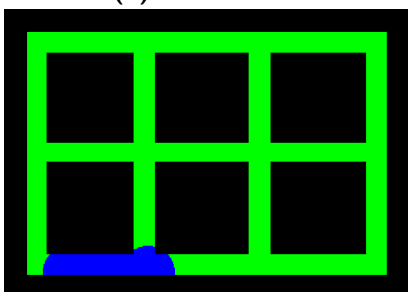
(a) Iteration: 10



(b) Iteration: 30

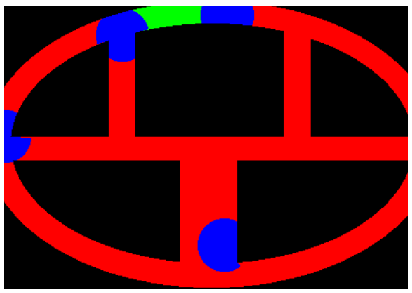


(c) Iteration: 100

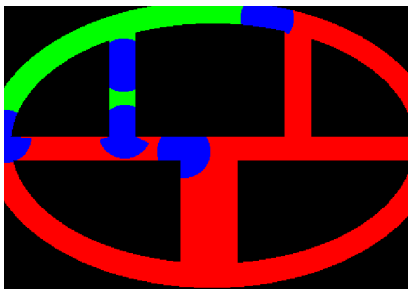


(d) Final Iteration (147)

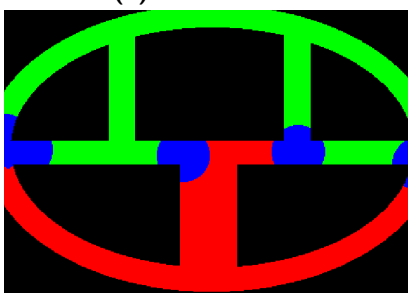
Figure 6.12: Snapshots of clearing process on 3×4 grid map with all narrow passages using 4 pursuers.



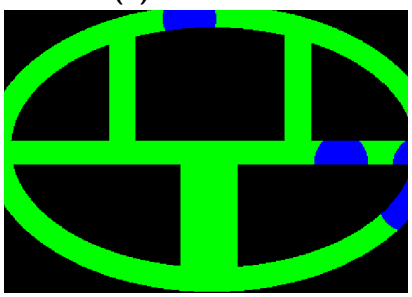
(a) Iteration: 30



(b) Iteration: 70



(c) Iteration: 250



(d) Final Iteration (309)

Figure 6.13: Snapshots of clearing process on curved hallway with vary passages size using 5 pursuers.

Map	Our framework	Baseline planner
Tree with 1 branch, V= 8	0.014	0.04
Tree with 2 branches, V=13	0.034	49.95
Ladder with 2 steps, V=14	0.018	1082.77

Table 6.1: Comparison of execution time (s) between our framework and baseline planner.

ity of computation time as N increases, which conforms with our analysis on computational complexity of the partition algorithm. Nevertheless, the abstraction framework only needs to be executed once for each given map with the same number of pursuers. It is invariant of the initial position and thus we can quickly synthesize the strategy again for different initial position. This can be useful if an error occurs while executing the solution strategy and the pursuers are deviated from the planned strategy.

6.4.2 Comparison

We compare the results of our proposed algorithm with a baseline (brute force) planner which searches for a strategy on the full belief space, described in section 6.1.3. We used maps with tree and ladder structures. The baseline planner can only solve the maps containing up to 2 branches (13 vertices) for tree structure and the maps containing one loop (14 vertices).

The execution time of the baseline planner quickly grow exponential as V increases. Additionally, the baseline planner suffers greatly from the topological invariant of the loop structure (such as a ladder) because each configuration is mapped to a new believe state. On the other hand, our framework reduces the configuration space of 2 pursuers with one loop into 2 abstraction states; one for two pursuers being adjacent and other when they are separated.

6.4.3 Discussion

In this chapter, we proposed an abstraction framework to solve a worst-case adversarial pursuit-evasion problem where multiple pursuers with limited-range sensor coverage are used to detect all possible mobile evaders. This method involves constructing the graph representation of an environment using the sensor model equipped on the pursuers, par-

titioning the configuration space of the pursuers over graph into an abstract state space, searching for a strategy in the abstract state space, and finally refine the strategy into the configuration space. We validate our proposed method by simulating environments with different topologies and compare the result with a brute force searching over the full belief space. Since the full belief space grows exponentially with N and the number of vertices in V, the brute force search can only solve PE problems on small maps ($|V| < 20$) for $N = 2$. In contrast, our approach reduces the complexity into exponential of N with base $|V|$ and can solve the PE problems with a few hundred vertices for $N = 2$ and up to 50 vertices for $N = 5$.

Although the abstraction framework requires inspecting the full configuration space of N pursuers, this step needs to be done only once for a given map with the same number of pursuers. The output can be reused for different initial configurations. Furthermore, we observe that many maps with the similar structure yield the same abstraction. This opens up an interesting problem of how can we apply the result of one abstraction framework to the new maps with similar structure without recomputing it. Additionally, we are interested in converting an abstraction framework into an on-line algorithms so that we can concurrently synthesize for the solution strategy, which may avoid exploring the entire configuration space.

Chapter 7

Conclusion and Future Work

7.1 Contribution

This dissertation addresses a series of problems related to coverage and mapping of an unknown environment by a swarm of resource-constrained robots. In particular, we consider the instantiations of problems related to sensor coverage, mapping, exploration, and pursuit-evasion. This dissertation makes various contributions in each of these areas focusing on the design and development of algorithms that require a minimal amount of sensing and computational capability.

Using a swarm of robots only capable of measuring bearing angles to other robots within their sensing region, we present algorithms for the deployment of mobile robots to attain full sensor coverage, followed by the computation of a topological map of an unknown environment without knowledge of any global or metric information. We then adopt a frontier-based strategy on a landmark complex to develop an exploration strategy that does not rely on metric information for autonomous exploration that will ultimately create a topological map of a feature-rich environment with any number of robots. Lastly, we propose a pursuit-evasion algorithm to detect all possible evaders and clear a known environment, given as either a metric map or landmark complex.

7.2 Future Work

This dissertation only begins to scratch the surface of potential applications that can be provided by a swarm of resource-constrained robots. There are a number of interesting directions for future work.

One immediate question is how to implement them in real-world applications. The coverage and mapping algorithms, presented in Chapter 3 and 4, have been implemented in a heterogeneous team of real and virtual robots. Due to the limited number of physical robots and sensors, we only ran the experiment in a controlled space where sensing was greatly simplified through the use of a motion capture system and simulation. From our small experiment, which only deployed single robot per iteration, we noticed that the issue of total execution time would have to be addressed before application to a large scale system. Hence, the proposed method should be extended to handle multiple deployments per iteration as well as asynchronous execution between each deployment.

For the landmark-based exploration, presented in Chapter 5, one of the main challenges was the detection of landmarks, especially in terms of consistency and uniqueness. There are many potential representations of landmarks from visual inputs such as image features (SIFT [65] or ORB [85]) or object classes with Deep Learning techniques [44, 79]. Generally, the environment is rich with features/objects that can be used as landmarks. Hence, the key question here is how to find a set of unique landmarks that could correctly capture all the geometric properties of an environment. A potential solution is to use a persistent homology to filter out the common landmarks that occur frequently. Nevertheless, the problem of determining the uniqueness of features (such as landmarks) is a challenging task, even in the area of computer vision [96]. An object detector yields a classification confidence which seems useful, while it does not differentiate between two objects of the same class.

Another interesting direction would be to extend the landmark-based exploration algorithm to higher dimension. The landmark complex is already well-defined for 3D environment, while the stochastic different equation (SDE) model could be utilized to drive the robots in frontier-based exploration of 3D environment [51, 88]. Additionally, it would

be worth testing the relative bearing-only controllers proposed in [6, 9], despite the lack of evidence for global convergence, to relax the assumption of global compass direction in Chapter 5.

The next question is related to the use of topological maps for other tasks. In Chapter 6, we discussed the assumptions of the landmark complex, in which the proposed framework could solve the multi-pursuers and multi-evaders planning problem on the landmark complex. Those assumptions could potentially be further generalized or relaxed to solve a PE problem on a broader range of landmark complexes. Additionally, the topological map could also be utilized in other multi-robot applications where the objective is to find a feasible solution instead of the optimal one, such as patrolling or a multi-vehicle routing task.

Regarding the constraint on resources, our focus has been on sensing capabilities. However, there are other avenues for future research such as limited communication range or the deployment of the proposed strategies in a distributed system. Although our algorithms have been designed to require as minimal exchange of information as possible, we assume that information sharing has neither delay nor range limitation. The limited communication range would not only make information unavailable to/from disconnected robots but also introduce delays when propagating information across the swarm, which is likely to cause some of the proposed algorithms to fail. The issue with delays could be addressed by adding timestamps into the shared information, while the issue with disconnected swarms require each robot to model and predict the states of other robots to rendezvous and exchange information. Additionally, as the number of the robots in the swarm increases, the system should move toward being fully distributed by designing policies allowing each individual robot to compute with local information only without propagating information across entire swarms.

Finally, the coordination strategies in this dissertation are all designed based on greedy methods that, although a sensible choice in many combinatorial problems, can yield sub-optimal solutions, as it occurs in vehicle routing problems [10, 12, 57]. Thus, it would be an interesting direction to apply the techniques from the vehicle routing literature such as

auction-based multi-robot routing algorithms or divide and conquer policies [12, 57] to the coordination strategies of the robot swarms.

Bibliography

[1] Ercan U Acar, Howie Choset, Alfred A Rizzi, Prasad N Atkar, and Douglas Hull. Morse decompositions for coverage tasks. *The International Journal of Robotics Research*, 21 (4):331–344, 2002. doi: 10.1177/027836402320556359.

[2] Henry Adams and Gunnar Carlsson. Evasion paths in mobile sensor networks. *The International Journal of Robotics Research*, 34(1):90–104, 2015. URL `http://ijr.sagepub.com/content/34/1/90`.

[3] Francesco Amigoni and Vincenzo Caglioti. An information-based exploration strategy for environment mapping with mobile robots. *Robotics and Autonomous Systems*, 58 (5):684–699, 2010.

[4] Adrien Angeli, David Filliat, Stéphane Doncieux, and Jean-Arcady Meyer. Fast and incremental method for loop-closure detection using bags of visual words. *IEEE Transactions on Robotics*, 24(5):1027–1037, 2008.

[5] Antonis A Argyros, Kostas E Bekris, and Stelios C Orphanoudakis. Robot homing based on corner tracking in a sequence of panoramic images. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 2, pages II–II. IEEE, 2001.

[6] Antonis A Argyros, Kostas E Bekris, Stelios C Orphanoudakis, and Lydia E Kavraki. Robot homing by exploiting panoramic vision. *Autonomous Robots*, 19(1):7–25, 2005.

[7] David Avis and Binay K Bhattacharya. Algorithms for computing d-dimensional Voronoi diagrams and their duals. *Advances in Computing Research*, 1:159–180, 1983.

[8] Maxim Batalin and Gaurav S. Sukhatme. The design and analysis of an efficient local algorithm for coverage and exploration based on sensor network deployment. *IEEE Transactions on Robotics*, 23(4):661–675, Aug 2007.

[9] Kostas E Bekris, Antonis A Argyros, and Lydia E Kavraki. Exploiting panoramic vision for bearing-only robot homing. In *Imaging beyond the pinhole camera*, pages 229–251. Springer, 2006.

[10] Marc Berhault, He Huang, Pinar Keskinocak, Sven Koenig, Wedad Elmaghraby, Paul Griffin, and Anton Kleywegt. Robot exploration with combinatorial auctions. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, pages 1957–1962. IEEE, 2003.

[11] Subhrajit Bhattacharya, Robert Ghrist, and Vijay Kumar. Multi-robot coverage and exploration on riemannian manifolds with boundary. *International Journal of Robotics Research*, 33(1):113–137, January 2014. DOI: 10.1177/0278364913507324.

[12] Francesco Bullo, Emilio Frazzoli, Marco Pavone, Ketan Savla, and Stephen L Smith. Dynamic vehicle routing for robotic systems. *Proceedings of the IEEE*, 99(9):1482–1504, 2011.

[13] Wolfram Burgard, Mark Moors, Cyrill Stachniss, and Frank E Schneider. Coordinated multi-robot exploration. *IEEE Transactions on robotics*, 21(3):376–386, 2005.

[14] Jose A Castellanos, JMM Montiel, José Neira, and Juan D Tardós. The SPmap: A probabilistic framework for simultaneous localization and map building. *IEEE Transactions on Robotics and Automation*, 15(5):948–952, 1999.

[15] Hai-Chau. Chang and Lih-Chung Wang. A Simple Proof of Thue's Theorem on Circle Packing. *ArXiv e-prints*, September 2010.

[16] Howie Choset. *Sensor based motion planning: The hierarchical generalized Voronoi graph*. PhD thesis, California Institute of Technology, Department of Mechanical Engineering, 1996.

[17] Howie Choset and Keiji Nagatani. Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization. *IEEE Transactions on robotics and automation*, 17(2):125–137, 2001.

[18] Howie Choset, Sean Walker, Kunnayut Eiamsa-Ard, and Joel Burdick. Sensor-based exploration: Incremental construction of the hierarchical Generalized Voronoi Graph. *The International Journal of Robotics Research*, 19(2):126–148, 2000. doi: 10.1177/02783640022066789.

[19] Louis Coetzee and Johan Eksteen. The internet of things-promise for the future? an introduction. In *IST-Africa Conference Proceedings, 2011*, pages 1–9. IEEE, 2011.

[20] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, 2nd edition, 2001.

[21] Nikolaus Correll and Heiko Hamann. Probabilistic modeling of swarming systems. In *Springer Handbook of Computational Intelligence*, pages 1423–1432. Springer, 2015.

[22] Jorge Cortes, Sonia Martinez, Timur Karatas, and Francesco Bullo. Coverage control for mobile sensing networks. *IEEE Trans. Robot. Autom.*, 20(2):243–255, April 2004.

[23] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1052–1067, 2007.

[24] Vin De Silva and Robert Ghrist. Coordinate-free coverage in sensor networks with controlled boundaries via homology. *The International Journal of Robotics Research*, 25(12):1205–1222, 2006. doi: 10.1177/0278364906072252.

[25] Vin De Silva, Robert Ghrist, and Abubakr Muhammad. Blind swarms for coverage in 2-d. In *Robotics: Science and Systems*, pages 335–342, 2005.

[26] Jason Derenick, Vijay Kumar, and Ali Jadbabaie. Towards simplicial coverage repair for mobile robot teams. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 5472–5477. IEEE, 2010.

[27] Clifford H Dowker. Homology groups of relations. *Annals of mathematics*, pages 84–95, 1952.

[28] Gregory Dudek, Michael Jenkin, Evangelos Milios, and David Wilkes. Robotic exploration as graph construction. *Robotics and Automation, IEEE Transactions on*, 7(6): 859–865, Dec 1991.

[29] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[30] Karthik Elamvazhuthi and Spring Berman. Optimal control of stochastic coverage strategies for robotic swarms. In *Proceedings of the IEEE International Conference on Robotics and Automation,*, pages 1822–1829. IEEE, 2015.

[31] Karthik Elamvazhuthi, Chase Adams, and Spring Berman. Coverage and field estimation on bounded domains by diffusive swarms. In *of the IEEE Conference on Decision and Control*, pages 2867–2874. IEEE, 2016.

[32] Brendan Englot and Franz S Hover. Three-dimensional coverage planning for an underwater inspection robot. *The International Journal of Robotics Research*, 32(9-10):1048–1073, 2013.

[33] Gerald Weinberg et al. Hypecycle. In *AYE Conference*, September 5 2003.

[34] F. Ferrari, Enrico Grosso, Giulio Sandini, and M. Magrassi. A stereo vision system for real time obstacle avoidance in unknown environment. In *Proceedings of IEEE International Workshop on Intelligent Robots and Systems*, pages 703–708. IEEE, 1990.

[35] Steven Fortune. Voronoi diagrams and delaunay triangulations. In *Computing in Euclidean geometry*, pages 225–265. World Scientific, 1995.

[36] Dieter Fox, Jonathan Ko, Kurt Konolige, Benson Limketkai, Dirk Schulz, and Benjamin Stewart. Distributed multirobot exploration and mapping. *Proceedings of the IEEE*, 94(7):1325–1339, 2006.

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

[38] Santiago Garrido, Luis Moreno, Mohamed Abderrahim, and Fernando Martin. Path planning for mobile robot navigation using Voronoi diagram and fast marching. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 2376–2381, Oct 2006. doi: 10.1109/IROS.2006.282649.

[39] Brian Gerkey, Richard T Vaughan, and Andrew Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics*, volume 1, pages 317–323, 2003.

[40] Brian P Gerkey, Sebastian Thrun, and Geoff Gordon. Visibility-based pursuit-evasion with limited field of view. *The International Journal of Robotics Research*, 25(4): 299–315, 2006. URL `http://dl.acm.org/citation.cfm?id=1124588`.

[41] Robert Ghrist. *Elementary Applied Topology*. URL `http://www.math.upenn.edu/~ghrist/notes.html`. From http://www.math.upenn.edu/∼ghrist/notes.html.

[42] Robert Ghrist and Sanjeevi Krishnan. Positive alexander duality for pursuit and evasion. 2017.

[43] Robert Ghrist, David Lipsky, Jason Derenick, and Alberto Speranzon. Topological landmark-based navigation and mapping. *University of Pennsylvania, Department of Mathematics, Tech. Rep*, 8, 2012.

[44] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[45] Leonidas J Guibas, Jean-Claude Latombe, Steven M LaValle, David Lin, and Rajeev Motwani. A visibility-based pursuit-evasion problem. *International Journal of Computational Geometry & Applications*, 9(04n05):471–493, 1999. URL `http://www.worldscientific.com/doi/abs/10.1142/S0218195999000273`.

[46] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[47] Allen Hatcher. *Algebraic Topology*. Algebraic Topology. Cambridge University Press, 2002. ISBN 9780521795401. URL `https://books.google.com/books?id=BjKs86kosqgC`.

[48] Geoffrey Hollinger, Athanasios Kehagias, and Sanjiv Singh. GSST: anytime guaranteed search. *Autonomous Robots*, 29(1):99–118, 2010. URL `http://link.springer.com/article/10.1007/s10514-010-9189-9`.

[49] Andrew Howard. Multi-robot simultaneous localization and mapping using particle filters. *The International Journal of Robotics Research*, 25(12):1243–1256, 2006.

[50] Rufus Isaacs. *Differential games: a mathematical theory with applications to warfare and pursuit, control and optimization*. Courier Corporation, 1999.

[51] Simon Karpenko, Ivan Konovalenko, Alexander Miller, Boris Miller, and Dmitry Nikolaev. Uav control on the basis of 3d landmark bearing-only observations. *Sensors*, 15 (12):29802–29820, 2015.

[52] Jonghoek. Kim, Fumin Zhang, and Magnus Egerstedt. A provably complete exploration strategy by constructing Voronoi diagrams. *Autonomous Robots*, 29(3-4):367–380, 2010.

[53] Sven Koenig, Boleslaw Szymanski, and Yaxin Liu. Efficient and inefficient ant coverage methods. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):41–76, 2001. ISSN 1012-2443. doi: 10.1023/A:1016665115585. URL `http://dx.doi.org/10.1023/A%3A1016665115585`.

[54] Andreas Kolling and Stefano Carpin. The graph-clear problem: definition, theoretical properties and its connections to multirobot aided surveillance. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 1003–1008. IEEE, 2007. URL `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4399368`.

[55] David Kortenkamp and Terry Weymouth. Topological mapping for mobile robots using a combination of sonar and vision sensing. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 2)*, AAAI'94, pages 979–984, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence. ISBN 0-262-61102-3. URL `http://dl.acm.org/citation.cfm?id=199480.199508`.

[56] Benjamin Kuipers, Joseph Modayil, Patrick Beeson, Matt MacMahon, and Francesco Savelli. Local metrical and global topological maps in the hybrid spatial semantic hierarchy. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 5, pages 4845–4851 Vol.5, April 2004. doi: 10.1109/ROBOT.2004.1302485.

[57] Michail G Lagoudakis, Evangelos Markakis, David Kempe, Pinar Keskinocak, Anton J Kleywegt, Sven Koenig, Craig A Tovey, Adam Meyerson, and Sonal Jain. Auction-based multi-robot routing. In *Robotics: Science and Systems*, volume 5, page 343C350. Rome, Italy, 2005.

[58] Dimitrios Lambrinos, Ralf Möller, Rolf Pfeifer, and Rüdiger Wehner. Landmark navigation without snapshots: the average landmark vector model. In *Proceedings of Neurobiology Conference Göttingen*, 1998.

[59] Steven M LaValle. *Planning Algorithms*. Cambridge university press, 2006.

[60] Valentine Lebourgeois, Agnès Bégué, Sylvain Labbé, Benjamin Mallavan, Laurent Prévot, and Bruno Roux. Can commercial digital cameras be used as multispectral sensors? a crop monitoring test. *Sensors*, 8(11):7300–7322, 2008.

[61] SeoungKyou Lee, Aaron Becker, Sándor P. Fekete, Alexander Kröller, and James McLurkin. Exploration via structured triangulation by a multi-robot system with bearing-only low-resolution sensors. *CoRR*, abs/1402.0400, 2014. URL `http://arxiv.org/abs/1402.0400`.

[62] John Lim, Nick Barnes, et al. Robust visual homing with landmark angles. In *Robotics: science and systems*, 2009.

[63] Ming Liu, Cédric Pradalier, Qijun Chen, and Roland Siegwart. A bearing-only 2d/3d-homing method under a visual servoing framework. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4062–4067. IEEE, 2010.

[64] Savvas G Loizou and Vijay Kumar. Biologically inspired bearing-only navigation and tracking. In *Proceedings of the IEEE Conference on Decision and Control*, pages 1386–1391. IEEE, 2007.

[65] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[66] Z Lu, TY Ji, WH Tang, and QH Wu. Optimal harmonic estimation using a particle swarm optimizer. *IEEE Transactions on Power Delivery*, 23(2):1166–1174, 2008.

[67] N. Michael, J. Fink, and V. Kumar. Experimental testbed for large multi-robot teams: Verification and validation. *IEEE Robot. Autom. Mag.*, 15(1):53–61, March 2008.

[68] Nathan Michael, Shaojie Shen, Kartik Mohta, Yash Mulgaonkar, Vijay Kumar, Keiji Nagatani, Yoshito Okada, Seiga Kiribayashi, Kazuki Otake, Kazuya Yoshida, et al. Collaborative mapping of an earthquake-damaged building via ground and aerial robots. *Journal of Field Robotics*, 29(5):832–841, 2012.

[69] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598, 2002.

[70] Abubakr Muhammad and Magnus Egerstedt. Control using higher order laplacians in network topologies. In *Proceedings of the 17th International Symposium on Mathematical Theory of Networks and Systems*, pages 1024–1038, Kyoto, Japan, 2006.

[71] Renata Nascimento and Thomas Lewiner. Streamline-based topological graph construction with application to self-animated images. In *Graphics, Patterns and Images (SIBGRAPI), 2013 26th SIBGRAPI-Conference on*, pages 296–303. IEEE, 2013.

[72] Sylvie CW Ong, Shao Wei Png, David Hsu, and Wee Sun Lee. Planning under uncertainty for robotic tasks with mixed observability. *The International Journal of Robotics Research*, 29(8):1053–1068, 2010.

[73] Torrence D Parsons. Pursuit-evasion in a graph. In *Theory and applications of graphs*, pages 426–441. Springer, 1978. URL `http://link.springer.com/chapter/10.1007%2FBFb0070400`.

[74] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*, volume 3, page 5, Kobe, Japan, May 2009.

[75] Rattanachai Ramaithitima and Subhrajit Bhattacharya. Landmark-based exploration with a swarm of resource constrained robots. In *Robotics and Automation (ICRA), 2018 IEEE International Conference on*. IEEE, 2018. to appear.

[76] Rattanachai Ramaithitima, Michael Whitzer, Subhrajit Bhattacharya, and Vijay Kumar. Sensor coverage robot swarms using local sensing without metric information. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3408–3415. IEEE, 2015.

[77] Rattanachai Ramaithitima, Siddharth Srivastava, Subhrajit Bhattacharya, Alberto Speranzon, and Vijay Kumar. Hierarchical strategy synthesis for pursuit-evasion problems. In *Proceedings of the European Conference on Artificial Intelligence (ECAI) 2016 conference*, 2016.

[78] Rattanachai Ramaithitima, Michael Whitzer, Subhrajit Bhattacharya, and Vijay Kumar. Automated creation of topological maps in unknown environments using a swarm of resource-constrained robots. *IEEE Robotics and Automation Letters*, 1(2):746–753, 2016.

[79] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[80] Elon Rimon. Construction of C-space roadmaps from local sensory data. what should the sensors look for? *Algorithmica*, 17(4):357–379, 1997. ISSN 0178-4617. doi: 10.1007/BF02523678.

[81] Rui Rocha, Jorge Dias, and Adriano Carvalho. Cooperative multi-robot systems:: A study of vision-based 3-d mapping using information theory. *Robotics and Autonomous Systems*, 53(3-4):282–311, 2005.

[82] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[83] Anthony Rowe, Mario E Berges, Gaurav Bhatia, Ethan Goldman, Ragunathan Rajkumar, James H Garrett, José MF Moura, and Lucio Soibelman. Sensor andrew: Large-scale campus-wide sensing and actuation. *IBM Journal of Research and Development*, 55(1.2):6–1, 2011.

[84] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

[85] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011.

[86] Samuel Rutishauser, Nikolaus Correll, and Alcherio Martinoli. Collaborative coverage using a swarm of networked miniature robots. *Robotics and Autonomous Systems*, 57 (5):517 – 525, 2009.

[87] Manel Sghaier, Hayfa Zgaya, Slim Hammadi, and Christian Tahon. A distributed Dijkstra's algorithm for the implementation of a real time carpooling service with an optimized aspect on siblings. In *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, pages 795–800. IEEE, 2010.

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

[89] Cyrill Stachniss, Giorgio Grisetti, and Wolfram Burgard. Information gain-based exploration using rao-blackwellized particle filters., 2005.

[90] Andrew Tausz, Mikael Vejdemo-Johansson, and Henry Adams. Javaplex: A research software package for persistent (co) homology, 2011.

[91] Waymo Team. Introducing waymo's suite of custom-built, self-driving hardware, 2017. URL `https://medium.com/waymo/introducing-waymos-suite-of-custom-built-self-driving-hardware-c47d1714563`.

[92] Sebastian Thrun and Arno Bücken. Integrating grid-based and topological maps for mobile robot navigation. In *Proceedings of the AAAI Thirteenth National Conference on Artificial Intelligence*, pages 944–951, Portland, Oregon, 1996.

[93] Sebastian Thrun and Yufeng Liu. Multi-robot SLAM with sparse extended information filers. *Robotics Research*, pages 254–266, 2005.

[94] Roberto Tron and Kostas Daniilidis. An optimization approach to bearing-only visual homing with applications to a 2-d unicycle model. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4235–4242. IEEE, 2014.

[95] E.G. Tsardoulias, A.T. Serafi, M.N. Panourgia, A. Papazoglou, and L. Petrou. Construction of minimized topological graphs on occupancy grid maps based on GVD and sensor coverage information. *Journal of Intelligent and Robotic Systems*, 75(3-4): 457–474, 2014. ISSN 0921-0296. doi: 10.1007/s10846-013-9995-3.

[96] Tinne Tuytelaars, Krystian Mikolajczyk, et al. Local invariant feature detectors: A survey. *Foundations and trends in computer graphics and vision*, 3(3):177–280, 2008.

[97] Israel A Wagner, Michael Lindenbaum, and Alfred M Bruckstein. Distributed covering by ant-robots using evaporating traces. *Robotics and Automation, IEEE Transactions on*, 15(5):918–933, Oct 1999.

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

[99] Brian Yamauchi. Frontier-based exploration using multiple robots. In *Proceedings of the second international conference on Autonomous agents*, pages 47–53. ACM, 1998.

[100] Fumio Yasutomi, Makoto Yamada, and Kazuyoshi Tsukamoto. Cleaning robot control. In *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, pages 1839–1841. IEEE, 1988.