4-16-2018

A Dynamical System Approach for Resource-Constrained Mobile Robotics

Tauhidul Alam
talam005@fiu.edu

DOI: 10.25148/etd.FIDC006561

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

A DYNAMICAL SYSTEM APPROACH FOR RESOURCE-CONSTRAINED MOBILE ROBOTICS

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Tauhidul Alam

2018

To: Dean John L. Volakis
College of Engineering and Computing

This dissertation, written by Tauhidul Alam, and entitled A Dynamical System Approach for Resource-Constrained Mobile Robotics, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

S. S. Iyengar

Bogdan Carbunar

Wei Zeng

Kemal Akkaya

Dylan A. Shell

Leonardo Bobadilla, Major Professor

Date of Defense: April 16, 2018

The dissertation of Tauhidul Alam is approved.

Dean John L. Volakis
College of Engineering and Computing

Andrés G. Gil
Vice President for Research and Economic Development
and Dean of the University Graduate School

Florida International University, 2018

DEDICATION

To my family for their support and encouragement.

ACKNOWLEDGMENTS

I would like to praise Almighty God for his blessings of physical and mental strength to finish my dissertation research. This dissertation would never have been possible without the assistance and support of many people. First, I offer sincere thanks to my advisor, Dr. Leonardo Bobadilla, who helped me complete the research works presented in this dissertation through his continuous encouragement, support, and technical suggestions along the way. He made me interested in doing research on robotics and control theory.

I would like to express my sincere gratitude to all my dissertation committee members: Dr. S. S. Iyengar, Dr. Bogdan Carbunar, Dr. Wei Zeng, and Dr. Kemal Akkaya for their time and valuable feedback to improve my dissertation. My collaborator and committee member, Dr. Dylan Shell, has been an excellent mentor for me. I learned a lot from various insightful discussions with him. He has given many helpful comments on my research.

I shared my work environment with great PhD students: Mahbub, Sebastian, and Greg. It was intellectually stimulating and enjoyable to work together with these smart people who provided me with lots of constructive feedback about my research papers and presentations. Hence, I am thankful to them. I am also thankful to other graduate students: Franklin, Pedro, and Richard who helped me with the experiments and others.

I would have never been able to accomplish anything in life without the unconditional support of my parents. I am always grateful for their love and guidance over the years. Last but not the least, I would like to dedicate this work to my wife, Saima, for her help, encouragement, understanding, and patience. Thank you.

Finally, I would like to acknowledge the financial support of a Florida International University Graduate School Dissertation Year Fellowship. This dissertation was also supported in part by Army Research Office Grant 67736CSII.

ABSTRACT OF THE DISSERTATION

A DYNAMICAL SYSTEM APPROACH FOR RESOURCE-CONSTRAINED MOBILE ROBOTICS

by

Tauhidul Alam

Florida International University, 2018

Miami, Florida

Professor Leonardo Bobadilla, Major Professor

The revolution of autonomous vehicles has led to the development of robots with abundant sensors, actuators with many degrees of freedom, high-performance computing capabilities, and high-speed communication devices. These robots use a large volume of information from sensors to solve diverse problems. However, this usually leads to a significant modeling burden as well as excessive cost and computational requirements. Furthermore, in some scenarios, sophisticated sensors may not work precisely, the real-time processing power of a robot may be inadequate, the communication among robots may be impeded by natural or adversarial conditions, or the actuation control in a robot may be insubstantial. In these cases, we have to rely on simple robots with limited sensing and actuation, minimal onboard processing, moderate communication, and insufficient memory capacity. This reality motivates us to model simple robots such as bouncing and underactuated robots making use of the dynamical system techniques. In this dissertation, we propose a four-pronged approach for solving tasks in resource-constrained scenarios: 1) Combinatorial filters for bouncing robot localization; 2) Bouncing robot navigation and coverage; 3) Stochastic multi-robot patrolling; and 4) Deployment and planning of underactuated aquatic robots.

First, we present a global localization method for a bouncing robot equipped with only a clock and contact sensors. Space-efficient and finite automata-based combinatorial filters are synthesized to solve the localization task by determining the robot's pose (position and orientation) in its environment.

Second, we propose a solution for navigation and coverage tasks using single or multiple bouncing robots. The proposed solution finds a navigation plan for a single bouncing robot from the robot's initial pose to its goal pose with limited sensing. Probabilistic paths from several policies of the robot are combined artfully so that the actual coverage distribution can become as close as possible to a target coverage distribution. A joint trajectory for multiple bouncing robots to visit all the locations of an environment is incrementally generated.

Third, a scalable method is proposed to find stochastic strategies for multi-robot patrolling under an adversarial and communication-constrained environment. Then, we evaluate the vulnerability of our patrolling policies by finding the probability of capturing an adversary for a location in our proposed patrolling scenarios.

Finally, a data-driven deployment and planning approach is presented for the underactuated aquatic robots called drifters that creates the generalized flow pattern of the water, develops a Markov-chain based motion model, and studies the long-term behavior of a marine environment from a flow point-of-view.

In a broad summary, our dynamical system approach is a unique solution to typical robotic tasks and opens a new paradigm for the modeling of simple robotics systems.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF TABLES

LIST OF FIGURES

FIGURE PAGE

CHAPTER 1

INTRODUCTION

1.1 Background

Mobile robotics has had great success in the areas of manufacturing [CLC14], agriculture [LIWY09], healthcare [SG01], home automation [MVH14], military missions [WBL+99], transportation [MFK11], and surveillance systems [GKKP06]. Furthermore, mobile robotics is evolving quickly due to the improvement of sensing technology, computational power, mathematical models, and algorithmic techniques. The proliferation of mobile robots is spreading into many facets of our daily lives. As such, these are exciting times for the robotics research community, which further motivate us to study the robotics systems. The commercial robotics industry and government organizations are also encouraging new research ideas to be brought to life through fundings and robotics competitions. Select examples include:

- Google self-driving car project "Waymo" began test-driving of fully self-driving vehicles on public roads [sel]. It is expected that people will soon use these vehicles in their daily lives. The technological development of these self-driving vehicles started through the DARPA Grand Challenge[1].
- National Institute of Standards and Technology (NIST) organized an Agile Robotics for Industrial Automation Competition (ARIAC)[2] to promote agility in industrial robot systems by utilizing the latest advances in artificial intelligence and robot planning. The goal is to enable industrial robots on the shop

[1] http://www.darpa.mil/grandchallenge/

[2] https://www.nist.gov/el/isdms/agile-robotics-industrial-automation-competition

floors to be more productive, more autonomous, and more responsive to the needs of the shop floor workers.

- The VEX Robotics Competition, presented by the Robotics Education and Competition Foundation, is the largest and fastest growing global robotics program for middle school and high school students[3]. Students, under the guidance of their teachers and mentors, build innovative robots and compete in an exciting engineering challenge.

Additionally, the robotics industry continues to develop sensor-rich mobile robots (Figure 1.1) which increased the availability of autonomous robots in practice. As evidence, Waymo leads in the race of developing self-driving cars (Figure 1.1(a)) and autonomous miles driven by these cars [way18]. In California, they drove 352,545 autonomous miles with 63 total disengagements[4], for a yearly average of 5,595 miles per disengagement from December 2016 to November 2017. They are now expanding their service to more cities across the United States.

Security robots are increasingly being used as guards [sro14]. And Knightscope, Inc. is developing autonomous security robots (Figure 1.1(b)) having a long-term vision to predict and prevent crime by utilizing autonomous security robots, analytics and engagement [sro]. Moreover, an unmanned aerial vehicle (UAV) or a drone is a common robotic platform for numerous civilian, commercial, military, and aerospace applications. For instance, the U.S. Department of Homeland Security built drones (Figure 1.1(c)) that can detect civilians carrying guns and track their cell phones using "signal interception" and "direction finding" technologies [drob]. Commer-

[3]https://www.roboticseducation.org/competition-teams/vex-robotics-competition/

[4]A disengagement is when the driver takes over the car due to technology failure or the intervention for the safe operation.

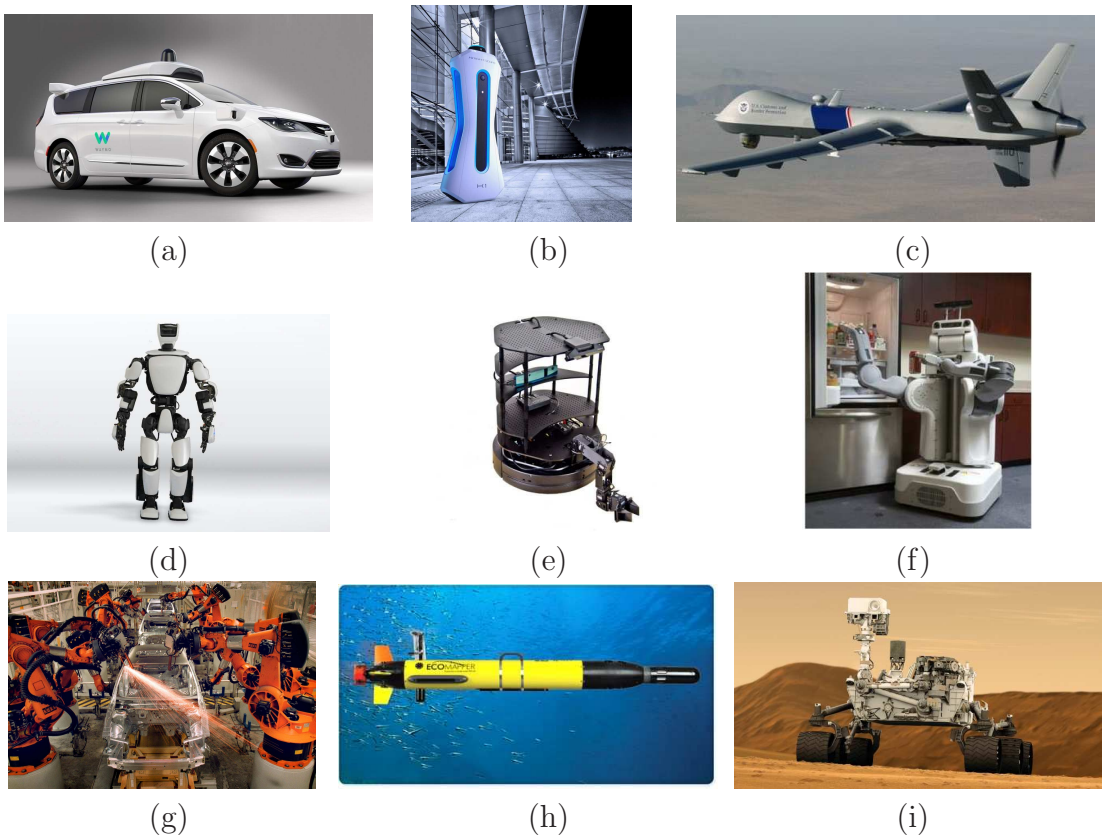


Figure 1.1: Examples of sensor-rich mobile robots: (a) The Google's Waymo self-driving car [sel]; (b) A Knightscope security robot [sro]; (c) The department of homeland security's surveillance drone (UAV) [drob]; (d) The Toyota third generation humanoid robot T-HR3 [hr]; (e) The TurtleBot 2 mobile robot [tur]; (f) The Willow Garage's PR2 robot [pr2]; (g) The Kuka industrial robot [kuk]; (h) The YSI Ecomapper autonomous underwater vehicle (AUV) [gli]; (i) A rendering of the NASA's Curiosity rover for Mars exploration [rova].

cial uses of drones as data capturing devices are also on the rise [droa]. Among the industries and fields where drone technology uses are thriving include security, construction, agriculture, mining, and infrastructure inspections.

Humanoid robots have come a long way since Piero Fiorito unveiled the first gigantic humanoid robot called *Cygan* in 1957. A modern third generation humanoid robot example designed and developed by Toyota is shown in Figure 1.1(d) that can coexist with humans and assist them in their daily lives like mobility needs [hr]. An alternative platform for interacting with humans is a mobile manipulator (Figure 1.1(e)–(g)). The Turtlebot (Figure 1.1(e) and the PR2 manipulator (Figure 1.1(f)) from Willow Garage combine the mobility to navigate human environments and the dexterity to grasp and manipulate objects in those environments [tur, pr2]. Some industrial manipulators from Kuka (Figure 1.1(g)) have long been used in assembly lines for improved efficiency and accuracy which perform repetitive tasks such as spot welding and painting [kuk].

On the other hand, aquatic robots or underwater vehicles provide a means of collecting data about aquatic ecosystems or monitoring a marine environment. One example is the YSI EcoMapper autonomous underwater vehicle (AUV) (Figure 1.1(h)) which can collect bathymetry and water quality data [gli]. NASA's Curiosity rover (Figure 1.1(i)) is expected to drive autonomously to search for actual signs of past Mars life by doing on-site measurements and collecting samples [rovb].

1.2 Fundamental Challenges for Typical Robotic Tasks

The advancement of autonomous vehicles has led to the development of robots with abundant sensors, actuators with many degrees of freedom, high-performance computing capabilities, and high-speed communication devices. These sensor-rich

robots are utilized to solve diverse robotic tasks. Some typical tasks in mobile robotics are as follows [LaV13]:

- **Modeling:** The modeling task aims to develop a mechanically stable system and model the system using system identification [ÅE71]. Probabilistic modeling has proved to be useful in taking noise and model uncertainties into account that arise when using real sensors.
- **Mapping:** Once a robot has a system model, it needs a representation of its environment. The mapping task constructs the representation of the environment accessible by the robot. This environment representation can be occupancy grid maps, topological maps, and bitmaps. Robots can use several sensors such as LIDARs or high-resolution cameras for mapping [TBF05].
- **Localization:** A robot needs to localize itself after having the representation of the environment. Particularly, the localization task determines the robot's pose (position and orientation or the direction it is facing) relative to a given environment taking information from GPS, laser range finder, compass, and camera sensors. There is a subtle distinction between *localizing* a robot and *tracking* a robot. In tracking a robot, the initial robot pose is known. In localizing a robot globally, the initial pose of the robot is unknown which is also called global localization.
- **Navigation:** Given the representation of the environment and a prescribed location in the environment, this navigation task generates a trajectory for a robot to reach its prescribed or goal location as efficiently and reliably as possible. Then, the robot uses its sensor feedback for executing the navigation trajectory. The navigation task involves both *path planning* which is the decision making of a robot about what to do over the long term to achieve its

goal and *obstacle avoidance* which is the modulation of the trajectory of the robot to avoid collisions [SNS04].

- **Coverage:** The coverage task ensures that no regions of interest of the given environment have been left unexplored by the robot. This task is computationally expensive for a single robot. The coverage time is very high for a single robot as opposed to multiple robots. Thus, multi-robot coverage methods have been well studied [Cho01, GC13]. These approaches for solving the coverage task are divided into offline methods, in which the map of the environment is known, and online methods, in which the map of the environment is unknown.
- **Patrolling:** The patrolling task is the activity of going around or through an area at regular intervals for security purposes. Patrolling schemes are further categorized into perimeter patrolling, which is the activity of going through an area and area patrolling, which is the activity of going around an area [PR11]. The patrolling task entails having periodic visits of locations of interest whereas the coverage task requires just one pass. As with coverage, the patrolling task can be performed by single or multiple mobile robots.
- **Coordination:** When multiple robots are involved in a task, their motions need to be synchronized to avoid collisions and solve the task. A centralized controlling unit can work as a coordinator to communicate with multiple robots while solving the task. Robots can also communicate with each other to coordinate themselves.

For solving these typical tasks, the sensor-rich mobile robots use a large volume of information from sensors. However, no real sensor can measure everything entirely and accurately. In other words, perfect sensing is not possible in practice.

Information loss and time delays are expected in processing the wealth of sensor information. Only a *hypersensor* instantaneously measures everything, records it in the storage, and responds immediately. The completeness, accuracy, and timeliness of the information are critically dependent upon the hypersensor. Hence, solving a task using a sensor-rich mobile robot usually leads to a significant modeling burden as well as excessive cost, sensing, communication, memory, and computation requirements. Because of these issues, it is better for a robot to require less information from sensors about the physical world.

Furthermore, some extrinsic sensors such as GPS may not work precisely in indoor and underwater environments, and compass readings can be disturbed by electromagnetic fields. In some cluttered environments, visual perception can be ineffective and expensive in poorly illuminated conditions. In privacy-preserving environments, some sensors, e.g., a camera may be prevented from being used for data collection. The real-time processing power of a robot may be inadequate and the actuation control in a robot may be insubstantial. Natural or adversarial situations can impede the communication among multiple robots in a congested environment. For security reasons, the communication among robots may be kept at a minimum level. In these scenarios, we can alternatively rely on simple robots. Therefore, the planning and control of simple robots are required that will satisfy various objectives such as making the robot behavior unpredictable, minimizing the traveling duration, sensing information, actuation control, communication, and computation, or handling the limited visibility of the robot.

(a) (b)

Figure 1.2: Simple household robots: (a) The iRobot Roomba vacuuming robot [iro]; (b) The RoboMop floor duster [rmo].

1.3 Motivation

The development of sensor-rich mobile robots often starts with the introduction of simpler systems. The prevalence of sensors with lower costs and the trend of developing small mechanical, sensing, and computing components are making robots to enter our homes. Consequently, some simple robots are being incorporated into our household chores. The well-known Roomba vacuum cleaning robot (Figure 1.2(a)) lives in many households and uses zig-zag and wall following techniques for vacuuming. The Weaselball is a $4 toy that contains only a battery connected to an oscillating motor. One of the simplest robot designs is the RoboMop, which is essentially a Weaselball enclosed in a dusting ring (Figure 1.2(b)) and picks up dust as it rolls. Experimental mobile robotics is the main inspiration for the concepts presented in this dissertation. Considering the availability of simple robots, the primary motivation of our work is to use simple and inexpensive robots in solving fundamental robotic tasks.

More importantly, there are some stimulating reasons why we use simple or resource-constrained robots.

1. Simple robots have inexpensive actuators and sensors which make them suitable for long-term deployments, numerous applications, and research prototyping and verification. Moreover, simple robots can also reduce the engineering costs associated with sensor calibration and manufacturing.

2. There are limitations of sensors in some environments as explained before, e.g., GPS-denied, communication-challenged, cluttered, privacy-preserving, and adversarial environments. Simple robots can overcome these limitations using a minimal number of sensors.

3. Statistical assumptions about motion errors and sensor noise to model the uncertainty are not always accurate. The minimal number of sensors in the simple robots cause them to be less susceptible to failure and make them robust against the sensing uncertainty of a lot of sensors.

4. Simple robots consume less energy because they can feed sensor outputs directly to actuators, e.g., motors, and they use some minimal amount of logic circuitry to maintain sensor observations.

5. There is less sensed and transmitted information for a robot to solve a task which in turn reduces the computation time, communication, and memory requirements for the robot in addressing the task.

6. The code and models of these robots are more straightforward to verify since a small code base and simple communication protocols are easily deployable to real robots to evaluate the efficacy of the system.

7. The controlling techniques of simple robots can be used to enhance the robustness of sensor-rich robots by providing the solution to the intended task even when some sensors do not work. Moreover, the complex tasks in in-

dustrial settings can be carried out by simple robots reducing the associated cost [CG94].

8. With the advent of wireless sensor networks [PK00], a group of simple micro-robots offers an attractive and scalable architecture for large-scale collaborative tasks.

The *minimalist* and resource-constrained approach, which we take throughout this dissertation, has also been applied to solve several robotics problems. One inspiring example is the article of Mason [Mas93] which stated that robotic systems must be less dependent on complicated sensors as they are subject to significant errors. A planner for manipulating objects on a planar surface using minimal information was proposed in [EM88, AM98]. Canny and Goldberg [CG95] proposed sensing and manipulation strategies for performing complex operations from simple actuation and sensing elements. Yershova et al. proposed that simple robots can solve complex tasks based on the concept of *information spaces* avoiding the need for accurate state estimation [YTGL05]. This information space concept was the motivation for the construction of minimalist *combinatorial filters* that maintain minimally needed information to achieve a specified task [TCB$^+$14, OS17]. Combinatorial filters are the minimalist counterpart to the popular *recursive Bayesian filters* such as the Kalman filter [Kal60] and its extensions or particle filters [DGA00]. Bayesian filters are mostly applied in Robotics and Control to keep an estimate of state variables such as position and velocity. These filters represent the state uncertainty through probability distributions and update uncertainty using a state transition model and an observation model. In contrast, combinatorial filters handle uncertainty in discrete spaces that represent the essential information to solve a task.

1.3.1 Dynamical System Methods

The striking motivation for the approaches in this dissertation is the global analysis of simple robotics systems. This global analysis of robotics systems leads us to use a dynamical system method. The dynamical system we use here is the *cell-to-cell mapping* methodology (originally introduced by Hsu in 1980) [Hsu80, Hsu13]. In the cell-to-cell mapping, the state space is divided into small cells, where each cell is considered a state entity. In our approaches, we utilize two cell-to-cell mapping methods which are the *simple cell-to-cell mapping* (SCM) and the *generalized cell-to-cell mapping* (GCM). In the SCM, each cell has only one image cell. In the GCM, each cell has several image cells. The GCM is a generalization of the SCM. The modeling of the deterministic behavior of robots leads to the application of the SCM. The formulation of the nondeterministic behavior of robots in terms of the GCM leads to a finite Markov chain.

The cell-to-cell mapping based method is uncommon in solving the robotic tasks. Only the trajectory planning for manipulators utilized the cell mapping method before [ZL90, WL94]. Therefore, our approaches in this dissertation create a unique opportunity to apply the dynamical system method in solving diverse robotic tasks.

1.4 Key Themes and Contributions

In this dissertation, we propose a four-pronged approach for solving basic robotic tasks in resource-constrained scenarios using simple robots with limited sensing and actuation, minimal onboard processing, moderate communication, and limited memory capacity.

1.4.1 Combinatorial Filters for Bouncing Robot Localization

In the first technical contribution of the dissertation, we focus on a setup that considers a polygonal environment with holes and a simple robot equipped only with a clock and contact (or bump) sensors called a *bouncing robot*. We consider that the bouncing robot has access to a map of its environment, but is initially unaware of its position and orientation within that environment. This bouncing robot is modeled in a predictable way: the robot moves in a straight line and then bounces from the environment's boundaries by rotating in place counterclockwise through a bouncing angle. The problem of global robot localization is how the robot deduces its pose (position and orientation) following its modeled behavior. Can this bouncing robot be globally localized without even knowing its initial pose? Different methods have been proposed to address this localization problem for robots with limited sensing [OL07, EKOL08, EL13]. In this contribution, we synthesize finite automata-based combinatorial filters for the global robot localization that take less computation time and memory compared to traditional Bayesian filter-based localization approaches [TFBD01, Fox03, LDW91].

The main overview of this contribution is presented as a sequence of steps:

- **Geometry to Groups:** An algorithm based on the SCM [Hsu80] is proposed to find the periodic groups and their transient trajectories from the environment.
- **Groups to Information States:** We construct information state (I-state) graphs [LaV06] from the computed periodic groups and transient trajectories.

- **Information States to Filters:** We introduce combinatorial filters that are generated from I-state graphs and enable the robot to localize itself up to some intrinsic uncertainty.

1.4.2 Bouncing Robot Navigation and Coverage

In the second contribution, we use the same bouncing robot model to investigate both the navigation and coverage problems. The problem of navigation is finding a path for a robot between an initial pose and a goal pose. The coverage problem of the environment is visiting all locations of interest using one or more robots. How could the simple behavior of the bouncing robot be useful in solving the common robotic tasks, such as navigation and coverage, with limited linear and angular sensing? In multi-robot settings, will many such bouncing robots be useful as well to solve the coverage task? Our solution in this contribution has the following steps: 1) A directed graph is constructed from the environment geometry using the simple bouncing policies. 2) The shortest path on the graph, for navigation, is generated between either one given pair of initial and goal poses or all possible pairs of initial and goal poses. 3) The optimal distribution of bouncing policies is computed so that the actual coverage distribution is as close as possible to the target coverage distribution. Our solution also finds a multi-robot (or joint) trajectory incrementally for multiple robots to cover the given environment.

The contribution has the following steps:

- We propose an algorithm using the SCM to find the minimum navigation plan for a minimalist robot between an initial and a goal configuration in the environment.

- All minimum navigation plans between all possible initial and goal configuration pairs in the environment are generated.
- A method based on the GCM [Hsu80] is developed for finding a probability distribution of bouncing policies for the best possible coverage of the environment with respect to a target coverage distribution.
- We also present an algorithm based on the SCM to find a joint trajectory of multiple bouncing robots for covering a known environment.

1.4.3 Stochastic Multi-Robot Patrolling

In the third contribution, we investigate the problem of area patrolling in an adversarial situation in which a number of robots as patrollers visit a group of locations of interest in an environment to detect the intrusion of an adversary. In a communication-constrained and adversarial environment, it is a challenging problem for multiple robots to patrol the whole environment by sensing with their limited ability to see. In the multi-robot patrolling problem, what will be an efficient method for robots to patrol an area under the adversarial scenario? How can we remove the need for synchronization and coordination among the patrolling robots? How can the robots with limited visibility be used to patrol an adversarial and communication-constrained environment? Deterministic patrolling strategies could also be learned by an adversary observing them over time. In this contribution, we alternately use randomized patrolling strategies based on Markov chains for several reasons: 1) These will make it harder for an adversary to successfully complete an attack and evade its detection due to the unpredictability of the strategies. 2) A randomized motion can be easily implemented in a mobile robot, since its communi-

cation, sensing, and computation requirements are minimal. 3) Efficient algorithms can calculate Markov chains with desired properties [GBS08].

This contribution is summarized as follows:

- We present algorithms that do not require communication, are based on convex optimization, can scale well, and can also be applied to any type of environment represented as a graph where the distribution of cost over locations (vertices) is uniform for both patroller and adversary.
- A game theoretical framework is proposed for patrolling where the set of strategies are Markov chains. We also calculate the payoffs of each strategy and present approaches to generate the optimal mixed strategy for patrollers and the optimal strategy for the adversary.
- We propose a method for finding distributed patrolling policies based on limited visibility regions and convex optimization, where each patroller monitors the whole environment separately.
- We develop centralized and randomized patrolling policies using a central base station and visibility-based communication, where each patroller patrols a region of the environment and contacts the base station after a random period.
- The vulnerability of our patrolling policies is evaluated by finding the probability of capturing an adversary at a specific location of the environment when patrollers follow our proposed policies.

1.4.4 Deployment and Planning for Underactuated Aquatic Robots

In the final contribution, we are interested in tackling the problem of deploying multiple underactuated aquatic robots called *drifters* so that their desired long-term trajectories can gather aquatic data visiting all locations on the surface of a marine environment. We also tackle the problems of path planning and finding navigation policy for a drifter. The drifters drift passively with ambient ocean currents. Vertical actuation (buoyancy) enables them to alter their depth and achieve controllability by the use of different current layers in the ocean. How can we model the behavior of the drifter in a marine environment? In addition, the study of a marine environment is a challenging task because of the spatiotemporal variations of ocean phenomena and the disturbances caused by ocean currents. As such, we must collect data from a marine environment over long periods of time to better assess and understand a marine environment. The uncertainty of the drifter motion due to the disruption of ocean currents and winds needs to be taken into account in our motion model of the drifter. In this contribution, we present a data-driven, deployment and planning approach for the drifters. We extract the generalized flow pattern within a given region from ocean model predictions, develop a Markov chain-based motion model, and analyze the long-term water flow behavior. Based on this long-term behavior of the water flow, we find a minimum number of deployment locations for the drifters in the marine environment. All possible reachable locations from an initial deployment location of the drifter are determined as its planned, long-term drifter trajectory. An optimal navigation policy is developed to demonstrate the best possible action from any location to a goal location in the environment.

The overview of this contribution is given as below:

- An algorithm is proposed to deploy a minimum number of drifters at their starting locations on the water surface to cover an oceanic region over a given (long) period of time.
- A deployment policy is developed for the visibility-based, total coverage problem that requires images of an entire seafloor environment through the long-term and cumulative image collection.
- We compute the reachability set for a drifter, i.e., all possible reachable locations for the drifter over an extended period of time starting from its initial deployment location.
- We propose an optimal navigation policy to find the best possible action from any location to a goal location of the environment for drifting vehicles.

1.5 Organization of the Dissertation

We conclude this introductory chapter with a preview of the remainder of the dissertation. Chapters 2, 3, 4, and 5 contain original contributions. Concluding remarks, open problems, and some potential avenues for future work appear in Chapter 6. The structure and dependencies between chapters are shown in Figure 1.3. The contributions of this dissertation are laid out in the following chapters as follows:

- **Chapter 2:** We describe the synthesis of combinatorial filters for a bouncing robot localization based on the output of the modified SCM. Section 2.1 provides the motivation and challenges of the global robot localization with limited sensing. Section 2.2 reviews the literature of robot localization and

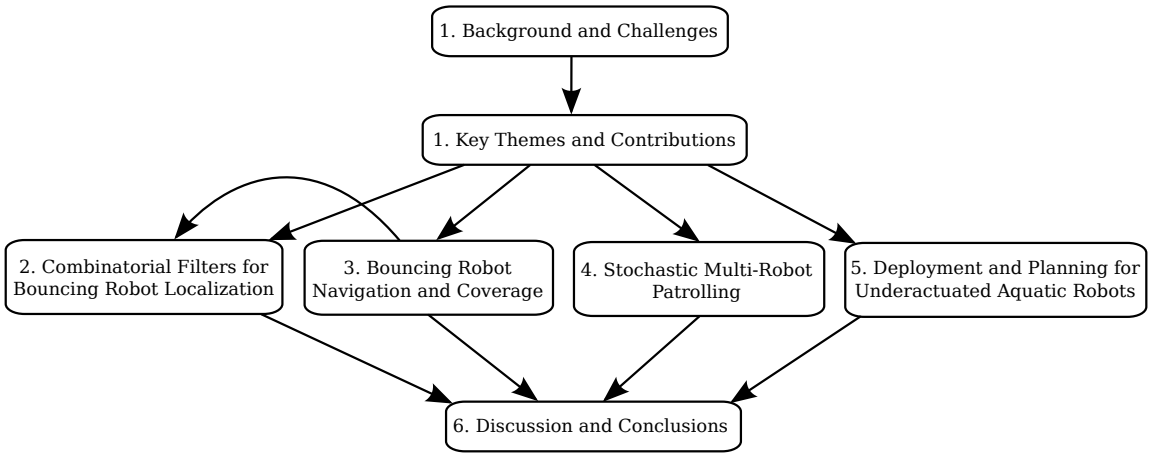


Figure 1.3: Organization of this dissertation with arrows indicating dependencies.

combinatorial filters. Section 2.3 defines the robot model, explains the concepts of the SCM, and formulates the problems we solve. In Section 2.4, we describe the methodology of our work in detail. Section 2.5 illustrates our simulation results and physical deployments of our filters on a real robot. Finally, we conclude our first contribution with the discussion in Section 2.6.

- **Chapter 3:** We present our work of solving navigation and coverage problems using one or more bouncing robots based on the output of both the SCM and the GCM. In Section 3.1, the problem statement and challenges in solving these problems with limited linear and angular sensing are initially explained. Section 3.2 discusses the related literature of the simple robot navigation and coverage. In Section 3.3, we account for the robot model and the fundamentals of cell-to-cell mapping, and then formulate the problems we consider. Section 3.4 describes our proposed approach to solve problems as formulated, in detail. Then, we outline the implementation of our approach with simulation results and hardware experiments in Section 3.5. Section 3.6 explains the concluding remarks of the second contribution.

- **Chapter 4:** A scalable method is described to find stochastic strategies for multi-robot patrolling under an adversarial and communication-constrained environment. In Section 4.1, we motivate the problem of patrolling in adversarial settings with the limitations of the traditional methods. Section 4.2 reviews the literature on the multi-robot patrolling problem. Section 4.3 explains the preliminaries of our work and formulates the problems of our interest. In Section 4.4, we outline the method of finding both distributed and centralized patrolling policies and their vulnerability evaluation in detail. Section 4.5 presents the simulation results and the physical implementation of our proposed method. Finally, we summarize our work of the third contribution in Section 4.6.

- **Chapter 5:** A data-driven deployment and planning approach for an underwater vehicle is presented in this chapter. Section 5.1 introduces an underactuated underwater vehicle called drifter and how we can use this simple vehicle for the long-term assessment of an aquatic ecosystem. After this introduction, Section 5.2 discusses some works related to the applications of underwater vehicles for persistent monitoring. In Section 5.3, the preliminary description of the environment and the motion model of drifters are explained. Also, our problems of interest are formulated in the same section. Section 5.4 outlines the proposed algorithm of our work. The simulation results of the algorithm are presented in Section 5.5. Section 5.6 summarizes our work of the final contribution.

CHAPTER 2

COMBINATORIAL FILTERS FOR BOUNCING ROBOT LOCALIZATION

In this chapter, we present a global localization method based on combinatorial filters for a bouncing robot equipped with only a clock and contact sensors. This work appeared in its current form in [ABS18].

2.1 Motivation and Challenges

Mobile robot localization is the problem of determining a robot's pose or configuration (position and orientation) in an environment, typically within a given map or a similar representation [LaV06]. Localization is a fundamental problem in mobile robotics, and is typically a prerequisite to solving tasks such as navigation, coverage, mapping, searching, planning, and patrolling for applications in agriculture, security, surveillance, and home robotics among many others. This work addresses the problem of global robot localization [TFBD01, FBT99], where a robot has to find its configuration in the entire environment without having any information about its initial configuration. Most localization approaches rely on Bayesian filters such as particle filters [TFBD01, Fox03] or Kalman filters [LDW91, JLV99], which, compared with the focus of our study, are far more expensive in terms of computation time and memory, and require sophisticated sensors and motion modeling. The originality of our work is that we synthesize finite automata-based combinatorial filters for solving the problem of localizing a robot in a particular environment that is suitable for a device of meager computational ability, potentially even being realized directly in a field programmable gate-array. This work fits within a broader research program of hardware synthesis for robots.

With few or very limited sensors, the localization problem is challenging to solve and has consequently attracted considerable theoretical attention [OL05, OL07, EKOL08, EL13]. The motivation of our work is to use a robot with limited linear and angular sensing as a basis for investigating the intrinsic limits of the localization problem. In particular, we wish to understand the strongest possible version of a localization task that such a robot can solve, recognizing that the robot may be too deficient, ultimately, to resolve its position down to a unique pose with certainty. What is possible depends on the environment and parameters of the robot controller, so we explore automated processes to uncover answers to these questions that are given in a particular setting as input.

Continuing the growing vein of work exploring the properties of sensing-constrained systems, we examine a robot equipped with a bump (or contact) sensor and a clock. The robot inhabits a planar polygonal environment with holes and has behavior parameterized by a single parameter, which, for reasons that will be obvious, we call the *bouncing angle*. The robot moves on straight lines, and when it encounters a wall it rotates through the bouncing angle (measured with respect to the direction of its pre-collision motion).

Though the robot is too deficient to localize in the traditional metric sense, we show that there is a relaxed instance of the localization problem that it is capable of solving. The setting we study enables the construction of an estimator that still suffices to localize with an accuracy that is compromised only by the symmetries involved. In other words, we make the limits of localization accuracy precise by establishing the fundamental limits imposed by symmetry as revealed by the robot's sensors. This contribution is also motivated by the concepts of limit cycles and basins of attraction which we define here as periodic groups and transient trajectories respectively and are often used in control theory [Hsu13] and control of robots with

sophisticated dynamics [BRK99]. Consequently, we modify the simple cell-to-cell mapping (SCM) to find periodic cycles and transient trajectories of the robot path as it bounces within an environment filled with obstacles. Based on the cycles and transient trajectories, space-efficient and automata-based combinatorial filters are synthesized to solve localization problems modulo symmetries.

2.2 Related Work

2.2.1 Robot Localization

There are several antecedent works which have examined bouncing robots in related contexts. In [LO13, EKOL08], the authors consider a robot whose bouncing angle varies as a function of the number of prior bounces. In [EL13], the bouncing angle of the robot is a constant angle relative to the normal of the impacted edge of the environment irrespective of its angle of incidence. These contrast from the type of bounce we study. The bounce we investigate ensures that the robot will end up in a small bounded set of possible locations.

In [OL07], the authors study a robot equipped with a contact sensor and compass or a robot equipped with linear and angular odometers, providing theoretical results on localization for environments without holes using geometric reasoning. The robot they study is more powerful than what we explore herein. Further, holes within the environment pose no special challenge for the techniques we describe below. We note that in [EKOL08], the authors considered a simple environment with holes for localization with a robot having only a clock and a contact sensor. They presented a probabilistic technique for finding a probability distribution over regions on the

boundary of the environment. However, they assume that the robot knows its initial orientation whereas in our work, no such assumption is needed.

2.2.2 Combinatorial Filters

Nearly sensorless robots called "weasel balls" that bump and bounce around the environment were studied in [BSC+12]. Their bounce is not associated with a fixed angle and thus require complete state estimation for solving various tasks. Since doing so is difficult, an information space view was introduced in [TYOL05] to avoid this onerous state estimation. The information space consists of all histories of actions and sensing observations of a mobile robot for problems involving uncertainty. A related perspective is adopted in the information state (I-state) formalism, which led to the use of combinatorial filters to process information from sensors for solving tasks such as manipulation [KS12], navigation [TMCL07], and target tracking [YL12]. In [OS13], the problem of filter reduction is introduced, which involves finding the filter that uses the fewest information states for a given filtering task. To the best of our knowledge, our work is the first to automate the process of compiling a geometric description of the environment into an I-state graph, which we then explicitly turn into a filter to solve the localization task.

2.3 Model and Definitions

2.3.1 Robot Model

We start with a differential drive mobile robot equipped with only a contact sensor and a clock. The robot moves in a planar and bounded polygonal two-dimensional *workspace* $\mathcal{W} \subset \mathbb{R}^2$. There is a set of polygonal *obstacles* represented as $\mathcal{O} \subset \mathcal{W}$. Let

$E = \mathcal{W} \setminus \mathcal{O}$ be free-space in which the robot can move freely and let ∂E represent the boundary of the free space. We assume that the robot has a map of the environment E and knows its bouncing angle ϕ but does not know its initial configuration. We also consider a noise-free model of the robot in terms of the translation and the rotation. Certainly, generating perfect motions for any angle poses a problem, especially for a low-cost differential drive robot. However, in practice, we found that for some given ϕ, we are able to produce repeatable and reliable rotations (see Section 2.5, where we describe our physical robot experiments).

The robot moves straight until touching the boundary of the environment ∂E which is detected by the contact sensor. The robot measures the number of *steps* in its straight-line motion by using its clock. Once it bounces at ∂E, the robot rotates with the angle ϕ counterclockwise from its current orientation by commanding a constant angular velocity and using a clock to rotate for some fixed period of time. It then moves straight until contacting ∂E and repeats the behavior. This simple behavior is illustrated in Figure 2.1.

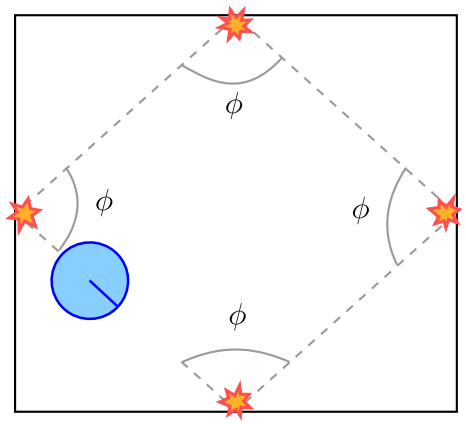


Figure 2.1: An example of a simple bouncing robot.

2.3.2 Simple Cell-to-Cell Mapping

Including the robot's orientation, the physical state space of the robot is $X = E \times S^1$ where $S^1 = [0, 2\pi)$. Let $x \in X$ denote the state of the robot where $x = (x_t, y_t, \theta)$, (x_t, y_t) is its position, and θ is its orientation. Let $R(x) \subset \mathbb{R}^2$ represent the robot. The obstacle region X_{obs} in the state space is defined as

$$X_{\text{obs}} = \{x \in X | R(x) \cap \mathcal{O} \neq \emptyset\} \tag{2.1}$$

and $X_{\text{free}} = X \setminus X_{\text{obs}}$.

The subset of the state space where the robot is allowed to move is denoted by X_{free}. To apply the cell-to-cell mapping method [Hsu80, vcS94], we divide X_{free} into equally sized 3-dimensional box cells since the robot's configuration has three degrees of freedom. Let N be the total number of cells. We define N as $N = N_E \times N_S$, where N_E is the discretization resolution of the 2-D free space E and N_S is the discretization resolution of S^1. This discretized state space is called *cell state space*. Each cell represents an indivisible state entity. The state of the system is described by a cell index $z \in \{1, \ldots, N\}$. Let $Z = \{1, \ldots, N\}$ denote the collection of cells.

The evolution of a system can be explained as a sequence of cells by investigating its state at discrete times. Let $e(i)$ denote the cell containing the state of the system at $t = i\Delta t$, $i = 0, 1, \ldots$ with Δt being the time between two state examinations, and being large enough to support crossing a cell. The system evolution is then governed by

$$e(i+1) = \mathcal{C}(e(i)), \tag{2.2}$$

where the mapping $\mathcal{C} : \mathbb{N} \to \mathbb{N}$ is called a simple cell-to-cell mapping (SCM). In this model, Equation 2.2 implies that the next state of the system is determined entirely by its current state and is explicitly independent of the mapping step i.

For the sake of completeness, we summarize some important definitions of the cell-to-cell mapping method. An extended treatment can be found in [Hsu13].

Definition 2.3.1 (Periodic Cell) *A cell z satisfying $z = \mathcal{C}^m(z)$, for some $m \in \mathbb{N}$ is called a periodic cell with a period of m.*

Definition 2.3.2 (Transient Cell) *A cell that is not periodic is called a transient cell and it maps into a periodic cell in a finite number of steps.*

Definition 2.3.3 (Periodic Group) *A sequence of K distinct cells $e(m)$, where $m = 1, 2, \ldots, K-1$, that satisfies*

$$\begin{aligned} e(m+1) &= \mathcal{C}^m(e(1)), m = 1, 2, \ldots, K-1 \\ e(1) &= \mathcal{C}^K(e(1)), \end{aligned} \tag{2.3}$$

is called as a periodic group with a period K and each of the cells $e(\cdot)$ is said to be a periodic cell with the period K. This periodic group is also called an attractor or a limit cycle.

Definition 2.3.4 (Transient Trajectory) *A transient trajectory is the set of initial cells that are finally leading to a particular periodic group (attractor). The collection of transient trajectories is called a basin of attraction.*

2.3.3 Problem Formulation

As the robot moves in the environment E, it receives a sequence of *observations* from an *observation space* $Y = \{0, 1\}$. Given some agreed upon resolution, the robot can measure the distance by the number of *steps* between bounces up to some quantization error. For example, the observation string that represents the stream of observations that the robot in Figure 2.1 is processing might be $\{0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0\}$,

where the ones represent a bump event, and the zeros otherwise. Depending on scale, resolution, or both, there could be more than or less than three 0s between bumps. It is also possible to observe multiple 1s in a row (for example, it may happen when the robot bounces in a corner). This abstract symbolic representation can be realized with various implementations:

- The robot measures the number of steps for linear distance traversed since the last bump by a number of 0s. This measurement is quantized at some resolution.
- The robot moves forward at a constant speed and keeps observing a sequence of zeros. A bump event results in observing a 1 and resets the clock for the next linear distance measurement. These observations are encoded as a string of 0s, interspersed with 1s.

For a fixed bouncing angle ϕ, the cell-to-cell mapping method allows one to track the motion of the robot from any initial location in E and to find periodic groups, of which we assume there are r in total. Sometimes the robot's motion begins in a transient trajectory and sometimes it is already in a periodic group. It will eventually converge to one of the r periodic groups.

We are interested in the following problems:

Problem 1. Closed-world localization:

Given an environment E, a bouncing angle ϕ, the fact that robot x can only be within E, find the state of robot x as precisely as possible.

Problem 2. Open-world localization:

Given an environment E and a bouncing angle ϕ, find the state of robot x, determining whether the robot is within E and if so ascertain the state of the robot as precisely as possible; otherwise indicate that the robot is not in E.

2.4 Method

This section describes the sequence of steps that produce discrete filters for localization. It consists of the following steps: 1) find the periodic groups and transient trajectories and construct I-state graphs based on them; 2) create a nondeterministic automaton combining I-state graphs and 3) convert the nondeterministic automaton into a deterministic automaton to design filters that solve Closed- and Open-world localization problems.

2.4.1 Finding Periodic Groups and Transient Trajectories and Constructing Periodic Group I-State Graphs

In our method, we modify the simple cell-to-cell mapping to find all periodic groups (attractors) and associated transient trajectories (basins of attraction) [vcS94] from a cell state space X_{free}. We also borrow the definition of an I-state graph from [OS13], though doing away with the starting vertex.

Definition 2.4.1 (I-State Graph) *An I-state graph $G = (V, E, \ell : E \to Y)$ is an edge-labeled directed graph where:*

1. *V is the finite set of vertices consisting of* I-states.
2. *E is the set of edges that represent transitions between vertices.*
3. *ℓ is the function that represents edges labeled by an observation in Y.*

This I-state graph encodes the information state introduced by LaValle [LaV06], integrating the history of observations made by a system during its execution. As the number of cells is finite, we can construct an I-state graph for each periodic

group along with its transient trajectories, which we term a *periodic group I-state graph*.

In this step, Algorithm 2.1 receives as input the geometric description of the environment E and a bouncing angle ϕ, finds all r periodic groups P, consisting of periodic cells and transient trajectories T, consisting of transient cells, for these r periodic groups, and constructs a set of r periodic group I-state graphs denoted by $\mathcal{G}(V, E)$ as output.

In Algorithm 2.1, cells are assigned a group number and a step number. For each cell $z \in Z$, the group number g_z denotes the periodic group to which z belongs, the step number s_z denotes the number of mappings necessary for z to end up in a periodic group, and the next mapped cell is denoted by c_z. Initially, all cells are identified as *virgin cells* by assigning their group number zero. Each virgin cell $z \in Z$ determines the location (centroid) and orientation of a cell (line 6). In lines 7–12, a cell sequence $z, \mathcal{C}(z), \mathcal{C}^2(z), \cdots, \mathcal{C}^k(z)$ where $k \in \mathbb{N}$ and $k \leq N$, is generated for each virgin cell $z \in Z$ and cells in the sequence are identified as *cells under processing* by temporarily assigning to them their group number -1. The next mapped cell z' represents the subsequent cell after z. Thus, z' (line 9) is computed as:

$$\begin{aligned} x' &= x + \cos\theta, \\ y' &= y + \sin\theta, \\ \theta' &= \begin{cases} \theta, & \text{if } (x', y') \in E, \\ (\theta + \phi) \mod 2\pi, & \text{otherwise.} \end{cases} \end{aligned} \tag{2.4}$$

where ϕ is the bouncing angle of the robot. The cell number of z' is calculated from the center location and orientation, (x', y', θ'), of z' (line 10). Then, z' is stored in c_z and z is updated with z' (line 11). The generation of cell sequences is continued

as long as z is a virgin cell, which also means it does not have the next mapped cell. This sequence generation is terminated in one of the following two cases:

1. If z has appeared again in the same sequence then all the cells in the sequence form a cycle. This case can be further subdivided into two scenarios, as illustrated in Figure 2.2. In the first scenario of Figure 2.2(a), when the initial and ending cells are the same, then all cells in the sequence are classified as periodic cells. In the second scenario of Figure 2.2(b), when the initial and ending cells are different, then the cells prior to the cell that forms the cycle are classified as transient cells and the rest of the cells, which form the entire cycle, are classified as periodic cells.
2. If z appeared in one of the previous sequences then all the cells in the sequence are classified as transient cells.

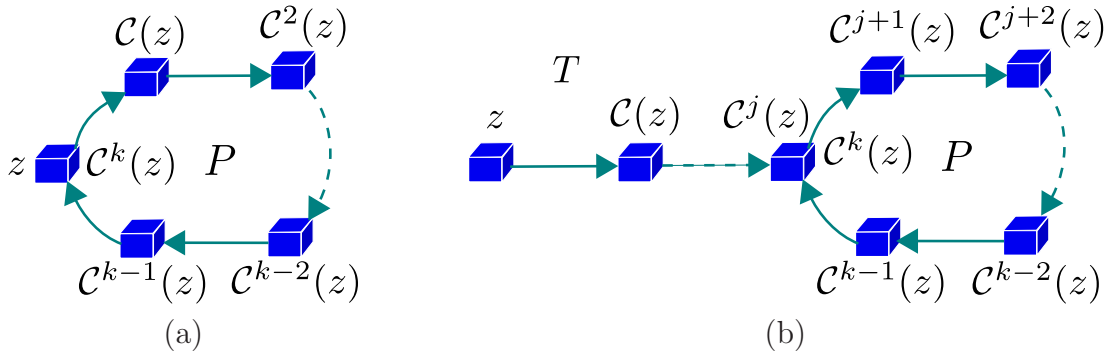


Figure 2.2: Two cycle forming scenarios in the cell sequence: (a) Same initial and ending cells; (b) Different initial and ending cells.

All periodic cells in the j-th periodic group where $j \in \{1, \ldots, r\}$ are found with the update of their group number j and step number as zero (lines 16–18, 23–25). All transient cells in transient trajectories of the j-th periodic group are found with the update of their group number j and corresponding mapping number to get to the j-th periodic group as a step number (lines 20–22, 27–29).

Algorithm 2.1: ModifiedSimpleCellMapping(E, ϕ)

Input: E, ϕ – Environment and bouncing angle
Output: $\mathcal{G} = \{G_1, \cdots, G_r\}$ – Set of periodic group I-state graphs

```
g[1,...,N] ← 0,   s[1,...,N] ← ∞,   c[1,...,N] ← ⊥
r ← 0
for i ← 1 to N do
    if g_i == 0 then
        k ← 0,   z ← i
        x, y, θ ← CellConfiguration(z)
        while c_z == ⊥ do
            g_z ← −1
            x', y', θ' ← NextCell(x, y, θ, φ)
            z' ← CellNumber(x', y', θ')
            c_z ← z',   z ← z'        // save and update next cell
            k ← k + 1
        if g_z == −1 then                          // new periodic group
            r ← r + 1, P_r ← ∅, T_r ← ∅
            if i == z then              // same initial and ending cells
                for j ← 0 to k − 1 do               // add periodic group
                    g_i ← r, P_r ← P_r ∪ {i}, s_i ← 0
                    i ← c_i
            else                    // different initial and ending cells
                for j ← 0 to d − 1 do              // cycle at d-th index
                    g_i ← r, T_r ← T_r ∪ {i}, s_i ← d − j
                    i ← c_i             // add transient trajectory
                for j ← d to k − 1 do               // add periodic group
                    g_i ← r, P_r ← P_r ∪ {i}, s_i ← 0
                    i ← c_i
        else        // cell appeared in one of the previous sequences
            for j ← 0 to k − 1 do             // add transient trajectory
                g_i ← g_z, T_r ← T_r ∪ {i}, s_i ← s_z + k − j
                i ← c_i
G ← {BuildI-StateGraph(P_i, T_i) | i ∈ {1,...,r}}
return G
```

For each periodic group and its associated transient trajectories, Algorithm 2.1 adds two consecutive cells to the vertex set, and their ordered pair to the edge set of the corresponding periodic group I-state graph, using the function BuildI-StateGraph (line 30). In this function, the absolute difference between orientations of these two consecutive cells, i.e., $|\theta - \theta'| > 0$ is checked. If the difference is not greater than zero, which means the robot moves forward with the same orientation, then the edge of this consecutive cell pair is labeled with 0. Otherwise, the transition between vertices causes a 'bump' event at the boundary of the environment $\partial E \subset E$ and the robot changes its orientation from θ to θ', thus the edge of this consecutive cell pair is labeled with 1. After construction, each periodic group I-state graph forms an octopus-like structure.

We repeat the same procedure for all r periodic groups and union the disjoint graphs. Thus, the set of r periodic group I-state graphs $\mathcal{G}$ is constructed. The total number of vertices of the r periodic group I-state graphs $\mathcal{G}$ is $|\mathcal{G}.V| = N$. We denote the set of vertices in the periodic groups of $\mathcal{G}$ as $\mathcal{G}.V_P$ where $\mathcal{G}.V_P \subset \mathcal{G}.V$. We illustrate one periodic group I-state graph in Figure 2.3. In the periodic group I-state graph, vertices (cells) in a periodic group form a cycle and vertices (cells) in a transient trajectory can terminate in one of two ways. It can terminate with its last vertex (cell) being either coincident with a cell (vertex) in the periodic group or coincident with a vertex (cell) in another transient trajectory which itself terminates in the aforesaid periodic group.

Complexity of Algorithm

The running time of Algorithm 2.1 is $O(N)$ where N is the total number of cells since its complexity is dominated by line 3, which iterates over all the cells, processing each cell exactly once.

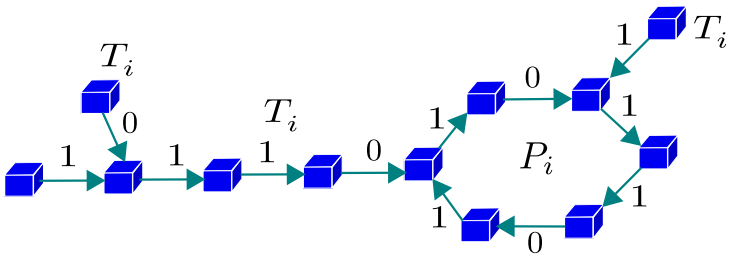


Figure 2.3: A periodic group I-state graph.

2.4.2 Creating Nondeterministic I-State Automaton

In the next step, we create a nondeterministic I-state automaton, A, amalgamating the entire set of periodic group I-state graphs $\mathcal{G}$ and define a nondeterministic I-state automaton [HH79] as follows:

Definition 2.4.2 (Nondeterministic I-State Automaton)
Let $A \triangleq (Q, \Sigma_\epsilon, \delta, q_0, F)$ be a nondeterministic automaton which accepts a stream of discrete observations from Y in which:

1. *$Q = \{q_0\} \cup \mathcal{G}.V$ is a finite set of states.*
2. *$\Sigma_\epsilon = Y \cup \{\epsilon\}$ is a finite alphabet where $Y = \{0, 1\}$.*
3. *δ is the state transition function for any $q \in Q$ and any input alphabet $a \in \Sigma_\epsilon$ as below:*

$$\delta(q, a) = \begin{cases} \{q'\} & q \in Q \setminus \{q_0\},\ a = \ell(q, q') \\ & \text{and } (q, q') \in \mathcal{G}.E, \\ Q \setminus \{q_0\} & q = q_0 \text{ and } a = \epsilon, \end{cases}$$

   and $|\delta(q_j, 0)| + |\delta(q_j, 1)| = 1, \forall j = 1, \ldots, N$.
4. *q_0 is the newly created initial state.*
5. *$F = \mathcal{G}.V_P$ is the set of final states that represents the set of vertices in periodic group I-state graphs.*

The states of A except the initial state q_0 are essentially the same states (or vertices) as the periodic group I-state graphs $\mathcal{G}$. The number of states of A becomes $N+1$. A nondeterministic I-state automaton using only the periodic group I-state graph of Figure 2.3 is illustrated in Figure 2.4.

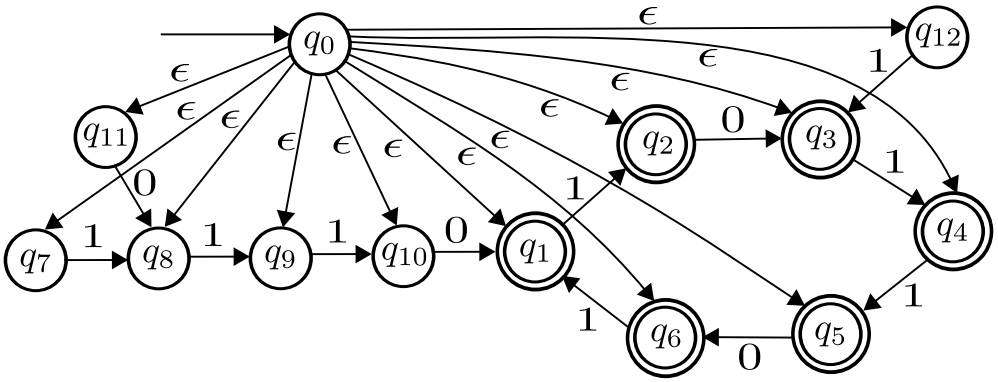


Figure 2.4: A nondeterministic I-state automaton.

2.4.3 Nondeterministic I-State Automaton to Deterministic I-State Automaton Conversion

Given the nondeterministic I-state automaton A, we construct a deterministic I-State automaton A', converting the ϵ-nondeterministic automaton into a deterministic one using lazy evaluation method as follows:

Definition 2.4.3 (Deterministic I-State Automaton)
Let $A' \triangleq (Q', Y, \delta', q_0', F')$ be a deterministic automaton that also accepts the stream of discrete observations from Y as [HH79] where:

1. *$Q' = \{S : S \subseteq Q$ and $S = \epsilon$-Closure$(S)\}$ where ϵ-Closure(S) is the set that contains S including all states reachable from any state in S following one or more ϵ-transitions.*

2. $Y = \{0, 1\}$.

3. $\delta'(S, a) = \bigcup \{\epsilon\text{-}Closure(p) : p \in \delta(s, a)$ *for some* $s \in S\}$.

4. $q'_0 = \epsilon\text{-}Closure(q_0)$.

5. $F' = \{S : S \in Q'$ *and* $S \cap F \neq \emptyset\}$.

All transitions that are not defined lead to the 'trap' state implicitly. The converted deterministic I-state automaton A' produces a directed graph in which the outdegree of each state is at most two and each state represents one or more vertices of the periodic group I-state graphs $\mathcal{G}$. The states of A' that represent vertices in the transient trajectories of $\mathcal{G}$ form a directed acyclic graph. The states that represent the last vertices of transient trajectories lead to simple cycles (e.g., closed paths where no vertices and edges are repeated) in A'. We use the term *knowledge cycles* for these cycles. The states in the knowledge cycles of A' represent set of the vertices of periodic groups, $\mathcal{G}.V_P$, of $\mathcal{G}$. These knowledge cycles act like attractors; once the robot reaches one via states in the transient trajectories, it cannot leave.

Proposition 2.4.4 *The number of states in the deterministic I-state automaton* A' *is* $O(N^2)$ *with respect to the number of states* N *in the nondeterministic I-state automaton* A.

Proof. The only non-determinism in A is the ϵ-transitions from the initial state to all other states. Moreover, there are no transitions in A that return back to the initial state. Every transition, except the initial one, is deterministic as there is at most one observation (either a '1' or a '0') from a state. There are no self-loops in A because the translation or the rotation of the robot changes the underlying state of the system. There are two parts in A': the first part consists of states that

represent set of states composed of both transient states and periodic states in A and the second part consists of states that represent set of periodic states in A. Let $N = N_t + N_p$ where N_t is the number of transients states in A, N_p is the number of periodic states in A. The states in the first part of A' form a full binary tree in the worst case because these states have two children, labeling two observations (0 and 1) on their transitions, and no child has more than one parent. In this part, the number of transient states decreases or remains same from the root to the leaves of the tree because applying the transition function δ' on the root q'_0 that represents Q, for two observations creates two disjoint sets Q_1 and Q_2 such that $|Q_1| + |Q_2| \leq |Q|$ and subsequent states follow this inequality. Thus, it follows by induction that the height of the binary tree is $O(\log(N_t))$ and the total number of states in the first part of A' is $O(2N_t - 1)$. This tree has at most N_t leaves that transition to knowledge cycles, which is the second part of A'. Hence, the second part of A' has at most N_t knowledge cycles; one cycle for each leaf. The length of each knowledge cycle is at most N_p because in the worst case the states in the cycle can include all periodic states N_p and each state represents one periodic state (or singleton) of N_p. Then, the total number of states in the second part of A' becomes $O(N_t N_p)$. Therefore, the number of states in A' is $O(N_t N_p) + O(2N_t - 1)$ or $O(N^2)$. □

2.4.4 Filters for the Closed- and Open-World Problems

The final step produces filters to solve the two localization problems formulated in Section 2.3. We follow the standard filter definition from [OS13]. The definitions of localization filters for closed and open world problems are as follows:

Definition 2.4.5 (Filter for the Closed-World) *A localization filter for the closed world problem is a tuple* $\mathcal{F}_C \triangleq (Q', Y, \delta', q'_0, c : Q' \to \mathbb{N})$*, where function* c *augments the deterministic I-state automaton by adding a color to its states.*

The filter $\mathcal{F}_C$ receives an observation string as input and reports a color as output. There are no final states in $\mathcal{F}_C$. Instead, we assign color 1 to every state that represents the transient trajectory vertices of $\mathcal{G}$. We assign the different color numbers to the states of different knowledge cycles ranging from 2 to one more than the number of cycles in A'. The same color number is assigned to every state of the same knowledge cycle. A filter for the closed world problem augmenting the deterministic I-state automaton is depicted in Figure 2.5. Here, the states in two knowledge cycles are assigned green and cyan colors, and the states that are not in knowledge cycles are assigned the white color.

The filter $\mathcal{F}_C$ is used for localization of a robot in the closed-world problem case. When the robot enters into the colored knowledge cycle, it looks up the state $q' \in Q'$. Each state q' in the knowledge cycle of A' represents a set of states in the A. The cardinality of this set of states in A determines the uncertainty level of robot's position for solving localization problem. These states of A are also indexed by cell numbers. As these cell numbers indicate the configurations of the robot in E, the robot localizes itself. Depending on the aforementioned number, the robot may localize itself in one or more configurations in E. As an example, in Figure 2.5, if the robot gets to a green knowledge cycle then it can localize up to a single configuration as it has an uncertainty of 1. On the other hand, if the robot gets to the cyan knowledge cycle then it can localize up to two configurations as it has an uncertainty of 2. Thus, this filter solves the localization problem, and because it is a deterministic automaton, captures all the state needed to localize explicitly.

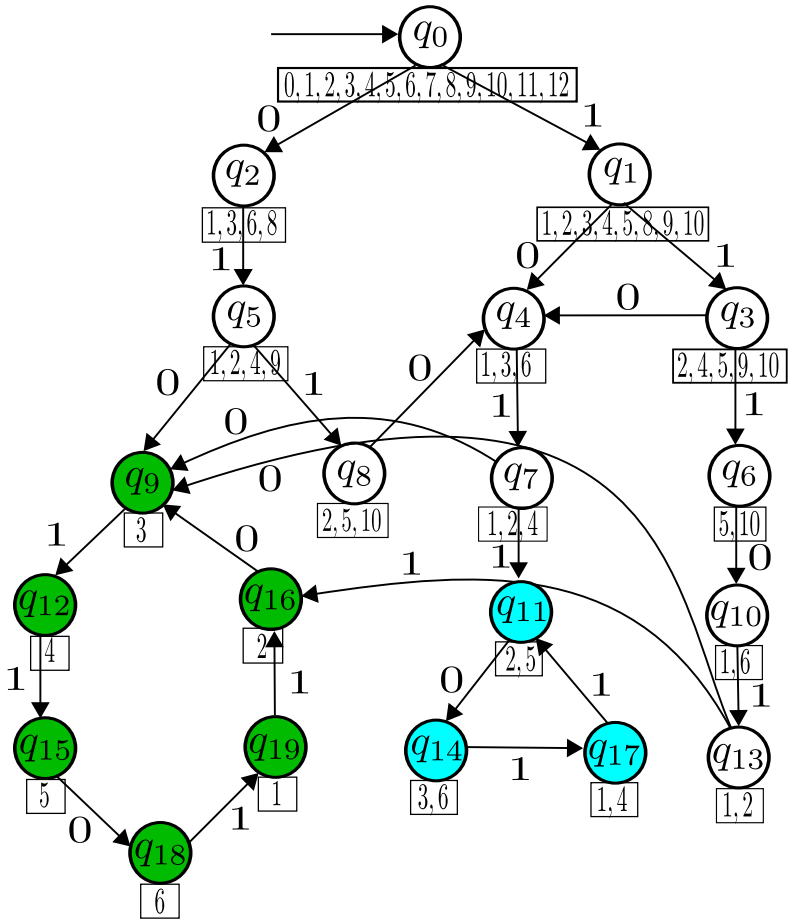


Figure 2.5: A filter for the closed-world localization problem.

Definition 2.4.6 (Filter for the Open-World)) *A localization filter for the open world problem is a tuple* $\mathcal{F}_O \triangleq (Q = Q' \cup \{q_t\}, Y, \delta', q'_0, c : Q \to \mathbb{N})$. *It augments the deterministic I-state automaton adding a "trap" state* q_t *along with assigning colors to all states.*

The filter $\mathcal{F}_O$ also receives an observation string as input and reports a color as output to indicate whether the robot is in E or not. In the filter $\mathcal{F}_O$, there is no state transition for some states on a specific observation symbol. From these states on the missing observation symbol, we add transitions to the "trap" state q_t. We assign a new color number to q_t. Aside from this, we do the same process as $\mathcal{F}_C$

for the construction of $\mathcal{F}_O$. The "trap" state acts as a reject state in $\mathcal{F}_O$. Once the robot observes an observation string and if the evaluation of the observation string using $\mathcal{F}_O$ takes it to q_t, the robot can report that it is not in the environment E. Otherwise, $\mathcal{F}_O$ gives an output color as $\mathcal{F}_C$ which solves the closed world localization problem.

If the robot needs to localize itself in one of the k environments, then $\mathcal{F}_O$ can solve this problem too. For example, the robot knows a set $\mathcal{E}$ of three possible environments $\{E_1, E_2, E_3\}$ and some bouncing angle ϕ. Following the above method, we create three filters $\mathcal{F}_O$ for three environments. Then, we run them in parallel inside the robot. The robot will be able to declare that it is in one of these environments or not because of the "trap" state in the $\mathcal{F}_O$.

2.5 Implementation

2.5.1 Simulation Results

We implemented the proposed modified simple cell mapping presented in Algorithm 2.1 in a simulation. The algorithm takes as an input the environment E and a bouncing angle ϕ and models the robot as a point.

We set the size of the environment E of Figure 2.6 to 200×125 grid unit lengths, excluding variable-size obstacles, $S^1 = [0, 2\pi)$. The cell size was set to 1 unit $\times$ 1 unit $\times$ 1°. We executed a simulation of Algorithm 2.1 and changing the obstacle region as follows:

- E_1: Randomly placing a square obstacle of fixed size inside the environment.
- E_2: Randomly placing a square and rectangular obstacles with fixed size inside the environment.

- E_3: Randomly placing a square, a rectangular, and a rectilinear obstacle with fixed size inside the environment.
- E_4: Randomly placing a scaled square obstacle inside the environment.

We ran the simulation of Algorithm 2.1 100 times for each of the four environments (E_1, E_2, E_3, E_4), keeping the bouncing angle $\phi = 90°$. We recorded the total number of periodic groups r and maximum transient trajectory length. Figure 2.7(a) and (b) illustrate the values of r and maximum transient trajectories lengths. From these results, we conclude that values of r increase with the addition of obstacles and change with the scaling of an obstacle and also that the maximum transient trajectory length varies with increasing numbers of obstacles and the modification of the size of an obstacle. Some outliers are present in the plot of maximum transient trajectory length in Figure 2.7(b) that are useful for the coverage problem [ABS17].

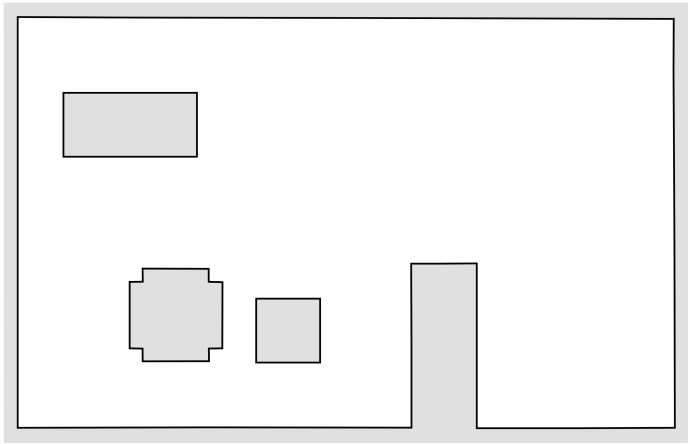


Figure 2.6: A simple environment with three randomly placed obstacles (completely interior) and one static obstacle (touching boundary).

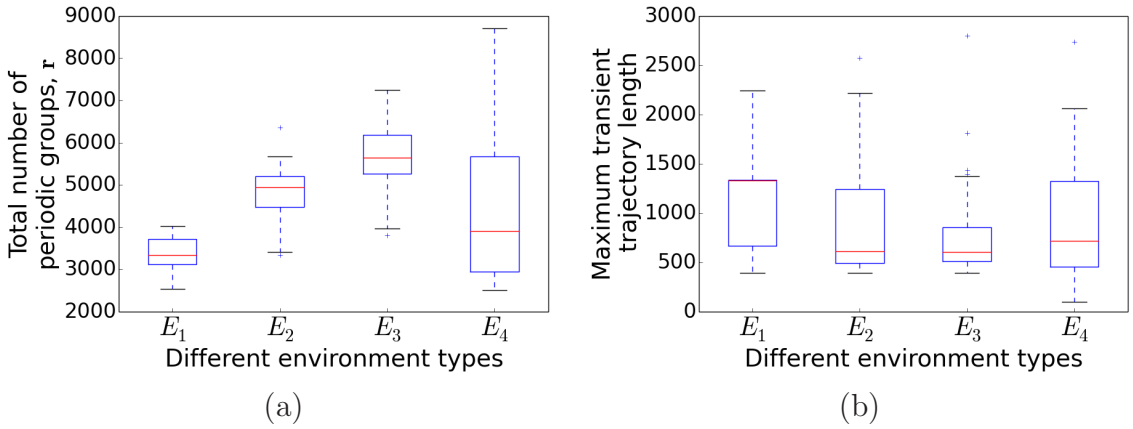
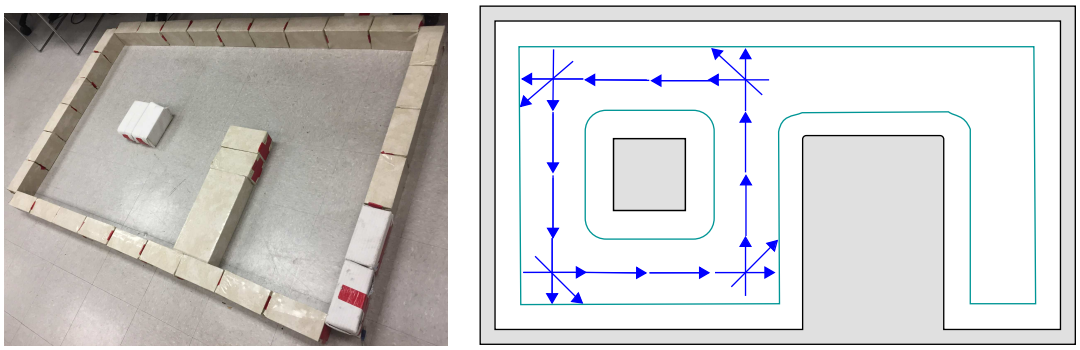


Figure 2.7: A comparison of simulations for different environment types: (a) the total number of the periodic groups r; (b) the length of the longest transient trajectory.

2.5.2 From Simulation to Physical Implementation

We tested our Algorithm 2.1 with a differential drive robot, the iRobot Create/Roomba, in two environments using the bouncing angles $\phi = 45°, 135°$. The Roomba is equipped with many sensors but we only use the bump sensors and the clock. Since the Roomba is a disk robot, rather than a point robot, we analytically calculate the free configuration space X_{free} of the robot for both environments. For both environments, the free space which is also the cell state space X_{free} is discretized in $N = 152$ cells having 19 cells in each of 8 different orientations of S^1 with 45° separation between each orientation.

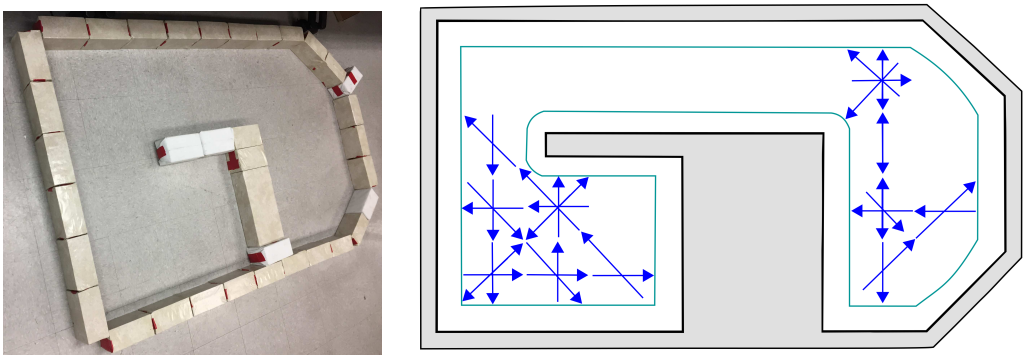
We ran our first simulation test on the X_{free} of the environment of Figure 2.8(a) using the bouncing angle $\phi = 45°$ and our second simulation test on the X_{free} of the environment of Figure 2.9(a) using the bouncing angle $\phi = 135°$. We found $r = 1$ periodic group including its corresponding transient trajectories from the first simulation test and $r = 2$ periodic groups along with their corresponding transient trajectories from the second simulation test. We visualize one periodic group of our



(a) (b)

Figure 2.8: (a) The first lab environment and (b) the simulation result showing the visualization of the periodic group for this environment and the bouncing angle $\phi = 45°$.

first simulation run in Figure 2.8(b) and rest of the configurations are the transient trajectories part of the illustrated periodic group. We also show two periodic groups of our second simulation run in Figure 2.9(b).



(a) (b)

Figure 2.9: (a) The second lab environment and (b) the simulation result showing the visualization of all periodic groups for this environment and the bouncing angle $\phi = 135°$.

From the periodic group and the transient trajectories of the first simulation run, we constructed $\mathcal{G} = \{G_1\}$, the set of periodic group I-state graph. Based on $\mathcal{G}$, we created the nondeterministic I-state automaton A as shown in Figure 2.10. In A,

we added a new initial state and ϵ-transitions to all other states from it and made the states in the periodic group as final states. We made use of JFLAP [RF15] to create A. The indices of the states of A except the newly added initial state are the cell numbers in X_{free}.

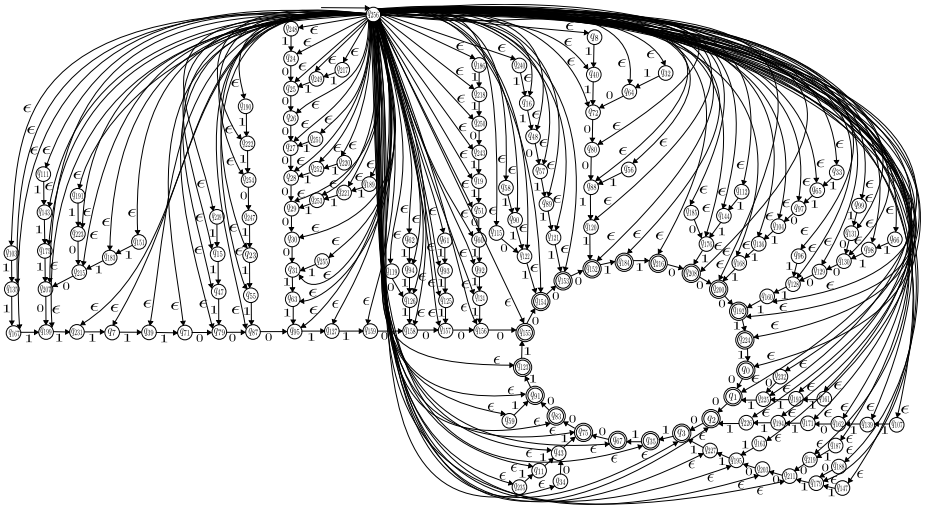


Figure 2.10: Created nondeterministic I-state automaton for the environment and the simulation result of Figure 2.8.

Again using JFLAP, we converted the nondeterministic automaton A into deterministic automaton A' as illustrated in Figure 2.11. This deterministic automaton has 3 knowledge cycles. One of the knowledge cycles has an uncertainty of 1, one of them has an uncertainty of 3, and one has an uncertainty of 4. We created the localization filter for solving Problem 1, adding 3 colors to the states of the deterministic automaton. We colored the states outside of knowledge cycles white and chose 3 distinct colors for the states of 3 knowledge cycles. Next, we produced a filter for solving Problem 2 by adding a new "trap" state to the previous filter and we assign 5 colors to it as a new color is required for the "trap" state.

We applied the same process to the periodic groups and the transient trajectories of the second simulation run and created the localization filters. We present the

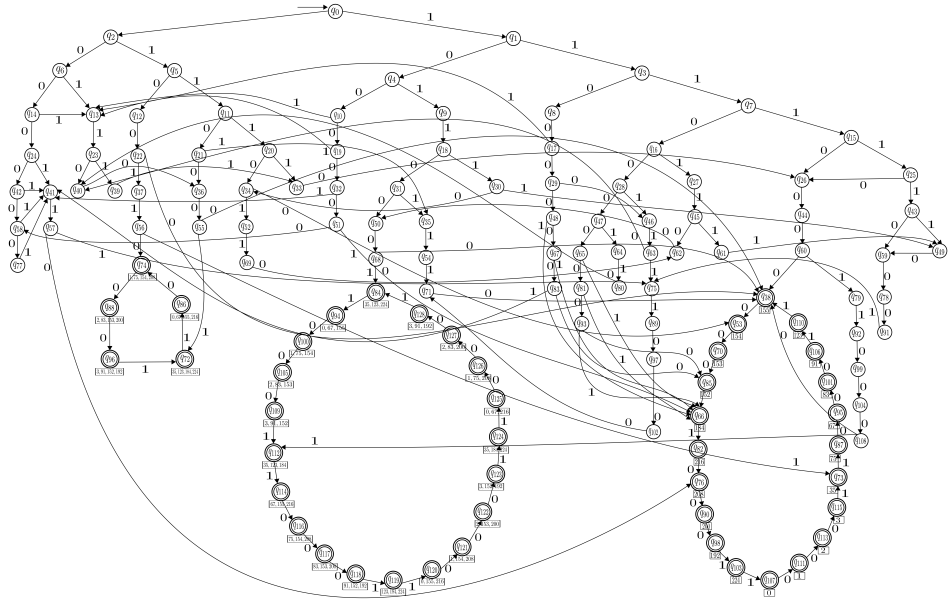


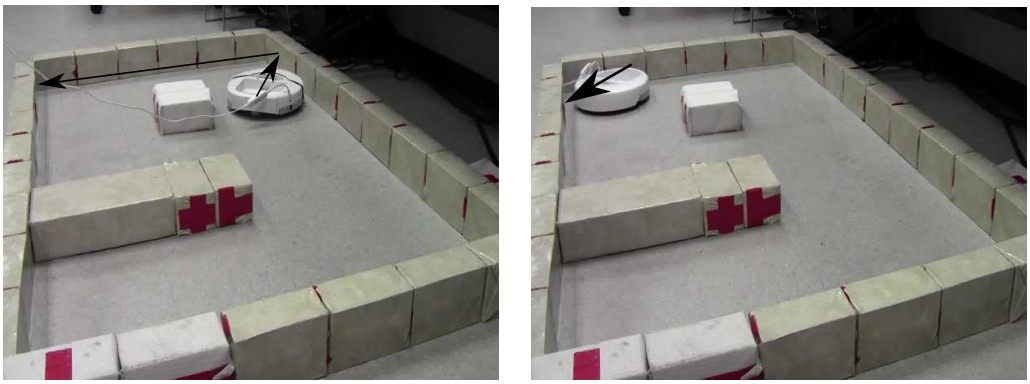
Figure 2.11: Converted deterministic I-state automaton from the nondeterministic I-state automaton of Figure 2.10.

empirical results of 1) the number of states in deterministic I-state automaton A after converting from nondeterministic I-state automaton A', and 2) the computation time for this conversion in Table 2.1. This conversion was performed on a GNU/Linux computer with Intel Core i7 3.6GHz processor and 16GB memory.

Table 2.1: No. of states and computation time comparison.

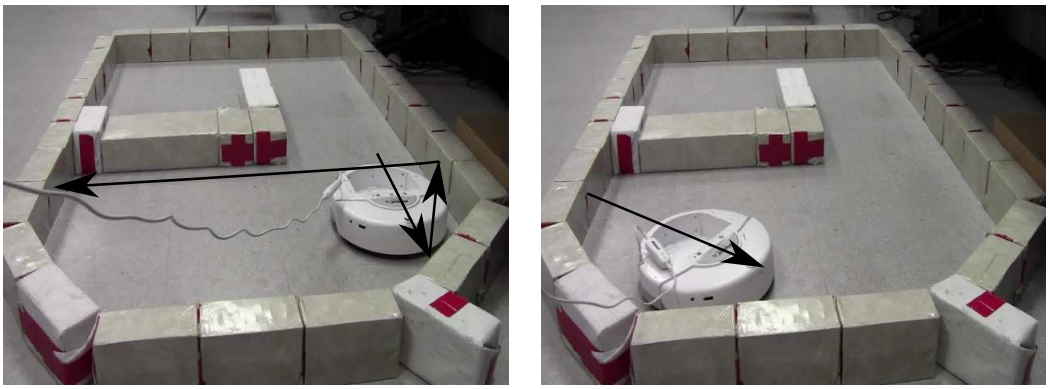
Input		No. of states of non-deterministic I-state automaton, $N+1$	No. of states of deterministic I-state automaton	Computation time (sec.)
E	ϕ			
E of Figure 2.8(a)	45°	153	129	24
	135°	153	124	20
E of Figure 2.9(a)	45°	153	147	27
	135°	153	202	71

We deployed the created localization filters on a Roomba and performed 10 physical experiments using the environments of Figure 2.8(a) and Figure 2.9(a),



(a) (b)

Figure 2.12: Physical localization experiment in the environment of Figure 2.8(a): (a) The robot was placed initially in the top left part of the environment; (b) after moving forward and bouncing with $\phi = 45°$, it was localized up to 3 configurations in the periodic group visualized in Figure 2.8(b).



(a) (b)

Figure 2.13: Physical localization experiment in the environment of Figure 2.9(a): (a) The robot was placed initially in the top right part of the environment; (b) after moving forward and bouncing with $\phi = 135°$, it was localized up to 1 configuration in one of the periodic groups visualized in Figure 2.9(b).

and the bouncing angles $\phi = 45°, 135°$. Two of them are illustrated in Figure 2.12 and Figure 2.13. In these experiments, the robot was localized and stopped once a knowledge cycle of the filter was reached starting from the initial state. Since all states in each cycle represent the same cardinality of the set of configurations, the maximum and the minimum number of configurations represented by the states

where the robot was able to localize, are tabulated in Table 2.2. Thus, the localization limits for an environment E and a bouncing angle ϕ are determined by the minimum and the maximum number of configurations of the robot, and the strongest possible localization is the minimum number of possible configurations of the robot.

Table 2.2: Comparison of no. of localization configurations.

Input		Number of localization configurations	
E	ϕ	Minimum	Maximum
E of Figure 2.8(a)	45°	1	4
	135°	1	2
E of Figure 2.9(a)	45°	1	2
	135°	1	1

2.6 Summary

In the first contribution, we presented a localization method for a robot equipped with a contact sensor and a clock. Our method is based on finding periodic groups and transient trajectories of the robot path as it bounces within an environment filled with obstacles. Based on the periodic groups and transient trajectories, space-efficient and automata-based combinatorial filters are synthesized to solve localization problems modulo symmetries. Experimental results from multiple simulations and from real robot demonstrations attest to the feasibility and practicability of our method.

In practice, the online computation time of our localization filter is the time required to evaluate an observation string, which is linear with respect to the length of the observation string only. The offline construction of the filter is linear in terms of the number of cells, and the conversion from the nondeterministic automaton

to the deterministic automaton is quadratic in terms of the number of cells. We adapted the comparison of different mobile robot localization methods from [TBF05] (see Section 8.5) by adding our combinatorial filter (CF) based localization method as illustrated in Table 2.3 to show the pros and cons of the proposed method. These localization methods use stronger robot sensing models with cameras and range sensors which make them more robust compared to our sensing model, having only the clock and contact sensors.

Table 2.3: Comparison of different localization methods.

	EKF	MHT	Coarse (topological) grid	Fine (metric) grid	MCL	CF (our method)
Measurements	Landmarks	Landmarks	Landmarks	Raw measure-ments	Raw measure-ments	Time and bump measure-ments
Measurement noise	Gaussian	Gaussian	Any	Any	Any	None
Posterior (on new observation)	Gaussian	Mixture of Gaussian	Histograms	Histograms	Particles	Single state
Efficiency (memory)	++	++	+	-	+	$O(\log N^2)$
Efficiency (time)	++	+	+	-	+	+++ [1]
Ease of implementation	+	-	+	-	++	++ [2]
Resolution	++	++	-	+	+	+ [3]
Robustness	-	+	+	++	++	- [4]
Global localization	No	No	Yes	Yes	Yes	Yes

[1] Our filter takes constant time for the sensor update on a new observation.

[2] Ease of implementation of our filter is the same as MCL.

[3] The resolution of our method is similar to the fine (metric) grid method.

[4] Our method is not robust to the noise or erroneous output.

CHAPTER 3

BOUNCING ROBOT NAVIGATION AND COVERAGE

This chapter provides the solutions to navigation and coverage tasks using single or multiple bouncing robots. The preliminary version of this work appeared in [ABS17].

3.1 Motivation and Challenges

The problem of navigation is finding a path between an initial pose and a goal pose. The coverage problem of the environment is visiting all locations of interest using one or more robots. These problems are important for many applications, such as search and rescue, surveillance, map generation, oil spill cleanup, vacuum cleaning, lawn-mowing, mine sweeping, exploration, automated farming, and painting. Moreover, multiple robot systems have the potential to improve (or speed-up) performance compared to single robots, especially in the coverage problem. The motivation of our work is to find solutions to two robotic problems: (i) navigation and (ii) coverage using one or more bouncing robots.

In both problems, single or multiple robots move in a known polygonal environment, executing an elementary behavior: the robots move straight until they discover walls by driving into them, or they collide with each other (in the multi-robot case) while covering the environment; then they turn (with respect to their current motion) counterclockwise by some angle. The motion is parameterized by a set of angles, which we term bouncing angles. This motion model enables the robots to navigate from one pose to another and cover the environment through the trajectories of such robots.

These motions can be executed by simple robots equipped with cheap sensors, and we are interested in solving navigation and offline coverage problems in known

environments, possibly with obstacles, for such robots. Although sensors are available with lower costs now, but the overuse of sensors requires powerful computation systems and abundant memory inside a sophisticated robot. Instead, we use a simple robot that can feed minimal sensor outputs directly to motors. This research also falls within the broader context of control and sensing with simple (or even minimal) robots. Several researchers have examined minimal sensing robots to solve several tasks such as localization [OL07, ABS18, SP12, EKOL08], navigation [ABS17, LO10, MSZ09], and mapping [TGL04]. These works, along with our own, eschew robots with an extensive sensory, computational, and memory capabilities, motivated both pragmatically—to reduce costs for the individual units, and theoretically—to explore sufficient conditions for the task performance. The approach in this work uses robots equipped with only a clock and contact sensors.

Furthermore, the navigation with limited linear and angular sensing is a challenging problem since the inadequate sensor information is available to a robot for executing the desired path. We also emphasize that the coverage of an area is hard as finding a path of optimal length for a given region is $\mathcal{NP}$-hard (via reduction to the Traveling Salesman Problem [Cho01]). Even the best zig-zag motion-based and boustrophedon motion-based [CP98] coverage solutions typically require robots to follow paths using feedbacks from powerful (and hence expensive) sensors.

In our first contribution, we use the SCM to synthesize combinatorial filters for solving the localization task [ABS18]. In the second contribution, we use the same SCM method for solving the navigation task using a single bouncing robot and extend this SCM method to tackle the coverage task in multi-robot settings. Additionally, we apply a nondeterministic dynamical system method called generalized cell-to-cell mapping (GCM) [Hsu80] to address the coverage problem for a single bouncing robot after its localization.

3.2 Related Work

3.2.1 Robot Navigation

Early works on landmark-based robot navigation include [LL95, RBFT99]. In these works, authors consider that the robot goes from one landmark to another with the explicit sensing of landmarks. However, in our work, we use the geometric description of the environment for navigation instead of the explicit sensing of landmarks. The dual-directional RFID antenna was proposed to enable autonomous navigation for mobile robots in indoor environments by obtaining a distance to a radio source [KC09]. Nonetheless, they need additional sensor data to be fused for enhancing the navigation capability of the robot in a cluttered environment.

More recent works on the robot navigation are belief roadmaps [PR09], randomized belief space trees [AmCA12], and feedback-based information roadmaps [Hau10] which consider the robot's uncertainty in its navigation. In our work, we have used a simpler robot model compared to those used in these other works. The works [LO10, LO12, LO13] use a robot model which is much closer to ours. In these works, authors use a robot equipped with a compass and contact sensors whereas we use a robot equipped with a clock and contact sensors. In their work, the robot can orient itself using the compass in the desired direction relative to a global reference frame which makes their robot stronger than our robot. Our robot instead follows a simple bouncing behavior to get to the goal configuration from its initial configuration. While they do not consider the weight of the navigation path, we minimize it.

3.2.2 Robot Coverage

The problem of coverage by a mobile robot has been investigated in different studies. In one survey on coverage path planning [GC13] studied where the approaches are evaluated based on whether they can be used online or offline and in the type of environments they can handle. In [WM03], an online topological coverage algorithm for mobile robots is presented that uses the detection of landmarks. Again, explicit sensing of landmarks is required so that the area of the environment will remain uncovered where no landmarks are available. Also, their method cannot find the critical points of concave landmarks as obstacles. In [GDS04], a coverage solution for mobile robots is presented which finds critical points of obstacles in unstructured environments and gives the entry and exit critical points for each obstacle.

In [VKS13], the authors propose fast coverage of the environment based on the unpredictable trajectory of a mobile robot with the use of a Logistic map and provide a chaotic random bit generator for a time-ordered succession of future robot locations. However, in their work, some parts of the environment stay uncovered. In an adversarial setting, the probabilistic method can optimally cover the environment and maximize the chances of detecting adversaries [AKK11]. Hence, in this work, we are interested in finding the optimal distribution of the bouncing policies used by a single robot to get our intended coverage of the environment or multiple robots cover the boundary of a given area.

Gabriely and Rimon [GR01] proposed a spanning-tree based coverage algorithm for a single robot. This work was extended for multi-robot coverage as a multi-robot spanning-tree based coverage algorithm in [AHK08, HK08], and as a multi-robot forest coverage algorithm in [ZJKK05]. These multi-robot coverage algorithms are either centralized or require reliable communication between robots and depend on extensive broadcast messages. The computational and memory complexities for

handling the sensor information of this kind of system are very high. Our bouncing robots do not communicate with each other for covering an area.

A recent offline spanning tree-based multi-robot coverage method presented by Fazli et al. [FDPM10] deals with the case where the robots have a limited visibility range. This approach is also shown to be complete and robust with regard to robot failure. In [FDM13], the authors designed the multi-robot repeated area coverage as the Multiple Traveling Salesman Problem and proposed three distributed cluster-based algorithms. Fazli and Mackworth [FM12] also proposed the repeated coverage of the boundaries of a target area and the structures inside it by multiple robots with limited visual and communication range. We do not use any visual sensing or communication medium in our robots. Instead, we address the coverage problem using the simple bouncing behavior of multiple robots.

3.3 Preliminaries

3.3.1 Robot Model

We consider a 2D polygonal *workspace* $\mathcal{W} = \mathbb{R}^2$, and a collection of static obstacles composing an *obstacle region*, $\mathcal{O} \subset \mathcal{W}$, where each element in $\mathcal{O} = \{\mathcal{O}_1, \mathcal{O}_2, \cdots, \mathcal{O}_k\}$ is modeled as a polygon. One or more differential drive mobile robots are modeled which share the workspace $\mathcal{W}$. Each robot is equipped with only a clock and contact sensors. The speed of each robot is fixed. The free workspace for each robot is denoted by the environment $E = \mathcal{W} \setminus \mathcal{O}$. Let $\partial E \subset E$ be the boundary of E. We consider that each robot knows the map of the environment and its initial configuration. If the initial configuration is unknown, the global robot localization can be solved using our first contribution [ABS18]. Let the set of bouncing angles

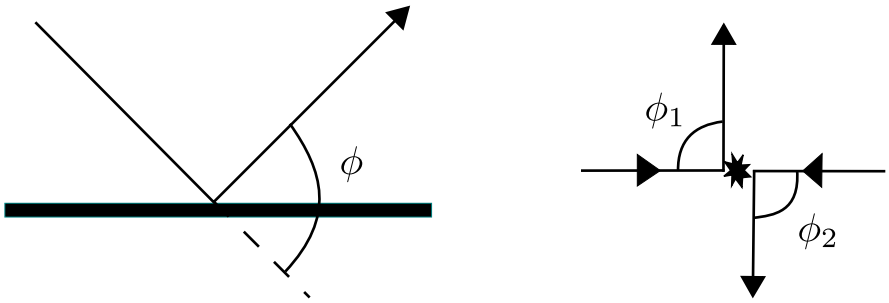


Figure 3.1: Simple bouncing strategy: (a) The robot rotates counterclockwise by the bouncing angle ϕ with respect to its current direction while it bounces off of the boundary of the environment; (b) When two robots collide with each other (only in the multi-robot coverage task), they then turn counterclockwise with their bouncing angles ϕ_1, ϕ_2 from their current moving orientations.

for a robot be Φ by which it can rotate reliably. In the multi-robot case, the set of bouncing angles for all robots is also denoted by Φ.

Each robot moves forward in the environment and records the number of *steps* using the clock. It continues this forward motion until it bumps into the boundary of the environment ∂E or collides with another robot (in solving the multi-robot coverage task only), as illustrated in Figure 3.1. Once a robot's contact sensors detect a bump, it rotates counterclockwise with a specified bouncing angle $\phi \in \Phi$ from its current orientation by commanding a constant angular velocity and using a clock to rotate for some fixed period. Thereafter, if it faces free space, it travels forward again and repeats this simple behavior.

3.3.2 System Model

We consider that there are m robots, $\mathcal{A}^1, \mathcal{A}^2, \cdots, \mathcal{A}^m$, ($m = 1$ in the case of a single robot). Each robot, $\mathcal{A}^i$, has its associated configuration space $X^i = E \times S^1$, where S^1 is the set of directions in the unit circle that represents the robot's orientations.

The configuration spaces of all robots have the same dimensions. Let $x^i \in X^i$ denote the configuration of each robot, in which $x^i = (x_t^i, y_t^i, \theta^i)$, where (x_t^i, y_t^i) is the corresponding robot's position and θ^i is its orientation. A physical state space is defined as the configurations of all robots simultaneously, $X = X^1 \times X^2 \times \cdots \times X^m$. A state $x \in X$ specifies all robot configurations.

There are two sources of obstacle regions in the state space: 1) *robot–obstacle* collisions and 2) *robot–robot* collisions [LaV06]. The robot-obstacle collision, $X_{\text{obs}}^i \in X$, for each robot $\mathcal{A}^i$, where $1 \leq i \leq m$, that collides with the obstacle region $\mathcal{O}$ is defined as:

$$X_{\text{obs}}^i = \{x \in X | \mathcal{A}^i(x^i) \cap \mathcal{O} \neq \emptyset\}. \tag{3.1}$$

The robot–robot collision, $X_{\text{obs}}^{ij} \in X$, between each pair of robots $\mathcal{A}^i$ and $\mathcal{A}^j$ is defined as:

$$X_{\text{obs}}^{ij} = \{x \in X | \mathcal{A}^i(x^i) \cap \mathcal{A}^j(x^j) \neq \emptyset\}. \tag{3.2}$$

Thus, the entire obstacle region in X is:

$$X_{\text{obs}} = \left(\bigcup_{i=1}^{m} X_{\text{obs}}^i \right) \bigcup \left(\bigcup_{ij, i \neq j} X_{\text{obs}}^{ij} \right). \tag{3.3}$$

The free state space $X_{\text{free}} = X \setminus X_{\text{obs}}$. Taking the configurations of all robots into account, each state $x \in X_{free}$ is a $3m$-dimensional vector and $x = (x_t^1, y_t^1, \theta^1, \cdots, x_t^m, y_t^m, \theta^m)$. Hence, we discretize the free state space X_{free} into N-dimensional cells. This discretized state space is called the *cell state space*. Let N be the total number of cells. For a single robot ($m = 1$), we define N as $N = N_E \times N_S$, where N_E is the discretization resolution of the 2-D free space E and N_S is the discretization resolution of S^1. The free state space of the system X_{free} is described by a cell index $z \in \{1, \ldots, N\}$. Let $Z = \{1, \ldots, N\}$ denote the collection of $3m$-dimensional cells.

The system dynamics can be explained as a series of cells by finding the system's state at discrete times. Let $e(k)$ be the cell describing the state of the system at

$t = k\Delta t$, $k = 0, 1, \ldots$ with Δt being the time between two state inspections. In the deterministic case, the system dynamics of the cell-to-cell mapping are described as

$$e(k+1) = \mathcal{C}(e(k)). \tag{3.4}$$

The above system evolution $\mathcal{C} : \mathbb{N} \to \mathbb{N}$ is called a simple cell-to-cell mapping (SCM) [Hsu13] in the cell state space. In this mapping, the next state of the system is dependent on only its current state instead of the mapping step k.

The definitions of the periodic cell, the transient cell, the periodic group, and the transient trajectory for the SCM method are explained in Section 2.3.2.

3.3.3 Uncertainty Model

We also consider the uncertainty in the angular motion in solving the coverage problem using a single bouncing robot. After bumping off the boundary of the environment, the robot rotates counterclockwise with a bouncing angle $\phi \in \Phi$ including an error range ϵ, from its current direction. We model this uncertainty of the angular motion using a nondeterministic cell-to-cell mapping method. In the nondeterministic case, the system dynamics of the cell-to-cell mapping are described as below:

$$p(n+1) = \mathcal{P}p(n) \text{ or } p(n) = \mathcal{P}^n p(0). \tag{3.5}$$

This dynamical system evolution is called the generalized cell-to-cell mapping (GCM) that creates finite Markov chains where $\mathcal{P}$ is the one-step transition probability matrix and $\mathcal{P}^n$ is the n-step transition probability matrix, $p(0)$ is the initial probability distribution vector over the cell configuration space, and $p(n)$ is the n-step probability distribution vector over the same configuration space. Let $p_{ij}^{(k)}$ denote the k-step transition probability from cell i to cell j and be (i,j)-th element of $\mathcal{P}^{(k)}$. If it is possible through the mapping to go from cell i to cell j, we say that

cell i leads to cell j, symbolically $i \Longrightarrow j$. Analytically, cell i leads to cell j if and only if there exists a positive integer k such that $p_{ij}^{(k)} > 0$. The cells i and j are said to communicate if and only if $i \Longrightarrow j$ and $j \Longrightarrow i$ which is denoted by $i \Longleftrightarrow j$.

We also define the following definitions of the GCM method [HX99, Hsu13]. More definitions and descriptions of cell-to-cell mapping methods can be found in [HX99, Hsu13, vcS94].

Definition 3.3.1 (Persistent Cell) *A cell z is called a persistent cell if it has the property that when the system is in z at a certain moment, it will return to z at some time in the future.*

Definition 3.3.2 (Probabilistic Transient Cell) *A cell that is not persistent is called a* probabilistic transient cell. *It leads to a persistent group in some number of steps.*

Definition 3.3.3 (Persistent Group) *A set of cells that is closed under the mapping is said to form a* persistent group *if and only if every cell in that set communicates with every other cell. Each cell belonging to a persistent group is called a persistent cell. A persistent group is also termed as an attractor or a limit cycle in the probabilistic case.*

3.3.4 Problem Formulation

We define a finite *action space* U, which is a set of all possible actions $u \in U$ using the sensors of a robot. The robot counts $\{0\}$ if it moves forward, and records the angle of rotation $\phi \in \Phi$ if it bumps and then rotates. Therefore, the action space is $U = \{0\} \cup \Phi$.

For finding a minimum navigation plan using a single robot, our robot measures the linear distance traversed by the number of *steps* up to some quantization error

based on the resolution and the clock time between two bumps. Thus, the robot executes a string of 0s as a stream of actions. During a bump event, the robot measures the bouncing angle ϕ using the clock time, resets the clock, and records the value of ϕ as an action. Therefore, the sequence of actions of our robot for the navigation, called the *action string* $\tilde{u}$, is encoded as a string of 0s interspersed by a value of ϕ.

Let $x_I \in X_{\text{free}}$ be the initial configuration of a single bouncing robot $\mathcal{A}$ ($m = 1$), and $x_G \in X_{\text{free}}$ be its desired goal configuration. We assume that $\mathcal{A}$ knows x_I and x_G and rotates reliably by bouncing angles Φ. In this context, a single query (x_I, x_G) navigation problem is formulated as follows:

Problem 1. Finding a minimum plan for a single robot:

Given an environment E, a set of bouncing angles Φ, an initial configuration x_I, and a goal configuration x_G, find the action string $\tilde{u}$ that represents the shortest path involving the minimum number of bouncing angle changes along the path, if one or more paths exist.

For answering multiple (x_I, x_G) navigation plan queries for the robot, we can extend the single query navigation plan problem by finding all possible shortest paths between the initial and goal configuration pairs using the given set of bouncing angles. As such, the multiple queries navigation problem is formulated as:

Problem 2. Generating all minimum plans for a single robot:

Given an environment E, a set of bouncing angles Φ, generate all action strings that represent all possible shortest paths for all (x_I, x_G) pairs in the X_{free}, involving the minimum number of bouncing angle changes along these paths.

In the coverage problem scenario for a single bouncing robot, each bouncing angle with an error range $\phi \pm \epsilon$ represents a separate bouncing policy for the robot. The robot has a target coverage probability distribution over E which is denoted by b. To combine the bouncing policies of the robot for covering an environment, the best solution is to find the optimal bouncing policy distribution of the robot based on the long-term robot's behavior resulting from the application of these policies. Let α be the bouncing policy distribution of the robot. Thus, the coverage problem for a single bouncing robot is formulated as:

Problem 3. Finding an optimal bouncing policy distribution for a single robot:

Given an environment E, a set of bouncing angles Φ, an error range ϵ, and a target coverage distribution b, find the optimal bouncing policy distribution α to get as close coverage as possible to b.

For solving the coverage problem using multiple bouncing robots, we assume that robots have their initial configurations. Let $x_0 \in X_{\text{free}}$ be the initial configurations of m robots where $m \geq 2$. The initial configuration of m robots is the initial cell from which we apply the cell-to-cell mapping. Then, the state of the system evolves over time and creates a trajectory for the robots. Let $T = [0, \infty)$ be a time interval of the execution of the system. We define a joint trajectory of m robots as $\tilde{x} : [0, T] \to X_{\text{free}}$ with $\tilde{x}(0) = x_0$ and where $\tilde{x}(t)$ represents the state of the system at time t. The joint trajectory of the robots will end up in a cyclic trajectory according to the properties of the system evolution of the simple cell-to-cell mapping [Hsu13].

Finally, we are interested in finding a joint trajectory of m robots that covers the environment. This motivates us to define the coverage problem for multiple robots as follows:

Problem 4. Finding a joint trajectory of multiple robots for coverage:

Given an environment E, a set of bouncing angles Φ for m robots, and an initial configuration of m robots x_0, find the joint trajectory $\tilde{x}$ of m robots for covering the given environment E.

3.4 Approach

In this section, we describe our approach for solving the problems formulated in Section 3.3 in detail.

3.4.1 Finding Roadmap and Minimum Navigation Plan for a Single Robot

Let a topological graph $\mathcal{G} = (V, E)$ be a weighted, directed graph and the weight function $w : E \to \mathbb{N}^+$ assign the nonnegative edge weight. Each vertex $v \in V$ represents a configuration (cell) $x \in X_{\text{free}}$ and each edge $(u, v) \in E$ where $u, v \in V$, represents a configuration transition from $x \in X_{\text{free}}$ to $x' \in X_{\text{free}}$. This topological graph is also called a *roadmap*. If any path exists between the initial configuration x_I and the goal configuration x_G on $\mathcal{G}$, then it is more likely to have one or more paths among (x_I, x_G) ordered pairs for the set of bouncing angles Φ. Let a shortest path of the robot $\mathcal{A}$ be $\tau : [0, 1] \to X_{\text{free}}$ such that $\tau(0) = x_I$, $\tau(1) = x_G$. After the robot's bump event, if the bouncing angle is preserved, the weight of the configuration transition is w_1 for the robot. Otherwise, if the bouncing angle changes, the weight of the configuration transition is w_2 for the robot.

In our approach, we modify the simple cell-to-cell mapping (SCM) to find the roadmap $\mathcal{G}$ and the minimum navigation plan between x_I and x_G in the cell configuration space X_{free}.

To attain these, Algorithm 3.1 receives as input the geometric description of the environment E, a set of bouncing angles Φ, the initial configuration x_I, and the goal configuration x_G. It produces the roadmap $\mathcal{G}$ and the action string $\tilde{u}$, which is the minimal navigation plan as output.

For each bouncing angle ϕ_i, where $i \in \{1, \ldots, |\Phi|\}$, Algorithm 3.1 computes the configuration of each cell $z \in Z$ that represents the location (centroid) and the orientation of the cell (line 6). The next mapped cell z' represents the subsequent cell after z (line 7) which is calculated as:

$$\begin{aligned} x_{z'} &= x_z + \frac{r}{2}(u_l + u_r)\cos\theta, \\ y_{z'} &= y_z + \frac{r}{2}(u_l + u_r)\sin\theta, \\ \theta' &= \begin{cases} \theta, & \text{if } (x_{z'}, y_{z'}) \in E, \\ (\theta + \phi) \mod 2\pi, & \text{otherwise}, \end{cases} \end{aligned} \tag{3.6}$$

where $(u_l, u_r) = (1, 1)$ specifies the left and right wheel velocities and $r = 1$ is the wheel radius of the robot.

If the new orientation of the cell θ' is equal to the previous orientation of the cell θ, then the cell number of z' is calculated from the new cell center location and previous orientation, $(x_{z'}, y_{z'}, \theta)$, of z' (line 9). Cells z, z' are added to vertices set and their ordered pair (z, z') is added to edges set of the directed graph $\mathcal{G}_i$ and weight of the corresponding edge is updated with w_1 on $\mathcal{G}_i$ (lines 10–12). Otherwise, Algorithm 3.1 calculates the set of possible bouncing cells with respect to the given bouncing angle set Φ. For the processing bouncing angle, we compute the next cell z' using the new cell center location and orientation of the cell $(x_{z'}, y_{z'}, \theta')$ (line

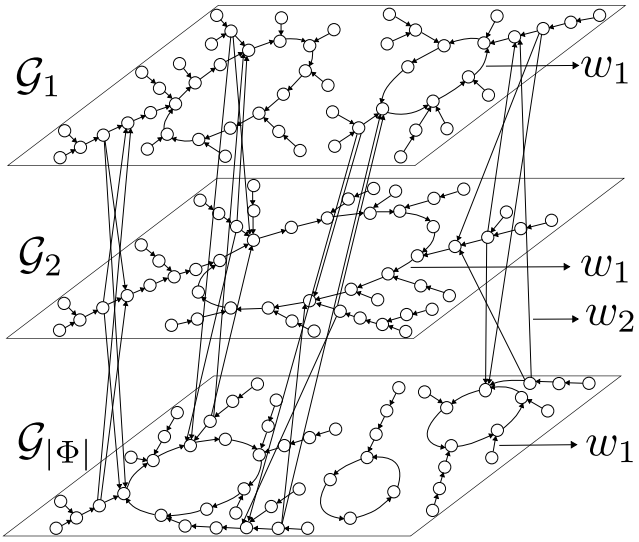


Figure 3.2: An example of a generated roadmap.

16). Both cells, their ordered pair as an edge, and weight of their edge are added to $\mathcal{G}_i$ as before (lines 17–19). For all other bouncing angles that represent the change of bouncing angles, we compute the next cell z' again from the previous cell center location using Equation 3.6, the previous orientation, and the bouncing angle (x_z, y_z, θ, ϕ) (line 21). Both cells and their ordered pair as an edge are added to $\mathcal{G}_i$ but the weight of their edge is updated with w_2 on $\mathcal{G}_i$ (lines 22–24).

For each bouncing angle, we get a set of periodic groups or limit cycles and associated transient trajectories leading to the periodic groups as the output of the SCM. We repeat the same process for all the bouncing angles in the given set Φ and create the roadmap $\mathcal{G}$ from the geometry as illustrated in Figure 3.2. In line 26, we use Dijkstra's shortest path algorithm [CLRS01] to find the shortest path τ on the roadmap $\mathcal{G}$ from all ordered pairs of initial configuration x_I and goal configuration x_G for different bouncing angles in Φ. The function NavigationPlan finally returns the action string $\tilde{u}$ based on the shortest path τ (line 27). In this function, if the consecutive cell distance in the shortest path τ is less than N_E, it implies "forward movement" and gives '0' as one action. Otherwise, it implies the "bump" event and

Algorithm 3.1: RoadmapandPlan$(E, \Phi, w_1, w_2, x_I, x_G)$

Input: $E, \Phi, w_1, w_2, x_I, x_G$ – Environment, Set of bouncing angles, Weights, Initial configuration, and Goal configuration

Output: $\mathcal{G}, \tilde{u}$ – Roadmap, Action String

```
1  G ← ∅
2  for i = 1 to |Φ| do
3      G_i.V ← ∅,   G_i.E ← ∅
4      for j = 1 to N do
5          z ← j
6          x_z, y_z, θ ← CellConfiguration(z)
7          x_z', y_z', θ' ← NextCell(x_z, y_z, θ, φ_i)
8          if θ == θ' then
9              z' ← Cellnumber(x_z', y_z', θ)
10             G_i.V ← G_i.V ∪ {z, z'}
11             G_i.E ← G_i.E ∪ {(z, z')}
12             w(z, z') ← w_1
13         else
14             for k = 1 to |Φ| do
15                 if k == i then
16                     z' ← Cellnumber(x_z', y_z', θ')
17                     G_i.V ← G_i.V ∪ {z, z'}
18                     G_i.E ← G_i.E ∪ {(z, z')}
19                     w(z, z') ← w_1
20                 else
21                     z' ←OtherCell(x_z, y_z, θ, φ_k)
22                     G_i.V ← G_i.V ∪ {z, z'}
23                     G_i.E ← G_i.E ∪ {(z, z')}
24                     w(z, z') ← w_2
25     G ← G ∪ G_i
26 τ ← ShortestPath(G, x_I, x_G)
27 ũ ← NavigationPlan(τ)
28 return G, ũ
```

gives the bouncing angle ϕ as another possible action. This action string $\tilde{u}$ provides the solution of a single (x_I, x_G) navigation query.

3.4.2 Generating All Minimum Navigation Plans for a Single Robot

We generate all minimum navigation plans from all possible shortest paths among all (x_I, x_G) pairs in the cell configuration space X_{free}.

To obtain all possible shortest paths, Algorithm 3.2 takes the roadmap $\mathcal{G}$ constructed from Algorithm 3.1 as input and generates all minimum navigation plans M from their shortest paths if one or more paths exist among ordered (x_I, x_G) pairs and their path weights L as output.

Algorithm 3.2: AllPlanGeneration($\mathcal{G}$)

```
   Input: G – Roadmap
   Output: M, L – Navigation Plans, Path Weights
 1 let M[1...N, 1...N], L[1...N, 1...N] be 2D lists
 2 for i = 1 to N do
 3     for i = 1 to N do
 4         M[i][j] ← NIL
 5         L[i][j] ← 0
 6 for i = 1 to N do
 7     for j = 1 to N do
 8         if i ≠ j then
 9             τ, l ← ShortestPathandWeight(G, i, j)
10             if τ ≠ NIL then
11                 L[i][j] ← −1
12             else
13                 ũ ← NavigationPlan(τ)
14                 M[i][j] ← ũ
15                 L[i][j] ← l
16 return M, L
```

Algorithm 3.2 initializes M and L lists (lines 2–5). From $x_i \in X_{\text{free}}$ on the roadmap $\mathcal{G}$ to all other $x_j \in X_{\text{free}}$ on $\mathcal{G}$, we run Dijkstra's algorithm to find the shortest path τ and minimum weight l among the (x_i, x_j) ordered pairs for the set of bouncing angles Φ on $\mathcal{G}$ (line 9). If τ is none, which means that there is no path among the (x_i, x_j) pairs, Algorithm 3.2 assigns -1 to the (i, j)-th entry of L (lines 10–11). Otherwise, it encodes the shortest path τ into an action string $\tilde{u}$ as a minimum navigation plan (line 13) and then assigns the plan $\tilde{u}$ and minimum path weight l to the (i, j)-th entry of M and L respectively (lines 14–15). We find minimum navigation plans and path weights for all $x \in X_{\text{free}}$. Finally, Algorithm 3.2 returns M and L lists. These minimum navigation plans and path weights can then be used to answer multiple (x_I, x_G) navigation plan queries and their comparison.

Algorithm Analysis

The running time of the Algorithm 3.2 is $O(N^2 \log N)$ since it applies Dijkstra's shortest path algorithm to all (x_I, x_G) pairs for each $x \in X_{\text{free}}$.

3.4.3 Finding Bouncing Policy Distribution for a Single Robot

We combine all bouncing policies represented by the set of bouncing angles Φ for the given environment E to get the closest coverage to a target coverage distribution b over E. Let the probability of reliable rotation of the robot be r. We apply the generalized cell-to-cell mapping (GCM) that uses a bouncing angle set Φ, a probability of reliable rotation r, and a nonzero error range ϵ. This method finds a number of persistent groups starting from all transient cells for each bouncing angle with an error range $\phi \pm \epsilon$. Since the persistent groups are the long-term behavior of the GCM, we consider the coverage distribution of these persistent groups of a

bouncing policy as the coverage of the environment by that bouncing policy. So, the transient cells are not considered for the coverage of the environment. A persistent group is an irreducible Markov chain as all its cells form a single communicating class. So, all persistent groups for a bouncing policy create a finite Markov chain $\mathcal{P}$. The limiting distribution π of $\mathcal{P}$ represents the coverage of E for each bouncing policy. First, we find the limiting distribution set Π for all the bouncing policies and then, using Π, we compute the bouncing policy distribution α of all bouncing policies through optimization.

Algorithm 3.3: PolicyDistribution(E, Φ, ϵ, r)

Input: E, Φ, ϵ, r – Environment, Set of bouncing angles, Error range, and Probability of reliable rotation

Output: $\Pi = \{\pi_1, \pi_2, \cdots, \pi_{|\Phi|}\}$ – Set of limiting distributions

```
for i = 1 to |Φ| do
    G.V ← ∅,   G.E ← ∅
    for j = 1 to N do
        z ← j
        x_z, y_z, θ ← CellConfiguration(z)
        x_z', y_z', θ' ← NextCell(x_z, y_z, θ, φ_i)
        if θ == θ' then
            z' ← Cellnumber(x_z', y_z', θ)
            G.V ← G.V ∪ {z, z'}
            G.E ← G.E ∪ {(z, z')}
        else
            Z' ← CellSet(x_z', y_z', θ' ± ε)
            G.V ← G.V ∪ Z' ∪ {z}
            G.E ← G.E ∪ {(z, z'), z' ∈ Z'}
    S ← StronglyConnectedComponent(G)
    T ← TransitiveClosure(G)
    P ← MCFromPersistentGroup(S, T, r)
    π_i ← NormalizedLimitingDistribution(P)
    Π ← Π ∪ π_i
return Π
```

In order to obtain the limiting distribution set Π, Algorithm 3.3 takes as input the geometric description of the environment E, the set of bouncing angles Φ, the error range ϵ, the probability of reliable rotation r. It returns Π as output.

In Algorithm 3.3, for each bouncing angle with error range $\phi \pm \epsilon$ from the set of bouncing angle Φ, we create an unweighted directed graph G following the same graph creation procedure of Algorithm 3.1 without adding weight to the edges of G. Additionally, for each cell $z \in Z$ when the new orientation of the cell θ' is not equal to the previous orientation of the cell θ, Algorithm 3.3 calculates the set of possible bouncing cells Z' where $Z' \subset Z$, using the new orientation of the cell with error range $\theta' \pm \epsilon$ (line 12). All cells z, Z' are added to the vertices set and their ordered pairs (z, z'), where $z' \in Z'$, are added the edges set of G for the processing bouncing angle (lines 13–14).

Then, it finds the strongly connected component S from G using Tarjan's strongly connected component algorithm [Tar72] (line 15). It also constructs the reachability matrix T, finding the transitive closure from G (line 16). From S and T, Algorithm 3.3 finds persistent groups using the function MCFromPersistentGroup (line 17). In this function, if each vertex in a strongly connected component is reachable from all other vertices in the strongly connected component then this strongly connected component is found as a persistent group and each cell of this persistent group is classified as a persistent cell. If a vertex in a strongly connected component is reachable from a subset of vertices in the strongly connected component then each cell of this strongly connected component is classified as a transient cell.

Further, in MCFromPersistentGroup function, an adjacency list is created from all persistent groups of the processing bouncing angle ϕ. Based on this adjacency list and reliable rotation probability r, the function creates the one-step transition probability matrix $\mathcal{P}$. To obtain this, the function uses the probability

$p_{ij} = r$ for reliable rotation from cell z_i to cell z_j, $p_{ij} = \frac{(1-r)}{2\epsilon}$ for unreliable rotation from cell z_i to cell z_j, and $p_{ij} = 1$ for forward movement from cell z_i to cell z_j. In the last step, it calculates the limiting distribution π of $\mathcal{P}$ for the processing bouncing angle with the error range $\phi \pm \epsilon$, normalizes π, and adds it to Π (lines 18–19). Finally, Algorithm 3.3 returns Π for all bouncing policies.

The probability distribution of choosing the bouncing policies for the robot can be represented as a k-dimensional vector where $k = |\Phi|$,

$$\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_k). \tag{3.7}$$

Equation 3.7 should satisfy: 1) $\alpha_i \geq 0$ for all $i \in \{1, \ldots, k\}$, and 2) $\alpha_1 + \alpha_2 + \cdots + \alpha_k = 1$. The value α_i is the proportion of the robot choosing the i-th bouncing policy.

We obtain the optimal bouncing policy distribution α from the limiting distribution set Π to achieve as close coverage as possible to the target coverage distribution b. We create the matrix A based on Π. We use the constrained least square [Gus11] to compute the optimal bouncing policy distribution α which is given by the following optimization equation:

$$\begin{aligned} &\text{minimize} && \|A\alpha - b\|^2 \\ &\text{subject to} && C\alpha = d, \alpha \geq 0. \end{aligned} \tag{3.8}$$

Here A is an $n \times k$ matrix, b is the n-vector where $n = N$, α is the k-vector, C is a $1 \times k$ matrix.

The bouncing policy distribution α is optimal for obtaining the coverage closest to the target coverage distribution b because it minimizes the norm of the residual error $\|A\alpha - b\|$ having the constraints of Equation 3.8. This bouncing policy distribution α states the time-based switching among the bouncing policies to cover the environment.

3.4.4 Finding Joint Trajectory of Multiple Robots for Coverage

In our approach, we find a joint trajectory $\tilde{x}$ of m robots starting from the initial configuration (cell) x_0 to cover the environment E. The m robots use their respective bouncing angles from the given set of bouncing angles Φ once they collide with each other, or against the boundary of the environment ∂E. In this case, we extend the SCM method in the high dimensional state space of multiple robots.

Algorithm 3.4: MultiRobotCoverage (E, x_0, Φ)

Input: E, x_0, Φ – Environment, Initial configuration, and Set of bouncing angles
Output: $\tilde{x}$ – Joint trajectory

```
1  pc ← ∅
2  b ← FindCellNumber(x_0)
3  x̃.init(b)
4  for l =1 to F do
5      k ← 1                          // Cell sequence generation
6      while pc_b ≠ ∅ do                          // Check unprocessed cell
7          pc_b ← 1                  // Identify processed cell
8          z ← CellConfiguration(b)
9          z' ← MappedCellConfiguration(z, Φ)
10         c_b ← NextCellNumber(z')
11         x̃.add_cell(c_b)      x̃.cell_transition(b, c_b)
12         b ← c_b      z ← z'      k ← k + 1
13     if CompleteCoverage(E, x̃) then
14         break
15     else
16         β_l ← BestUnprocessedNeighborCell(b)
17         x̃.add_cell(β_l)      x̃.cell_transition(b, β_l)
18         b ← β_l
19 return x̃
```

To find the joint trajectory $\tilde{x}$, Algorithm 3.4 takes as input the environment E, the initial configuration x_0, and the set of bouncing angles Φ for m robots,

and generates as output the joint trajectory $\tilde{x}$ that the robots follow to cover the environment.

In Algorithm 3.4, all cells are initially *unprocessed cells*. First, a cell b is calculated from the known initial configuration x_0 and added to the trajectory $\tilde{x}$ as the initial cell (lines 2–3). Then, we apply the SCM to create a cell sequence that is the part of the trajectory. For l iterations, the cell sequence is iteratively generated from the initial cell b in the high dimensional state space as explained in Algorithm 3.1. In lines 6–12, during the cell sequence generation, all cells are identified as *processed cells* by assigning the value 1 to these cells. The cell configuration z is calculated from the cell b, which represents the states of all m robots $x = (x_z^1, y_z^1, \theta_z^1, \cdots, x_z^m, y_z^m, \theta_z^m)$ (line 8). In line 9, the next mapped cell configuration of m robots z' after z is calculated as below:

$$\begin{aligned} x_{z'}^i &= x_z^i + \cos\theta_z^i, \\ y_{z'}^i &= y_z^i + \sin\theta_z^i, \\ \theta_{z'}^i &= \begin{cases} \theta_z^i, & \text{if } (x_{z'}^i, y_{z'}^i) \in X_{\text{free}}, \\ (\theta_z^i + \phi_z^i) \mod 2\pi, & \text{otherwise.} \end{cases} \end{aligned} \tag{3.9}$$

where $i \in \{1, \ldots, m\}$. The calculation of the next mapped cell configuration z' takes robot-obstacle collisions and robot-robot collisions into account.

This cell-to-cell mapping also finds the next mapped cell of each of the m robots. The cell number of the next mapped cell, c_b is computed from the center location of each robot and their orientations $(x_{z'}^1, y_{z'}^1, \theta_{z'}^1, \cdots, x_{z'}^m, y_{z'}^m, \theta_{z'}^m)$ of z' (line 10). We add the next cell c_b to the trajectory $\tilde{x}$ as a new cell and the cell transition from b to c_b is made in the trajectory $\tilde{x}$ as a connection (line 11). The next cell b is updated with c_b and the cell configuration z is updated with z' (line 12). The cell sequence generation continues as long as the next mapped cell b is an unprocessed

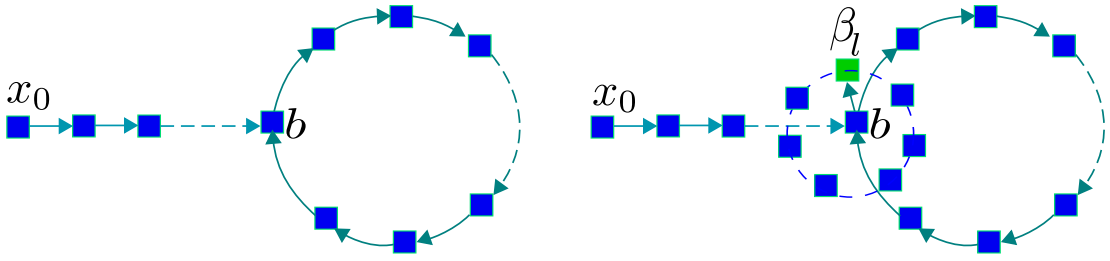


Figure 3.3: Trajectory generation and neighbor selection: a) a generated trajectory $\tilde{x}$ from an initial configuration x_I; b) the best nearest neighbor β_l (green square) among unprocessed neighbors of last cell b of the trajectory $\tilde{x}$.

cell in each iteration. A generated trajectory $\tilde{x}$ from an initial configuration x_0 as an initial cell is illustrated symbolically in Figure 3.3(a) where each square represents a $3m$-dimensional cell of the cell state space X_{free}. Some cells of $\tilde{x}$ form a cycle and others lead to this cycle.

Let β_l be the best unprocessed and nearest neighbor cell of the last cell of the trajectory $\tilde{x}$ at the l-th iteration of the algorithm. After each iteration, we check the coverage of the environment by the joint trajectory $\tilde{x}$ of m robots using the function CompleteCoverage. If the trajectory covers the whole environment then we stop the generation of the joint trajectory $\tilde{x}$. Otherwise, we select the best unprocessed and nearest neighbor cell β_l of the last cell b of the trajectory $\tilde{x}$ using the function BestUnprocessedNeighborCell. In the function BestUnprocessedNeighborCell, we take the neighbors of cell b by rotating m robots in both directions and keeping the locations of the robots fixed. Let

$$\Delta x_i = (0, \cdots, 0, \Delta\theta, 0, \cdots, 0), \tag{3.10}$$

in which the first $3i-1$ components and the last $3m-3i$ components are 0, and $\Delta\theta$ is the discretization resolution of the S^1.

The *neighborhood* of cell b in the state space is defined as

$$\mathcal{N}(x_b) = \{x_b + \Delta x_1, \cdots, x_b + \Delta x_{3m}, x_b - \Delta x_1, \cdots, x_b - \Delta x_{3m}\}. \tag{3.11}$$

We find at most $2 \times 3m = 6m$ neighbors in the state space at the l-th iteration. We simulate the cell sequences from these neighbors if the neighbors are not processed yet. We select the best nearest neighbor β_l based on the covered space of the environment and the minimal number of robot-robot collisions. The collection of unprocessed and neighbor cells of the cell b (the blue squares) and the selected best nearest neighbor β_l (the green square) are shown around the small circle of the cell b in Figure 3.3(b). The newly selected neighbor cell β_l is added to the trajectory $\tilde{x}$ as a new cell and the cell transition from b to β_l is made in the trajectory $\tilde{x}$ (line 17). The new initial cell b is updated with β_l (line 18).

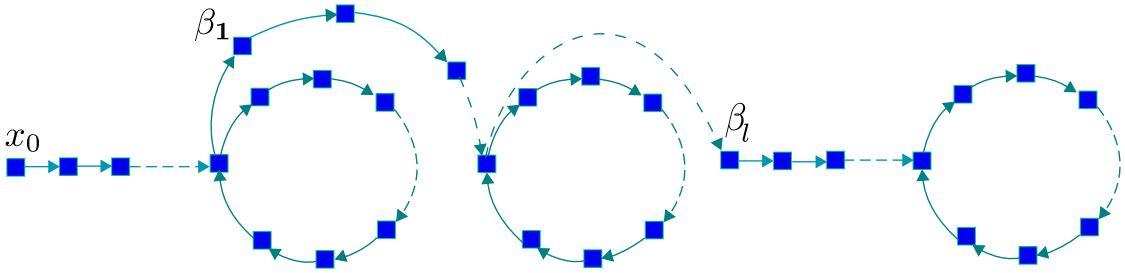


Figure 3.4: A joint trajectory of the robots connecting through the new neighboring cell β_l.

A pictorial representation of the generated joint trajectory of m robots from Algorithm 3.4 after F iterations is depicted in Figure 3.4, where each square also represents a $3m$-dimensional cell of the cell state space X_{free}. For the best neighbor cell β_l as an initial cell, the cell sequence forms the different part of the trajectory. The given initial configuration x_I provides the starting cell of the trajectory $\tilde{x}$ and creates the first part of $\tilde{x}$. After the first part of $\tilde{x}$, each subsequent part of $\tilde{x}$ is connected through the best neighbor cell β_l. This generated joint trajectory $\tilde{x}$ of m robots covers the environment completely or as much as possible.

Algorithm Analysis

The runtime of Algorithm 3.4 is $O(FN)$, where F is the number of iterations and N is the number of cells in the cell state space X_{free}.

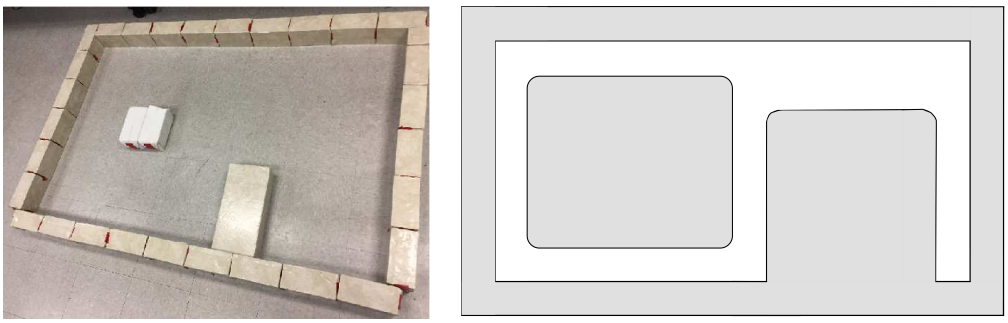
3.5 Experimental Results

3.5.1 Minimal Navigation Plan Result for a Single Robot

We tested Algorithm 3.1 for a single robot by developing a simulation and deploying it on a physical robot platform in the hardware experiment. We used the iRobot Create Roomba as a differential drive robot in an artificial laboratory environment of Figure 3.5(a). The Roomba has many sensors but we utilized only the bump and clock sensors. In the simulation, the configuration space of the laboratory environment is computed analytically for the disk robot Roomba, as illustrated in Figure 3.5(b). The cell configuration space X_{free} is discretized into $N = 384$ cells. In the simulation and experiment of navigation plans, we considered 8 different orientations of S^1 with 45° separation between each orientation.

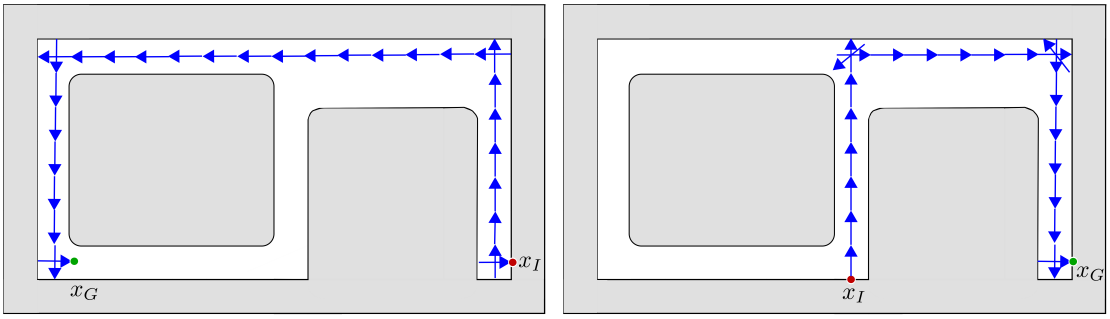
We ran our simulation for the above discretized cell configuration space X_{free} using the set of bouncing angles $\Phi = \{45°, 90°, 135°\}$ for the robot. We set the weight of using the same bouncing angle, $w_1 = 1$ and the weight of changing the bouncing angle, $w_2 = 100$ in our simulation. The illustrations of the first navigation plan between $x_I = 9$ and $x_G = 1$ and the second navigation plan between $x_I = 104$ and $x_G = 9$ are shown in Figure 3.6 with blue arrows. In Figure 3.6(a), the first navigation plan of the robot uses only one bouncing angle, 90°, to navigate from the bottom right corner of E, facing East, to the bottom left corner of E, facing East. In Figure 3.6(b), the second navigation plan of the robot uses two bouncing angles, 90° and 135°, to nav-

igate from the bottom left corner of the obstacle that is touching ∂E, facing North to the bottom right corner of E facing East. Our simulation gave two action strings as output for two navigation plans; $\tilde{u}_1 = 00000090^\circ 000000000000090^\circ 00000090^\circ$ and $\tilde{u}_2 = 000000135^\circ 135^\circ 000000135^\circ 135^\circ 00000090^\circ$.



(a) (b)

Figure 3.5: A laboratory environment: (a) an environment using floor and bricks that includes one completely interior obstacle and one obstacle touching the boundary of the environment; (b) the configuration space of the environment shown in (a).



(a) (b)

Figure 3.6: Simulation results of two navigation plans in the environment of Figure 3.5: a) the blue arrowed path from the initial configuration (bottom right corner of E, facing East) to the goal configuration (bottom left corner of E, facing East). b) the blue arrowed path from the initial configuration (bottom left corner of the obstacle attached to ∂E, facing North) to the goal configuration (bottom right corner of E, facing East).

Afterward, we deployed the two generated action strings $\tilde{u}_1$ and $\tilde{u}_2$ on the Roomba to navigate in the environment depicted in Figure 3.5(a). A netbook processes the sensor input from the robot and provides the output to the robot. We show snapshots of the two hardware experiments of the corresponding simulated navigation plans in Figure 3.7 and 3.8. In our first hardware experiment of Figure 3.7, we placed the Roomba in $x_I = 9$ and it followed the action string $\tilde{u}_1$ to get to $x_G = 1$. In our second hardware experiment (Figure 3.8), we also put the Roomba in $x_I = 104$ and it successfully reached to $x_G = 9$. In these hardware experiments, the Roomba uses its clock to measure the number of zeros as it moves forward and the duration of rotation for different bouncing angles. It also uses bump sensors for detecting the "bump event".

To test Algorithm 3.1 in a more complex environment, the configuration space, as illustrated in Figure 3.9, is discretized into $N = 1464$ cells considering 8 different directions of S^1 that are 45° apart of each other. For two pairs of initial and goal configurations in the given cell configuration space, Algorithm 3.1 found two navigation plans using the same set of bouncing angles $\Phi = \{45^\circ, 90^\circ, 135^\circ\}$. In Figure 3.9(a), the first navigation plan of the robot uses two bouncing angles, 45° and 135°, to get to the goal configuration x_G from the initial configuration x_I where locations of x_G and x_I are illustrated with red and green circles respectively. They face Eastward, and the navigation path is depicted with blue arrows. In Figure 3.9(b), the second navigation plan of the robot uses all bouncing angles, 45°,90°, and 135°, to complete its navigation task from x_I to x_G where the navigation path and its initial and goal configuration are illustrated the same way as before.

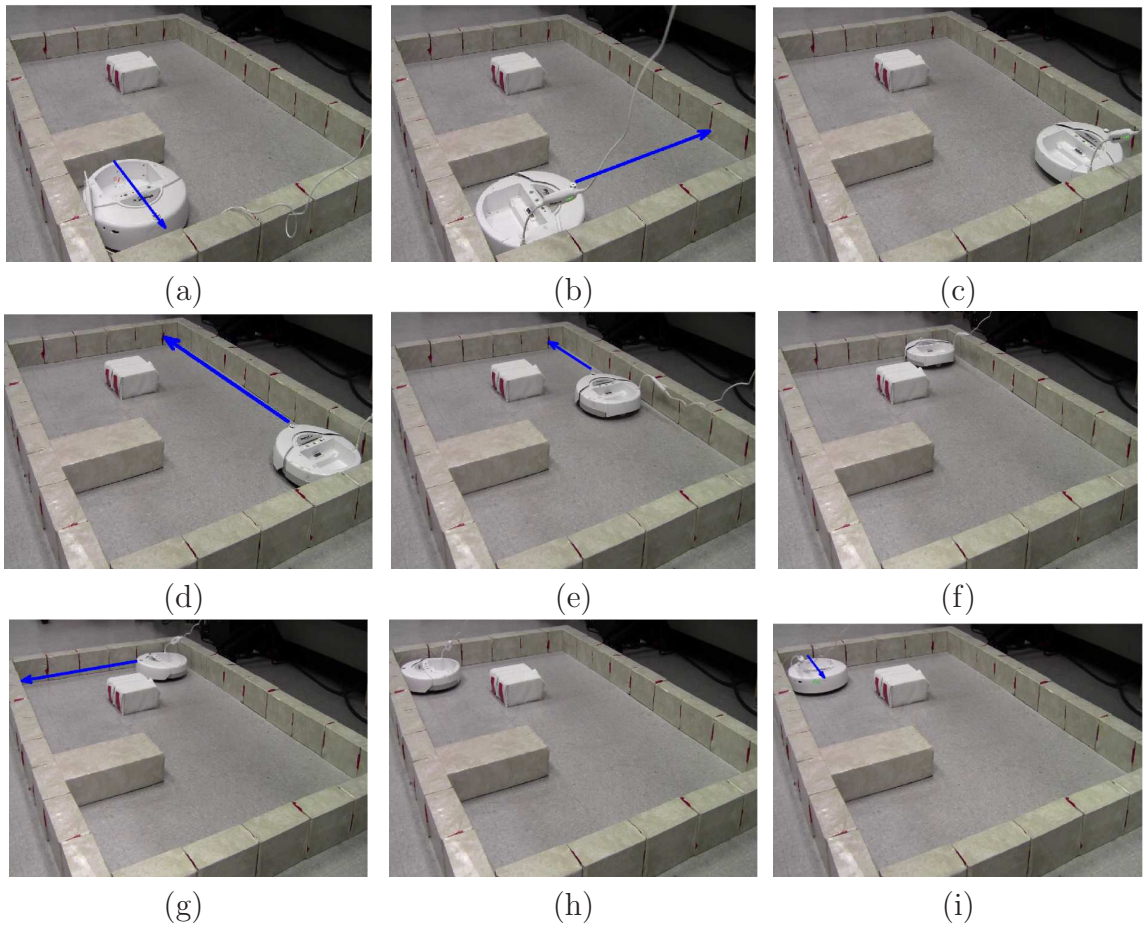


Figure 3.7: Snapshots of different configurations of the robot executing the first navigation plan of the simulation result of Figure 3.6(a): a) the initial configuration; 90° rotations are illustrated by the snapshot transitions a–b, c–d, f–g, and h–i; after snapshots b, d, e, and g, the robot moves forward; i) the goal configuration.

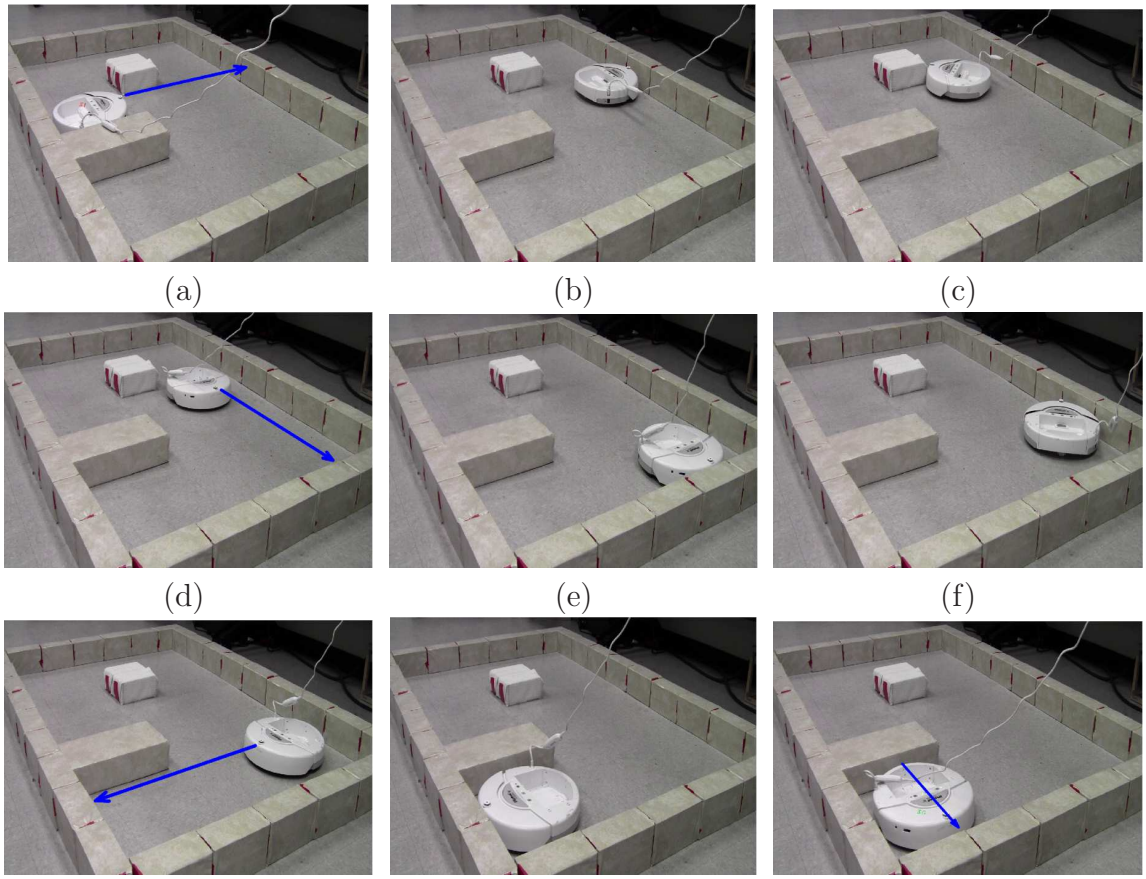
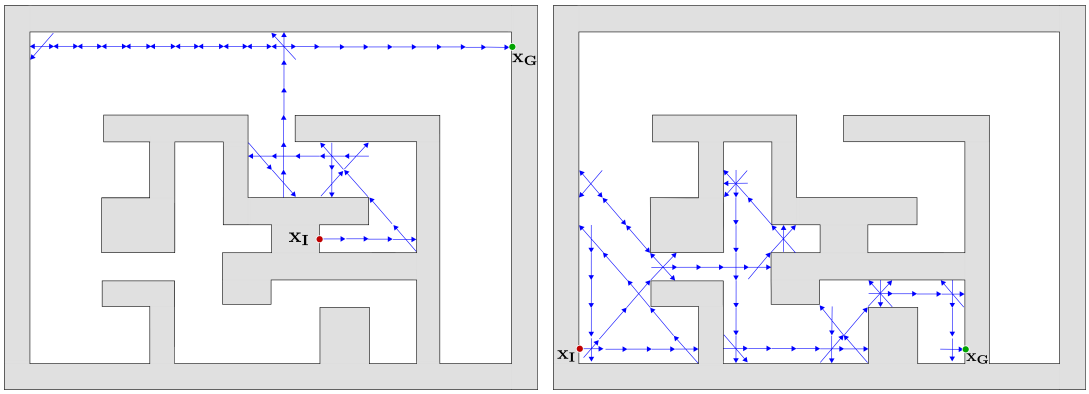


Figure 3.8: Snapshots of different configurations of the robot executing the second navigation plan of the simulation result of Figure 3.6(b): a) the initial configuration; 135° rotations are illustrated by the snapshot transitions b–c, c–d, e–f, f–g; a 90° rotation is illustrated by the snapshot transition h–i; after snapshots a, d, and g, the robot moves forward; i) the goal configuration.



(a) (b)

Figure 3.9: Simulation results of two navigation plans in a complex environment between pairs of initial configurations (red circle locations of the environment, facing East) and goal configurations (green circle locations of the environment, facing East).

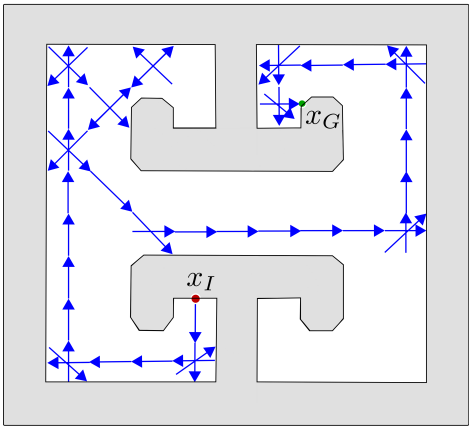


Figure 3.10: Another navigation plan generated by our algorithm that represents the blue arrowed path between the initial configuration (red circled location facing South) and the goal configuration (green circle location facing East) using all given bouncing angles 45°, 90°, and 135°.

3.5.2 All Minimum Navigation Plans Generation Result for a Single Robot

We generated all feasible minimum navigation plans for all (x_I, x_G) pairs in the cell configuration space X_{free} of the environment depicted in Figure 3.5 from the

simulation of Algorithm 3.2. All minimum navigation plans and path weights on X_{free} using Φ were stored. We used all of these navigation plans to answer multiple (x_I, x_G) navigation plan queries. Then, we compared the total number of minimum navigation plans using different numbers of bouncing angles. The comparison result is shown in Figure 3.11. This result suggests that most of the minimum navigation plans use only one bouncing angle, three bouncing angles are used less than two bouncing angles, and few of them use no bouncing angle, i.e., it does not bounce to get to the goal configuration.

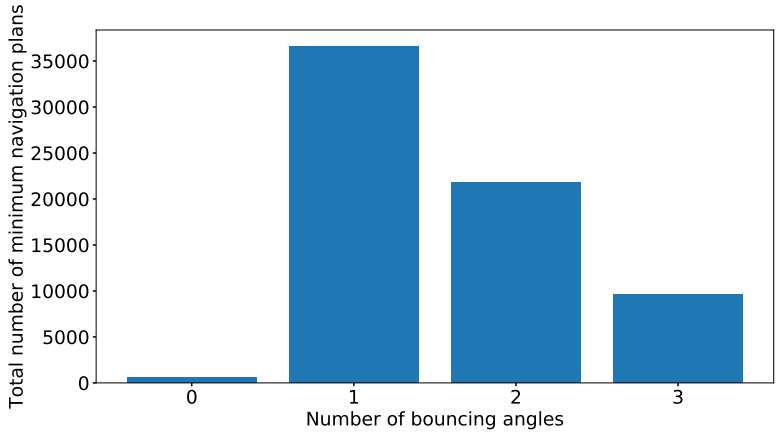


Figure 3.11: Comparison of using each number of bouncing angles for generated minimum navigation plans in the environment depicted in Figure 3.5.

We demonstrate another generated navigation plan from Algorithm 3.1 between the red circled initial configuration x_I and the green circled goal configuration x_G in the environment, as illustrated in Figure 3.10. We also generated all feasible minimum navigation plans for all pairs (x_I, x_G) in X_{free} of the environment depicted in Figure 3.10 from Algorithm 3.2. In this simulation, X_{free} had $N = 432$ cells considering the same 8 directions of S^1 with 45° separation between them and the set of bouncing angles $\Phi = \{45°, 90°, 135°\}$. Again, we recorded all minimum navigation plans and path weights, and compared the total number of navigation plans based

on the number of bouncing angles. The comparison result is shown in Figure 3.12. The result shows that most of its navigation plans use three bouncing angles. Fewer plans use two, one, or no bouncing angle. So, both comparison results show that the number of bouncing angles for all feasible navigation plans depends on the type of the environment and its complexity.

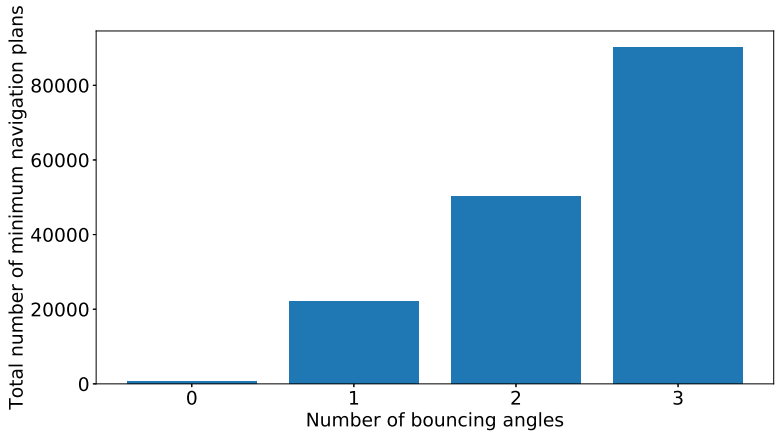


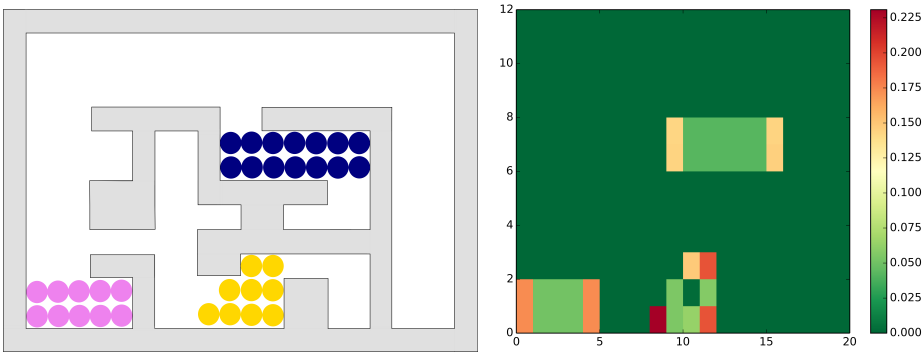
Figure 3.12: Comparison of using each number of bouncing angles for generated minimum navigation plans in the environment depicted in Figure 3.10.

In our simulation results, we do not have navigation paths for all pairs of (x_I, x_G) because for some of (x_I, x_G) pairs, there is no path on the roadmap $\mathcal{G}$. However, the inclusion of more reliable bouncing angles in the set, Φ, will guarantee the navigation path for all pairs of (x_I, x_G).

3.5.3 Result of Bouncing Policy Distribution for a Single Robot

We used the same complex environment E of Figure 3.9 for finding the bouncing policy distribution α to get as close to the uniform target coverage distribution b. The configuration space of the given environment X_{free} is discretized into $N = 65880$

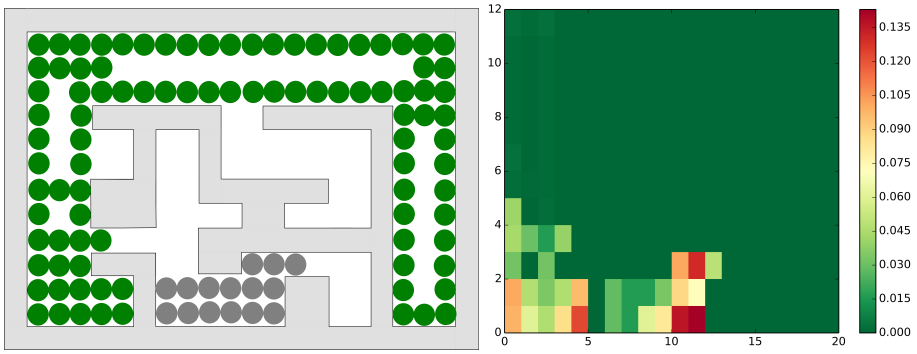
cells considering $S^1 = [0, 2\pi)$. We ran the simulation of Algorithm 3.3 on X_{free} for the bouncing angle set $\Phi = \{30°, 75°, 315°\}$ as three bouncing policies, the rotation error range $\epsilon = \pm 5°$, and the probability of reliable rotation $r = 0.8$. Based on the output from Algorithm 3.3, the visualizations of persistent groups in E for different directions of S^1 and their corresponding heatmaps of the limiting distribution set Π for bouncing angles including the error $30° \pm 5°$, $75° \pm 5°$, $315° \pm 5°$ are demonstrated in Figure 3.13, 3.14, and 3.15 respectively. The heatmaps show the probability of visiting different locations in the environment over time by a robot starting from any location.



(a) (b)

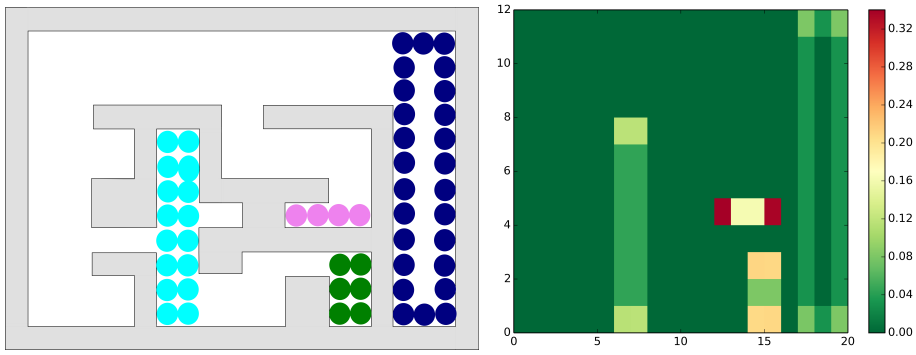
Figure 3.13: Persistent groups visualization in E for different directions of S^1 and their corresponding heatmap of the limiting distribution for the bouncing angle including the error, $30° \pm 5°$.

We applied the constrained least square method of Equation 3.8 using the result of the limiting distribution set Π and the uniform target coverage distribution b. As a result, the optimal probability distribution α of all bouncing policies is computed, and is tabulated in Table 3.1. This result implies that the robot has to use the bouncing policy of angle 315° about 50% of the time, and either the bouncing policy of angle 75° or the bouncing policy of angle 30° around 25% of the time.



(a) (b)

Figure 3.14: Persistent groups visualization in E for different directions of S^1 and their corresponding heatmap of the limiting distribution for the bouncing angle including the error, $75° \pm 5°$.



(a) (b)

Figure 3.15: Persistent groups visualization in E for different directions of S^1 and their corresponding heatmap of the limiting distribution for the bouncing angle including the error, $315° \pm 5°$.

Table 3.1: Optimal bouncing policy distribution result

Bouncing policies, Φ	Proportion of choosing bouncing policies α
30°	0.22138172
75°	0.28005791
315°	0.49856037

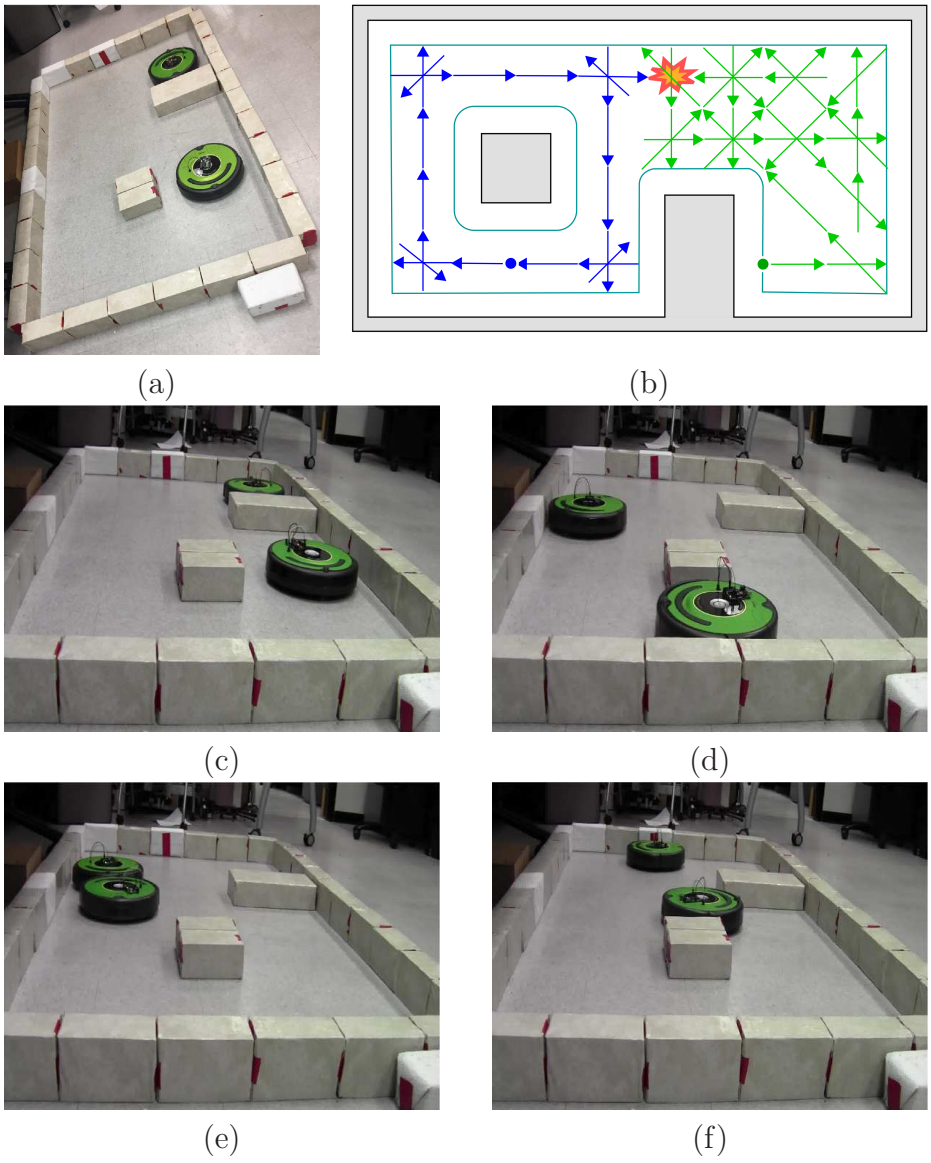
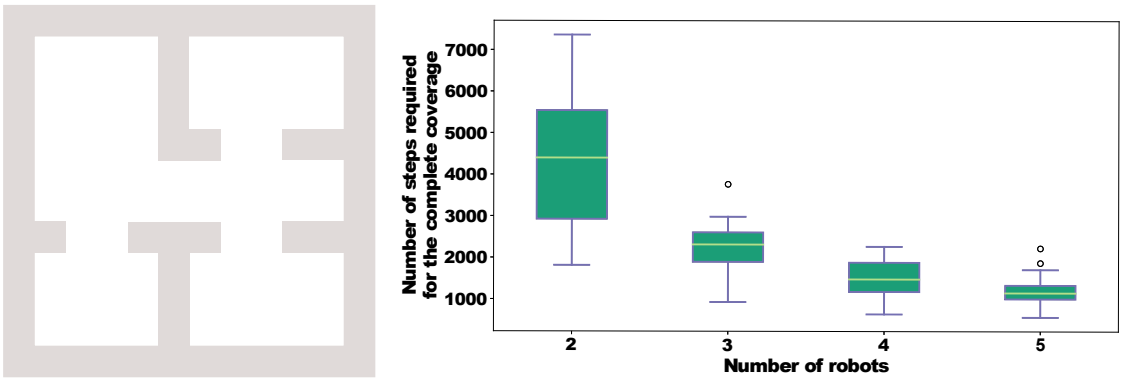


Figure 3.16: From the simulation to the physical implementation: a) the given environment to cover; b) a joint trajectory of the two robots generated from our simulation is depicted with the paths of blue and green arrows, the set of bouncing angles $\Phi = \{135°\}$ and the initial configuration of the two robots (locations of the blue circle, facing West and the green circle, facing East); there is only one robot-robot collision in the middle of upper part of the environment; c)–f) four different snapshots at different times of the hardware experiment of the generated joint trajectory of two iRobot Create 2.0 robots controlled with two Arduinos.

3.5.4 Result of Joint Trajectory of Multiple Robots for Coverage

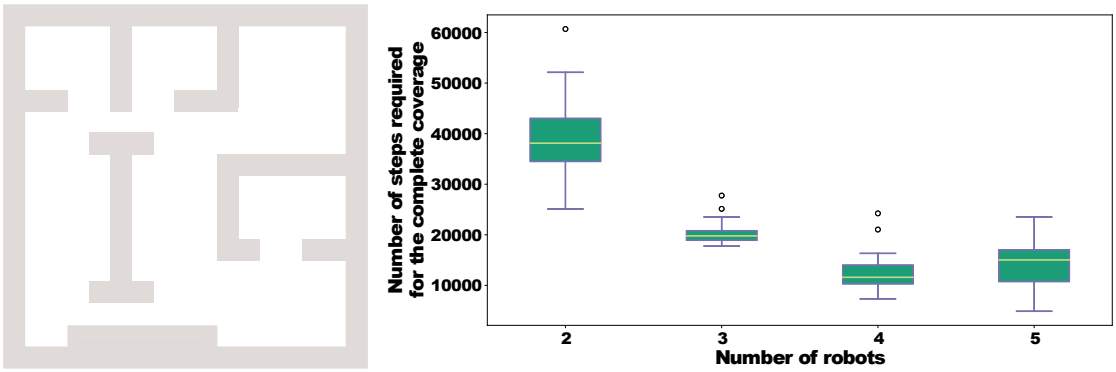
We also implemented our multi-robot coverage Algorithm 3.4 in the simulation. We used the laboratory environment and the associated configuration space X_{free} of Figure 3.16(a) for the iRobot Create Roomba. We verified our Algorithm 3.4 on X_{free} and a set of bouncing angles $\Phi = \{135^\circ\}$ for two robots, where both used the same bouncing angle and speed, and have identical sensing. The blue and green circles of Figure 3.16(b) represent the initial locations of the two robots facing West and East respectively. A joint trajectory generated from the simulation that covers X_{free} is depicted by blue and green arrows as paths of blue and green robots in Figure 3.16(b). To validate the simulated joint trajectory using a hardware experiment, we used two iRobot Create 2.0 robots and two Arduino Uno microcontrollers to drive them. As mentioned before, the iRobot Create 2.0 utilizes only the contact sensors and the clock. The same C++ program runs on both Arduinos that allows the robots to move forward in the free space at a fixed speed and rotate 135° after bumping at the boundary of the environment or with each another. Figure 3.9(c)–(f) shows four different snapshots of the hardware experiment.

We validated our Algorithm 3.4 again in the simulation for two environments of Figure 3.17(a) and Figure 3.18(a) considering multiple point robots and the bouncing angle $\phi = 135^\circ$ for all of them. The first environment is a 10×10 grid containing obstacles in it and the second environment is a 15×15 grid having obstacles inside of it. We used different numbers of robots (e.g., $m = 2, 3, 4, 5$) and different initial configurations for all robots in various simulation runs. We ran 20 simulation tests for both environments and demonstrated the plots in Figure 3.17(b) and Figure 3.18(b), comparing the result of the number of steps required for the complete coverage of the



(a) (b)

Figure 3.17: Simulation results of the multi-robot coverage: a) the first simulation environment; b) the comparison result of the number of steps required for the complete coverage of the first environment and the number of robots used.



(a) (b)

Figure 3.18: Simulation results of the multi-robot coverage: a) the second simulation environment; b) the comparison result of the number of steps required for the complete coverage of the second environment and the number of robots used.

environments and the number of robots used. This plot implies that the number of steps required for covering the whole environment decreases with the increase in the number of robots. In a complex environment, the increase in the number of robots does not guarantee the decrease in the number of steps required for the coverage of the whole environment due to the robot-robot collisions and the intricacy of the environment itself. An example of that is illustrated in Figure 3.18(b) where the

number of steps required for the complete coverage of the given environment with 5 robots is larger than the number of steps required for the complete coverage of the same environment with 4 robots.

3.6 Summary

In the second contribution, we proposed navigation and coverage methods using one or more robots equipped with a clock and contact sensors in a given environment. We constructed a directed graph from the environment using a set of bouncing policies. We found the minimum navigation plans on the graph between either one given pair of initial and goal configurations or all possible pairs of initial and goal configurations. The optimal bouncing policy distribution is calculated from the given set of bouncing policies to get the best possible coverage of the environment with respect to a target coverage distribution. We also constructed a joint trajectory for multiple bouncing robots incrementally until they covered the environment completely. We presented hardware experiments and simulation results of our methods, which shows the practical utility of our methods.

CHAPTER 4

STOCHASTIC MULTI-ROBOT PATROLLING

This chapter presents a scalable method to find stochastic strategies for multi-robot patrolling under an adversarial and communication-constrained environment. This chapter includes some of the content from our previous publications [AEBS15, ARBR17, Ala16] [1].

4.1 Motivation and Challenges

Patrolling is the problem of repeatedly visiting a group of locations of interest in an environment. This problem has applications in different areas such as environmental monitoring, infrastructure surveillance, and border security. In its multi-agent version, the action of patrolling is carried out by multiple robots as patrollers working together to ensure the safety of the area under surveillance. In its adversarial setup, one or more adversaries always try to penetrate the environment being patrolled by observing patroller's strategies and patrollers do not know where adversaries are going to attack. This problem can also become interesting when the action of patrolling is carried out by multiple patrollers with limited visibility in a communication-constrained environment. The communication based on visibility among patrollers in a congested environment may be impeded by various obstacles. In this context, autonomous robots can patrol the environment either in a distributed fashion where no communication among the robots is required or in a centralized manner, where each robot contacts a base location at a random time interval.

[1]The author acknowledges the help of Pedro Carrillo on the physical experiment presented in Section 4.5.7. Matthew Edwards and Md. Mahbubur Rahman provided help with the initial simulation codes.

Motivated by the presence of adversaries and the need for unpredictability, Agmon et al. [AKK08] presents perimeter patrolling strategies based on Markov chain models that maximize the probability of detecting an adversary [AKK08, Agm10, AKK11]. In this stream of research, only circular environments were considered, and the robots have to be synchronized and coordinated, which may prevent practical deployments of these strategies. Furthermore, all robots have to be placed in known locations and with equal distance apart in the environment. In our work, we do not require the robots to start from known locations nor to be synchronized in an environment. Instead, the robots share information by uploading and downloading data using a static node which is known as data muling [BTI11]. The robots in our solution will share information using visibility to the static node rather than mere proximity to it.

The problem of patrolling in the presence of adversaries can also be formulated as a *Bayesian Stackelberg Game*. In this game theoretical setting, the patroller first commits to an optimal mixed strategy generated through DOBSS (Decomposed Optimal Bayesian Stackelberg Solver) [PPM$^+$08] where each strategy is a path in a fully connected graph. Once the patroller has decided its mixed strategy, the adversary uses the knowledge of the mixed strategy to choose a location to attack. One of the potential drawbacks of this approach is that the set of strategies (paths) should be chosen *a priori* from a large set of possible paths and the particular connectivity of the graph is not studied in detailed.

Moreover, we consider that the autonomous robots move in a two-dimensional environment with polygonal obstacles and observe locations within their limited visibility region. They follow randomized patrolling paths that monitor the environment. The problem of finding a path for optimally patrolling the entire environment is related to an *optimal watchman route* [CJN99] which is an $\mathcal{NP}$-hard problem for

a polygonal environment with obstacles [CN88]. The difference between the optimal watchman route and our problem is that we have not considered the robot's visibility range to be infinite and we have added the adversary to the surveillance system of multiple robots.

Our work uses randomized patrolling policies in the form of Markov chains, taking inspiration from the problem of finding edge weights that minimize the effective resistance of a graph through convex optimization [GBS08, Boy06]. This problem is akin to the minimization of the average expected commute time for all pairs of vertices of the graph in the associated Markov chain. The purpose of this contribution is fourfold. First, we would like to extend current ideas into a more general class of graphs. The previous perimeter ideas can be extended to general graphs but this will need first finding a Hamiltonian cycle which is an NP-complete problem. Second, we would like to remove the need for communication, synchronization, and known initial placement of the randomized patrolling strategies. This will allow patrolling algorithms to be implemented with simple robots, in a decentralized fashion, and in communication denied environments. Third, we would like to adapt the game theoretical setup [PPM+08] to use Markov chains instead of deterministic strategies and to be applied to different sets of graphs. Fourth, we would also like to extend this optimization problem by modifying the minimization of average expected commute time for a subset of vertices that cover all or part of the environment with their visibility rather than using all the vertices of the graph.

4.2 Related Work

Multi-robot patrolling has been investigated in many different studies. Initial research [CSR04, AS06, EAK07] proposed deterministic approaches based on the op-

timization of the frequency of visits to the locations in the environment. Portugal et al. presented a multi-level partitioning algorithm (MSP) that assigns different locations to each patroller agent [PR10]. The performance of this approach is slightly better, but still deterministic. A survey of multi-agent patrolling strategies can be found in [PR11] where strategies are evaluated based on robot perception, communication, coordination, and decision-making capabilities. However, an adversary can easily penetrate the perimeter or area if a deterministic patrolling is used. For example, if a patrolling strategy ensures that a location around a perimeter is visited every 20 seconds and it takes an adversary 15 seconds to break in, then the adversary is guaranteed success if it attacks just after the location is visited [AKK08]. Nevertheless, with a nondeterministic strategy, patrolling robots would move more randomly around the environment, making it much more difficult for the adversary to effectively choose where to penetrate the perimeter environment.

In more recent works, Nicola et al. proposed the optimal strategy for patroller for patrolling the arbitrary environment like a set of connected cells, using single patroller or the smallest no. of patrollers required and based on several coordination dimensions among the robots [BGA09, NNF10, BGA12]. They have considered different penetration time for different cells of the environment and different target preferences among those cells for an intruder. They have formulated bilinear mathematical programming problem to find the optimal patroller's strategy. They have used Markovian property that maximizes the expected intruder-capture utility for patrollers. Nicola and Carpin again proposed probabilistic intrusions and a variable resolution sensing model that naturally applies to the domain of UAVs [BC12]. However, they have not used inherent Markov chain property to patrol the environment with optimal mixed strategy or fastest mixing strategy which is the best response from patroller against an intruder. Vorobaychik et al. presented a general

model of infinite-horizon discounted adversarial patrolling games [VATS14]. Here they assume payoffs for target locations are static over time.

Our visibility-based patrolling is related to the visibility-based pursuit-evasion solution for a single pursuer using optimal [SO12] and randomized [IKK05] and complete multiple pursuers [SO14] methods with only theoretical complexity where pursuers systematically search the environment to locate one or more evaders regardless of their motion. These works used a simple polygonal environment instead of the polygonal environment with holes which we have used in our work and the solution to the multi-pursuers problem is also computationally expensive. This problem is also different from the multi-robot coverage task [FDPM10] where a team of robots is used to jointly sweep an environment with their visibility sensors. The solution to this problem ensures the repeatedly visiting the locations of interest in the environment instead of going over all the locations once.

4.3 Preliminaries

4.3.1 Workspace and Motion Model

We consider m autonomous robots $\mathcal{B}_1, \mathcal{B}_2, \cdots, \mathcal{B}_m$ that patrol in a 2D polygonal environment $\mathcal{W} = \mathbb{R}^2$. Let $\mathcal{O}$ be the set of obstacles that block the visibility and are modeled as polygons. The free space in the environment is defined as $F = \mathcal{W} \setminus \mathcal{O}$. We model each patroller as a point robot without considering any orientation of it. As such, the configuration space of each robot is $X^i = F$. Let $x^i \in X^i$ define a configuration of a robot $\mathcal{B}_i$ where $x^i = (x_t^i, y_t^i)$ represents the robot's coordinate in the plane. We define the configuration space of the whole system as $X = X^1 \times X^2 \times \cdots \times X^m$. Let $X_{\text{obs}} = \{x \in X : x \cap O \neq \emptyset \text{ where } O \in \mathcal{O}\}$ be the obstacle

region of the whole state space. The collision-free state space is the remaining state space which is denoted as $X_{\text{free}} = X \setminus X_{\text{obs}}$. We discretize the free state space X_{free} into a finite number of the fixed-size 2-dimensional square-shaped cell. Let n be the number of cells in X_{free} and the navigable locations in the system are indexed by $z \in \{1, \ldots, n\}$. Accordingly, we define the collection of free cells as $Z = \{1, \ldots, n\}$ which is the set of all cells in X_{free}.

We consider each patroller as point robot that can translate in any direction and has an omnidirectional sensor or camera with a limited visibility range. In this case, a fixed 360° camera is considered for the robot which can also be achieved using multiple unidirectional cameras with a combined 360° view. Let the range of the visibility sensor for a robot be $l \in \mathbb{R}^{\geq 0}$. Here, the visibility sensor output is characterized as a *limited visibility polygon* in a circular region of radius l. Let $x_z \in F$ be the center point of a cell z. The limited visibility polygon $V(x_z)$ with a visibility range l centered at the point x_z is defined as [EGA81]:

$$V(x_z) = \{x_r | x_r \in F, \overline{x_z x_r} \cap F = \overline{x_z x_r}, |\overline{x_z x_r}| \leq l\}, \tag{4.1}$$

where $\overline{x_z x_r}$ is the line segment between two points x_r and x_z and $|\overline{x_z x_r}|$ is the distance between these two points. This limited visibility polygon $V(x_z^i)$, for a robot $\mathcal{B}_i$ with its current state $x_z^i \in X_{\text{free}}$ corresponds to the visible region of the environment while the robot $\mathcal{B}_i$ stays on the center point x_z of cell $z \in Z$.

4.3.2 Problem Formulation

We define the patrolling environment as an *undirected graph*, $G = (V, E)$ with $|V| = n$, $|E| = q$. Each vertex ($v \in V$) corresponds to a location, and each edge ($e \in E$) corresponds to a connection between two locations in the environment.

Next, a discrete time Markov chain M is defined on the graph G where M_{ij} denotes the probability of transitioning between vertex i and vertex j in one step. In this form, the patrollers need to know which node they are at in the graph when they arrive there to use the right weighting for outgoing edges. Once the robot has determined that it is at vertex i, it selects the i-th column from M, and uses this to weight its random choice of next vertex to visit. This is weaker than having to know your (x, y) locations at all times: indeed, given an initial location, a way of distinguishing the outgoing edges (e.g., in clockwise order), and a sensor which indicates to a robot that it has arrived at a vertex, no other localization information is required.

We assume that the patrolling mission is conducted in an adversarial setup where the adversary knows the information of the environment. We consider that the patrollers do not know the initial positions (vertices) of others in the environment (graph) and patrol the whole environment in a distributed manner. In this case, it is better to find a strategy for patrollers that minimizes the average expected commute time between every pair of vertices. The hitting time H_{ij} over a pair of vertices is the random time required for reaching vertex j for the first time when starting from vertex i in a Markov chain. The commute time, C_{ij} over a pair of vertices is the random time it takes to return to the vertex i for the first time starting from vertex i and passing through the vertex j using a Markov chain which is defined as $C_{ij} = H_{ij} + H_{ji}$. The average expected hitting time, $\overline{H}$, and average expected commute time, $\overline{C}$, for a Markov chain in the graph G are the expected hitting and commute time respectively, averaged over all pairs of vertices. In this context, the distributed patrolling problem will be defined as:

Problem 1. Finding distributed patrolling strategies:

Given the graph, $G = (V, E)$, find distributed strategies that minimize the average expected commute time ($\overline{C}$) : 1) for all vertices in V; 2) for a clique $v_1, \cdots, v_d \in V$, $d \leq n$; and 3) to a particular vertex $v \in V$.

We explicitly model an adversary who (1) knows all possible strategies that a patroller can choose, (2) has full knowledge of the environment, and (3) is able to optimally choose the vertex to attack in graph G. This scenario is known as a Bayesian Stackelberg Game where the two players of the game are the patroller and the adversary. In our proposed game theoretical setup, each patroller's strategy is a Markov Chain. The adversary also has a set of strategies to attack in any of the n vertices. The game is formulated as follows:

- A nonempty, finite set called the set of patrolling strategies $\mathcal{M} = \{M_1, \cdots, M_k\}$ where k is the number of Markov chains.
- A nonempty, finite set called the set of adversary strategies V. Each $v \in V$ is a vertex in the graph.
- A function $P : \mathcal{M} \times V \longrightarrow \mathbb{R} \cup \{\infty\}$ called the payoff matrix for the patroller.
- A function $Q : \mathcal{M} \times V \longrightarrow \mathbb{R} \cup \{\infty\}$ called the payoff matrix for the adversary.

In order to calculate both payoff matrices, we need the following values:

- d_i^{pat} : value of goods in location i to the patroller.
- d_i^{adv} : value of goods in location i to the adversary.
- c_i^{pat} : reward to the patroller of catching the adversary in the i-th location.
- c_i^{adv} : cost to the adversary of getting caught in the i-th location.

- p_i : the probability that the patroller will catch the adversary at the i-th location of the environment.

A patroller's reward must consider the factor c_i^{pat} for capturing the adversary (with probability p_i) and the reward value gets reduced when the adversary is not captured (probability $1 - p_i$). Conversely, the adversary pays cost c_i^{adv} (with probability p_i) but gains c_i^{adv} (with probability $1 - p_i$). Additionally, p_i also depends on the i-th hitting time of the Markov chain for each patrolling strategy. Given these definitions, we are interested in the following problem:

Problem 2. Generating the optimal strategies:
Given the payoff matrices, P and Q, the set of patroller strategies, $\mathcal{M}$, the set of strategies for an adversary, V, find the optimal mixed strategy for the patroller and the optimal strategy for the adversary.

We also convert the 2D patrolling environment into an equivalent undirected graph $G = (V, E)$. Each vertex $v \in V$ represents a cell z in X_{free} and each edge $e \in E$ represents the connectivity between two cells of the environment. Each vertex (cell) can be connected to 8-neighboring vertices (cells) from Z and the total number of edges in the graph G is q. Again, we consider that the patrollers patrol the whole environment (graph) through the visibility in a distributed manner. In this case, we find a visibility-based randomized strategy suitable for a patroller that minimizes the average expected commute time among a set of vertices (cells) permitting it to patrol the entire environment by itself. The patrollers will not be allowed to interact with each other and must act independently in a distributed fashion. Hence, we formulate the distributed patrolling policies as follows:

Problem 3. Finding visibility-based distributed patrolling policies: *Given an environment F and m autonomous robots as patrollers with limited visibility range l, find m visibility-based distributed patrolling policies where the policy of each patroller observes the whole environment from a subset of vertices independently, minimizing the average expected commute time $\overline{C}$ for the subset of vertices $v_1, \cdots, v_d \in V$, $d \leq n$.*

In contrast to the above problem, we can allow the robots to communicate with a central *base station*. We assume that the robots do not communicate with each other because they are in motion. The base station has two purposes: 1) collect information from the patrolling robots; 2) check whether a patrolling robot is functioning or not. The communication between the central base station and each patroller is established through the line of sight or the visible light. Thus, the communication range of the base station will depend on its visibility range.

Let b be the location of a base station placed on one of the cells in Z. The base station has a limited communication range (or the visibility range) r and does not move. We allocate m regions of the environment to m patrollers for patrolling. We assume that each patroller communicates with others by uploading and downloading data about its surveillance and availability to the base station after a random period of time. When a patroller fails or is disabled by the adversary, the base station makes that determination after some time and notifies other patrollers about it. In this scenario, we formulate the centralized patrolling policies of different patrollers as below:

Problem 4. Finding visibility-based centralized patrolling policies: *Given the environment F, a base station b, and m autonomous robots as patrollers, find the placement of the base station, m partitioned regions of the environment, and*

m randomized policies that monitor these regions and also contact the base station through the limited visibility of patrollers.

We define an adversary as D who can hide and attack at any location of the environment at any time. The time of patrolling is enumerated based on the number of time steps used in a Markov chain simulation. At the first time step, $t = 0$, the robots are placed in some random locations of the environment. At each time step, the robots will move simulating the Markov chain. Let t_a be the time steps required for the adversary to complete an attack at a location successfully and p_d be the probability of capturing the adversary. We assume that the adversary D attacks at a random location $x \in F$ and takes t_a time steps at the location x for completing the attack. It is considered that a robot can see the target location x from any boundary location of the limited visibility polygon $V(x_z)$ centered at the corresponding target cell z. Let the boundary region of the limited visibility polygon of the cell z be $\partial V(x_z)$. In this case, the probability of capturing the adversary at the location of cell z is calculated $p_d = p[H_{kz} = t_a]$ for a random location of cell or vertex $k \in V$ in the outside region of the limited visibility polygon $V(x_z)$. In other words, it refers to the hitting time between a random vertex in the outside region of the limited visibility polygon $V(x_z)$ and a vertex inside the boundary region of the limited visibility polygon $\partial V(x_z)$. To evaluate the vulnerability of patrolling policies, we find the probability of detecting the adversary at a given location. Thus, we formulate the vulnerability evaluation problem below:

Problem 5. Vulnerability evaluation of patrolling policies:
Given the environment F, a set of Markov chains $\mathcal{M}$ as patrolling policies, a target location (cell) $i \in F$, time steps of a successful attack for an adversary t_a, find the

probability of capturing the adversary p_d at the location (cell) i following the given patrolling policies.

4.4 Methodology

This section presents a game theoretical approach based on the distributed patrolling to obtain the optimal strategies and a method for finding both visibility-based distributed and centralized patrolling policies and their vulnerability evaluation.

4.4.1 Distributed Patrolling Strategies

As mentioned before, we consider patrolling strategies in a graph as a set of Markov chains $\mathcal{M}$. We define edge weights of the graph G as $w = (w_1, \cdots, w_q) \in \mathbb{R}^{\geq 0}$. An edge $e \sim (i, j)$ connects vertices i,j and so the incidence matrix A of the graph G is defined as:

$$A_{ie} = \begin{cases} 1 & \text{edge } e \text{ enters vertex } i \\ -1 & \text{edge } e \text{ leaves vertex } i \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, the incidence matrix A is an $n \times q$ matrix where $|V| = n$ and $|E| = q$. Now, the *weighted Laplacian* is the $n \times n$ matrix calculated as:

$$L = A \ \mathrm{diag}(w) \, A^T, \tag{4.2}$$

where $\mathrm{diag}(w)$ is the $q \times q$ diagonal matrix constructed from w. Since the weights are non-negative, L is positive semidefinite and it has the smallest eigenvalue 0 corresponding to the eigenvector $\mathbf{1}$.

We denote the eigenvalues of the Laplacian matrix L as

$$0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n.$$

Here, the average expected commute time $\overline{C}$ which is the expected commute time averaged over all pairs of vertices is formulated as [GBS08],

$$\overline{C} = \frac{1}{n^2} \sum_{i,j=1}^{n} \mathbf{E}\ [C_{ij}] = \frac{2}{(n-1)} \sum_{i=2}^{n} \frac{1}{\lambda_i}, \tag{4.3}$$

where $\mathbf{E}$ denotes the expected value.

In distributed patrolling scenarios, an environment with a small average expected commute time corresponds to a Markov chain with small expected commute times between vertices, and a large average expected commute time corresponds to a Markov chain with large expected commute times between at least some pairs of vertices. In [GBS08], a convex optimization method is proposed for minimizing the total effective resistance of the graph by allocating a fixed total conductance among the edges. Similarly, our first patrolling strategy of *minimization of average expected commute time* (MAECT) on a graph over all pairs of vertices is also the same convex optimization problem as follows:

$$\begin{array}{ll} \text{minimize} & \overline{C} \\ \text{subject to} & \mathbf{1}^T w = 1, \quad w \geq 0. \end{array} \tag{4.4}$$

The optimization variable is $w \in R^q$ which is the vector of edge weights. This MAECT patrolling strategy is equivalent to the problem of selecting weights on edges to minimize the expected commute time between vertices. Once $\overline{C}$ is considered as distances among the vertices, the MAECT is the problem of allocating the edge weights to a graph to make the graph small in terms of average distance between vertices.

In the second patrolling strategy, we have extended the MAECT problem for a subset of vertices or clique. This extension ensures a higher probability of traveling along the edges within the subset of vertices while still allowing for travel to the remaining vertices of the graph. For example, the edges set $g = \{\textsc{ShortestPath}(i, j), \textsc{ShortestPath}(j, k), \textsc{ShortestPath}(k, i)\}$, where $g \subset E$, is given priority for the subset of vertices {i, j, k}. This means that the edges along this shortest cycle will be optimized such that the edge weights will consist of the majority of the sum of all edge weights. We have formulated the convex optimization of minimizing average expected commute time (MAECT) problem for a subset of vertices as follows:

$$\begin{aligned} \text{minimize} \quad & \overline{C} \\ \text{subject to} \quad & \mathbf{1}^T w = 1, \quad w \geq 0 \\ & e = \text{ShortestCycle}(Q), \\ & \sum_e w_e \geq h. \end{aligned} \tag{4.5}$$

In this optimization, the variable e represents an edge along the shortest cycle between the vertices in the subset of vertices $Q \subset V$. The condition $\sum_e w_e \geq h$ ensures that the sum of the edge weights of $w_e \in w$, where $e \in g$, will be greater than or equal to a threshold h that represents the percentage of edge weights allocated to the cycle.

Let B be a coordinate matrix, $2 \times n$, for x and y coordinates of n vertices of the graph G. We present an algorithm for finding MAECT strategies towards each vertex.

Algorithm 4.1 generates a set of n Markov chains, $\mathcal{M}$, that minimizes the average expected commute time towards every vertex $v \in V$ for the graph G. For every Markov chain, the k shortest paths from v_j to the target vertex v_i are found (line

Algorithm 4.1: MAECTtowardsVertices(A, B)

Input: A, B – Incidence and coordinate matrices of the graph
Output: $\mathcal{M} = \{M_1, \cdots, M_n\}$ – Markov chains for each vertex

```
1  M ← 0
2  for i ← 1 to n do
3      M(n × n) ← 0
4      for j ← 1 to n do
5          if j == i then
6              for all edges e_jb ∈ A do
7                  M_jb ← M_jb + 1
8              continue
9          k ← K-ShortestPaths(B, j, i)
10         for all edges e_ab ∈ k do
11             M_ab ← M_ab + 1
12     normalize M
13     M ← M ∪ M
14 return M
```

9). The k shortest paths are found using Yen's algorithm [Yen71], which has a worst case runtime of $O(n^2)$. After the k shortest paths are found, every edge transition found within the k paths is incremented in the Markov chain that is being generated (lines 10–11). When $v_j = v_i$, all possible edge transitions from the target vertex v_j to adjacent vertices are incremented (lines 5–7). After all paths have been explored for a Markov chain, M is normalized and is added to the set of Markov chains $\mathcal{M}$.

4.4.2 Game Theoretical Approach

The game theoretical approach has two stages; the first stage is the payoff matrices calculation and the second stage is the optimal strategies generation.

Payoff Matrices Calculation

The payoff matrices for patroller and adversary, P and Q, are calculated from the set of Markov chains, $\mathcal{M}$. We calculate n Markov chains from Algorithm 4.1 and we also have two other Markov chains from MAECT for all vertices and MAECT for a subset of vertices or clique. Now, we have $n + 2$ Markov chains in the set of Markov chains, $\mathcal{M}$.

In Algorithm 4.2, we present a procedure to calculate the payoff matrices. The algorithm assigns uniform values of goods (d_i^{pat}, d_i^{adv}) in each vertex for both the patroller and the adversary. It calculates the hitting time matrix, H, (line 4) for each Markov chain. The mean hitting time vector (lines 6–7), $\bar{h}$, is calculated using the function HTofPreferredVertices. This function takes the vertices involved in the Markov chain optimization as follows: 1) In the case of MAECT strategy, it calculates the mean of the hitting times for all vertices; 2) In the case of MAECT for a subset of vertices or clique, it takes mean hitting time for the subset of vertices; and 3) In the case of MAECT towards each vertex, it takes the hitting time towards that particular vertex. In lines 11–17, it assigns the values to the variables needed to calculate the payoff matrices. In lines 18–20, it calculates the payoff matrices using these variables. Finally, it returns the payoff matrices P and Q for the patroller and the adversary respectively.

Optimal Strategies Generation:

The probability distribution of choosing the strategies for the patroller can be represented as a q-dimensional vector,

$$\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_q). \tag{4.6}$$

Algorithm 4.2: PayoffMatrixCalculation($\mathcal{M}$)

Input: $\mathcal{M} = \{M_1, \cdots, M_{n+2}\}$ – Markov chains for all patrolling strategies
Output: P, Q – Payoff matrices for patroller and adversary

for $i = 1$ **to** n **do**
  $d_i^{pat} \leftarrow 1/n \qquad d_i^{adv} \leftarrow 1/n$
for $i = 1$ **to** $|\mathcal{M}|$ **do**
  $H \leftarrow$ HittingTimes(M_i)
  $\bar{h} \leftarrow 0$
  for $j = 1$ **to** n **do**
    $\bar{h}_j \leftarrow$ HTofPreferredVertices(H)
  $asc \leftarrow$ SortAscending($\bar{h}$)
  $desc \leftarrow$ SortDescending($\bar{h}$)
  $c^{pat} \leftarrow 1 \qquad c^{adv} \leftarrow 1 \qquad j \leftarrow n$
  for $k = 1$ **to** n **do**
    $c^{pat}_{asc_k} \leftarrow c^{pat}_{asc_k} j$
    $c^{adv}_{desc_k} \leftarrow c^{adv}_{desc_k} j$
    $j \leftarrow j - 1$
  $p \leftarrow 1$
  for $k = 2$ **to** n **do**
    $p_k \leftarrow 1/2^{k-2}$
  for $k = 1$ **to** n **do**
    $P_{ik} \leftarrow p_k c_k^{pat} + (1 - p_k) d_k^{pat}$
    $Q_{ik} \leftarrow p_k c_k^{adv} + (1 - p_k) d_k^{adv}$
return P, Q

Equation 4.6 should satisfy: 1) $\alpha_i \geq 0$ for all $i \in \{1, \ldots, q\}$, and 2) $\alpha_1 + \alpha_2 + \cdots + \alpha_q = 1$. The value α_i is the proportion of the patroller choosing strategy α_i.

Similarly, u represents all the possible strategies for the adversary as an n-dimensional vector,

$$u = (u_1, u_2, \cdots, u_n). \tag{4.7}$$

Equation 4.7 should satisfy: 1) $u_i \in \{0, 1\}$ for all $i \in \{1, \ldots, n\}$, and 2) $u_1 + u_2 + \cdots + u_n = 1$. Since the adversary can attack any location $n \in |V|$ in the environment, the adversary's strategies, u_i, are strategies for attacking each location separately, and only one strategy can be chosen.

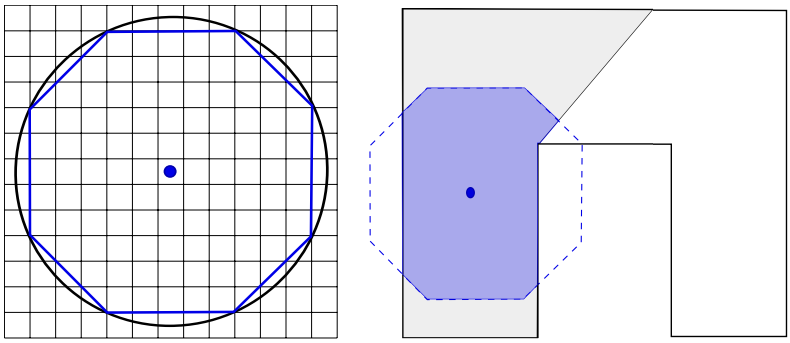
DOBSS generates the optimal mixed strategy for the patroller while considering an optimal adversarial response for all patrolling strategies [PPM+08]. It considers reward-maximizing strategies for patroller and cost-minimizing strategy for adversary.

4.4.3 Finding the Limited Visibility Polygons for Patrolling

To patrol the whole environment, we find a subset of cells for the limited visibility polygons that cover the whole environment.

In order to find the limited visibility polygons for patrolling, Algorithm 4.3 takes the environment F and the visibility range l as input and provides the set of minimum size subsets of cells that cover the environment completely through visibility for m patrollers separately as output. First, we initialize a set of visible cells Y that will contain the set of cells observed by all limited visibility polygons (line 1). In Algorithm 4.3, the function VisibleCells makes use of a visibility library [OC08] to compute a planar visibility polygon with infinite range for a point in the polygonal environment. Then, this function calculates a limited visibility polygon for the center point of a cell by taking the intersection of a visibility polygon with infinite visibility range and an approximated polygon with a limited visibility range. Thus, the limited visibility polygon $V(x_z)$ for every cell $z \in Z$ is calculated. An example of this calculation of the limited visibility polygon is illustrated in Figure 4.1(a)–(b). Once a robot is placed in the center of a cell, then it is possible for the robot to observe a subset of cells completely determined by its limited visibility polygon.

Therefore, the function VisibleCells finds the subset of cells that are completely visible by each limited visibility polygon $V(x_z)$. We add this subset of cells to Y (lines 2–3). So, the set of visible cells Y is a collection of subsets of Z such



(a) (b)

Figure 4.1: An illustration of a limited visibility polygon construction: (a) a blue approximated polygon of the blue circle centered cell that covers all the cells completely within a limited visibility range; (b) a limited visibility polygon (light blue region) taking the intersecting region between a visibility polygon of the infinite range (grey region) and the approximated polygon (in dotted line) with the limited visibility range.

that every element of Z belongs to at least one subset of Y:

$$Z = \bigcup_{S \in Y} S. \tag{4.8}$$

If we have one robot as patroller then it is an optimal solution to find the minimum size subset of cells $Q \subseteq Y$, whose elements represented by limited visibility polygons (they can be overlapping) cover all the elements of Z.

$$Z = \bigcup_{S \in Q} S. \tag{4.9}$$

However, the problem of finding the minimum size subset of cells Q is an instance of the set-covering problem which is $\mathcal{NP}$-hard [CLRS01]. Hence, our problem is similar to the set-covering problem and it does not have an optimal solution using a polynomial time algorithm.

Therefore, we apply a greedy approximation set cover algorithm, and the best feasible solution we get is an $O(\log n)$ approximation solution [Chv79]. Let P be the set of minimum size subsets of cells for limited visibility polygons that cover the

Algorithm 4.3: CompleteVisibility(F, l)

Input: F, l – Environment, Visibility range
Output: P – Set of minimum size subsets of cells that cover the complete visibility

```
1  Y ← ∅                          // Set of all subsets of visible cells
2  for z = 1 to n do
3      Y ← Y ∪ {VisibleCells(V(x_z))} // Calculate a subset of visible
         cells and add to Y
4  P ← ∅
5  for i = 1 to m do
6      U ← Z                      // Set of uncovered cells
7      Q ← ∅                      // Empty covered cells
8      while U ≠ ∅ do             // Check uncovered cells empty
9          select an S that maximizes |S ∩ U| in i-th order
10         U ← U \ S              // Remove covered cells
11         Q ← Q ∪ {S}            // Add covered cells
12     P ← P ∪ {Q}
13 return P
```

environment completely by m patrollers independently. It is empty initially (line 4). For each patroller, all the cells in Z are initially uncovered, and the covered set of cells Q is empty in the beginning. At each iteration of minimum size subset of cells generation, it selects a subset $S \in Y$ that covers the i-th maximum number of cells not yet covered where $i \in \{1, \ldots, m\}$ (line 9) (For example, for the second patroller, the algorithm finds the second maximum number of cells not yet covered.). Let U maintain the set of remaining uncovered cells and Q keep track of the collection of subsets of cells selected from Y. At each iteration, we remove the covered cells by selected subset S from U and add the subset to Q (lines 10–11). We continue this process until the set of uncovered cells becomes empty through the visibility of minimum size subset of cells. This minimum size subset of cells Q is added to P (line 12). We generate m minimum size subsets of cells for m patrollers. Finally,

Algorithm 4.3 returns the set of minimum size subsets of cells P in which the visibility of each minimum size subset of cells covers the environment completely.

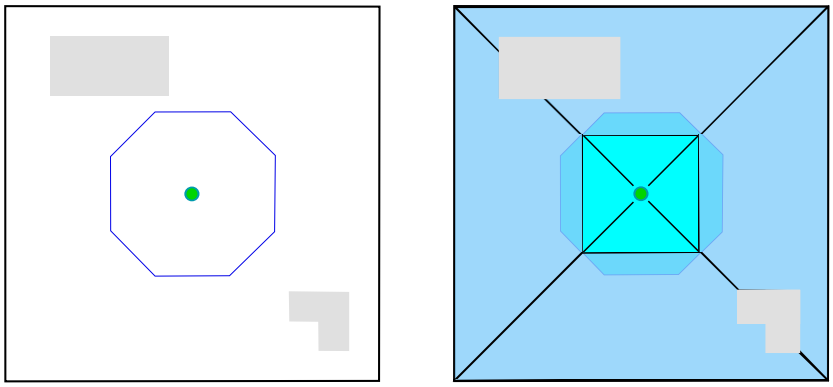
4.4.4 Finding Visibility-Based Distributed Patrolling Policies

The visibility-based distributed patrolling policies for m robots are presented as a set of Markov chains $\mathcal{M} = \{M_1, \cdots, M_m\}$ on the graph G. We minimize the average expected commute time for each minimum size subset of vertices in the graph applying the convex optimization of Equation 4.5. Then, we get m distributed and randomized patrolling policies for each minimum size subset of cells that cover the environment completely using their limited visibility polygons.

4.4.5 Finding Visibility-Based Centralized Patrolling Policies

To find the centralized patrolling policies, we place the central base station in the center of a cell based on two criteria. First, the placement of the base station is approximately at the center of the polygonal environment geometrically because it will be effective and convenient for the base station to coordinate other patrollers from the central location of the environment. Second, the visibility polygon for the center of a cell should have a large number of free cells covered by it so that the base station can communicate with other patrollers through its visibility rather than its proximity. To find the location of the base station, we find an initial set of cells around the center of the environment. Based on the two criteria mentioned above, we select a cell $b \in Z$ for placing the base station. We find the visibility polygon for

the center of cell b using the visibility range r as the communication range of the base station. One base station placement in a given environment and its visibility with the range r are shown in Figure 4.2(a).



(a) (b)

Figure 4.2: Base station placement and division of patrolling regions around the base station in the environment.

Next, we divide the environment F considering the location of the base station as the center or origin. We partition the environment F into m regions for assigning them to m patrolling robots. Since the exact partitioning of a polygon with holes is $\mathcal{NP}$-hard [LA06], so we apply a heuristic method where this partition is carried out by dividing the angular regions around the center of the cell location of the base station. To find the region for each cell location, we calculate the angular distance between the center location of the base station and the center of each cell. Based on the angular measurement, we assign each cell to respective regions. Each region has some overlapping cells with the visibility of the base station so that each robot can come within the visibility of the base station for communication purposes. An environment is partitioned into 4 regions around the base station for 4 patrollers according to 4 angular regions as illustrated in Figure 4.2(b). This partition also provides the overlapping (intersecting part between the visibility of the base station

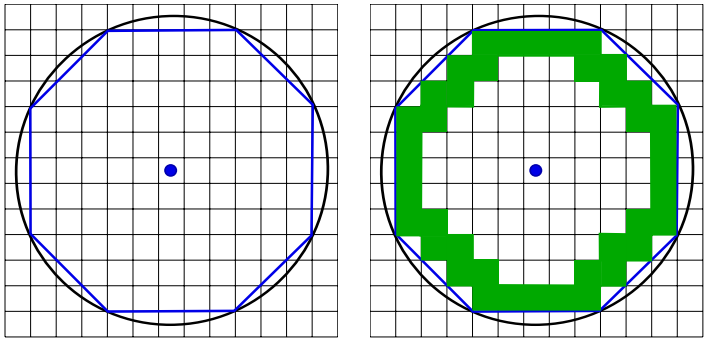
and each sky blue region of the environment) and non-overlapping part (cyan region) of the visibility region of the base station.

Then, we apply the same approximated set cover algorithm for covering each region completely. We obtain a minimum size subset of cells that need to be patrolled in each region and a contact cell from the overlapping cells with the base station visibility region. We generate a random patrolling policy for each patroller using the same convex optimization of Equation 4.5, minimizing the average expected commute time over the minimum size subset of cells and a contact cell for the corresponding region. Finally, we construct randomized patrolling policies for m patrollers monitoring m regions of the environment such that they can contact the base station via visibility-based communication. As the non-overlapped visibility of the based station is monitored by the base station at all times, this free area of the environment is not patrolled by any patroller.

4.4.6 Vulnerability Evaluation of Patrolling Policies

To evaluate the vulnerability of patrolling policies, we find the limited visibility polygon $V(x_z)$ as explained before for the center point of the given location of cell z in the environment F. We define the set of cells or vertices in $V(x_z)$ as $T \subset V$. Then, we calculate the boundary region $\partial V(x_z)$ from the limited visibility polygon $V(x_z)$ as illustrated in Figure 4.3. The boundary region $\partial V(x_z)$ of the limited visibility polygon consists of cells that are first reachable cells from the outside region of the limited visibility polygon $V(x_z)$. The target cell z is visible from the cells on the boundary region $\partial V(x_z)$ of $V(x_z)$. The set of cells or vertices in the outside region of the limited visibility polygon $V(x_z)$ in the environment is denoted as $T' = V \setminus T$. For the evaluation of patrolling policies, we simulate the given set of Markov chains

$\mathcal{M}$ starting from random states (cells) from T' until the simulation of one of the Markov chains reaches to one of the states (cells) on the boundary region $\partial V(x_z)$ of $V(x_z)$. We enumerate the hitting time for the simulation of the Markov chain to reach one of the states (cells) on the boundary region $\partial V(x_z)$ from the outside region of the limited visibility polygon $V(x_z)$. We repeat this process for n_o observations to obtain a distribution of hitting times. Since the hitting times are different due to the geometry and the characteristics of patrolling policies, it is hard to find a closed form of the distribution of hitting times analytically. Thus, we propose a numerical solution using the Freimer, Mudholkar, Kollia, and Lin (FMKL) generalized lambda distribution ($G\lambda D$) [FKML88]. Then, we fit the generalized lambda distribution to the empirical distribution of hitting times.



(a) (b)

Figure 4.3: A boundary region illustration: (a) the blue approximated visibility polygon $V(x_z)$ for a limited visibility range at a cell z; (b) the green boundary region $\partial V(x_z)$ of $V(x_z)$.

The distribution is defined by its quantile function, the inverse of the distribution function in FMKL $G\lambda D$ as below:

$$F^{-1}(p) = \lambda_1 + \frac{\frac{p^{\lambda_3}-1}{\lambda_3} - \frac{(1-p)^{\lambda_4}-1}{\lambda_4}}{\lambda_2}, \tag{4.10}$$

where p is the probability, $p \in [0, 1]$, λ_1 is the location parameter, λ_2 is the scale parameter, and λ_3, λ_4 are shape parameters jointly related to the strengths of the lower and upper tails, respectively. The advantage of the FMKL $G\lambda D$ is that the distribution is defined over all λ_3 and λ_4. The only restriction on the FMKL $G\lambda D$ is $\lambda_2 > 0$. Also, the maximum likelihood estimation is usually the preferred method for providing definite fits to a data set using the $G\lambda D$ [Su07]. Thus, we estimate the parameters of the $G\lambda D$ using the maximum likelihood method. Using the four parameters λ_1 , λ_2, λ_3, λ_4 of the generalized lambda distribution, we calculate the probability of detecting an adversary D for different time steps required for completing the attack t_a from the probability density function $f(F^{-1}(p))$ of $G\lambda D$. The probability density function $f(F^{-1}(p))$ of $G\lambda D$ does not exist in closed form. Hence, the probability density function $f(F^{-1}(p))$ is calculated from a numerical solution to Equation 4.10, using the Newton-Raphson method.

4.5 Experimental Evaluation

4.5.1 Decentralized Patrolling Result

We have compared our MAECT method with the three patrolling methods proposed by Agmon et al. [AKK08].In our approach, we place patrollers on the graph either randomly or an equal distance from each other (uniform) around the graph; Agmon et al's three methods, BMP, DCP, DNCP, all require that patrollers are placed an equal distance from each other on the graph. The patrollers are mobile robots such as UAVs or wheeled robots. In Figure 4.4, we show the mean first hitting time of MAECT with both uniform and random placement of patrollers compared to their three methods on a graph of 64 vertices and 2016 edges, (K_{64} : 2016).

Since their methods require a cycle graph to be tested on, we have tested their methods on a cycle graph of 64 vertices, (C_{64}), which is assumed to be equivalent to the Hamiltonian cycle of our original graph, K_{64}. In our test, we have simulated four patrollers so the distance between two consecutive robots, d, is 16. We also consider the number of state transitions it takes to penetrate the environment, t, as 12 units. Though our approach does not use synchronization and communication among robots, it still surpasses the mean hitting time of one of their methods and performance does not degrade much compared to the other two.

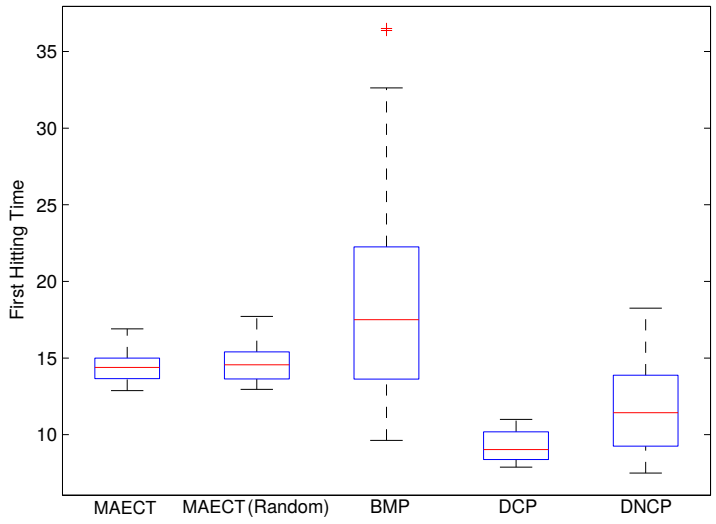


Figure 4.4: Comparison result of our MAECT method with uniform and random robot placement as well as existing three methods (e.g., BMP, DCP, DNCP) [AKK08] for patrolling. Each line represents the maximum, minimum first hitting time and each box represents the median along with the mean hitting time in the middle.

We compare more closely our MAECT method with DCP and DNCP methods using the varying number of patrollers on the graphs mentioned above (K_{64} and C_{64}). The result of this comparison is shown in Table 4.1. It shows that the more

patrollers are present on the graph, the closer the first hitting times of DCP and DNCP's methods approach the data from our approach.

Table 4.1: Comparison of average first hitting time of our approach, MAECT with uniform and random robot placement and existing two methods (DCP, DNCP) for 30 experiments

No. of Patrollers	Uniform MAECT	Random MAECT	DCP	DNCP
2 (d=32, t=24)	30.35	30.58	17.29	20.85
4 (d=16, t=12)	14.22	14.70	9.26	11.74
8 (d=8, t=6)	7.11	7.30	5.90	6.90
16 (d=4, t=3)	3.21	3.40	3.1	3.16

We have also tested our methods on different types of graphs, including line, tree, mesh, complete, and randomly generated graphs. Figure 4.5 shows the edge weight allocation on different graphs [GBY08] for our MAECT method. In each graph, the thickness of each edge corresponds to the optimal edge weight value, or the probability of that edge being chosen by a patroller. For example, a wider edge connection between two vertices represents a high probability of that edge being chosen for travel, and vice versa.

4.5.2 Game Theoretical Optimal Strategies Result

We have tested DOBSS with a small graph [GBY08] consisting of eight vertices and thirteen edges. The patroller has ten patrolling strategies available: eight which minimize the average commute time towards each of eight vertices, one that minimizes the average commute time towards a subset of vertices or clique, and one that minimizes the average commute time over all vertices. The resulting graphs for all ten patrolling strategies are shown in Figure 4.6.

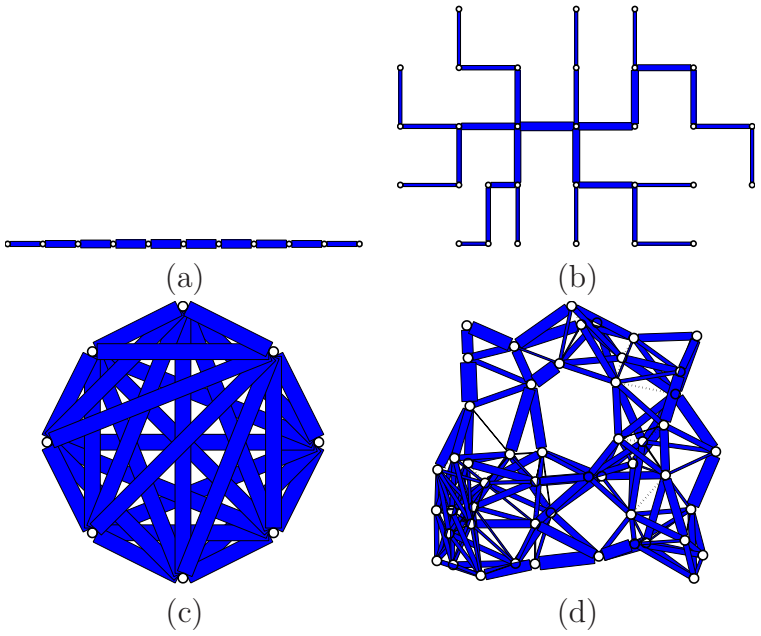


Figure 4.5: Different types of Graph for our MAECT method: a) Edge weight allocation on a line graph of 20 vertices; b) Edge weight allocation on a tree with 30 vertices; c) Edge weight allocation on a complete graph, K_8; and d) Edge weight allocation on a randomly generated graph with 50 vertices and 200 edges.

As an illustration, payoff matrices of a small graph are shown in Table 4.2, where the values of the payoff matrices for patroller and adversary, P and Q, for ten patrolling strategies are calculated using Algorithm 4.2.

DOBSS produces the optimal mixed strategy as shown in Table 4.3. The patroller will patrol the graph with an optimal mixed strategy consisting of strategies 7 and 9. Here the cost-minimizing strategy for the adversary generates an optimal response for attacking vertex 7 (i.e., $u_7 = 1, u_i = 0, i \neq 7$ for all $i \in \{1, \ldots, n\}$).

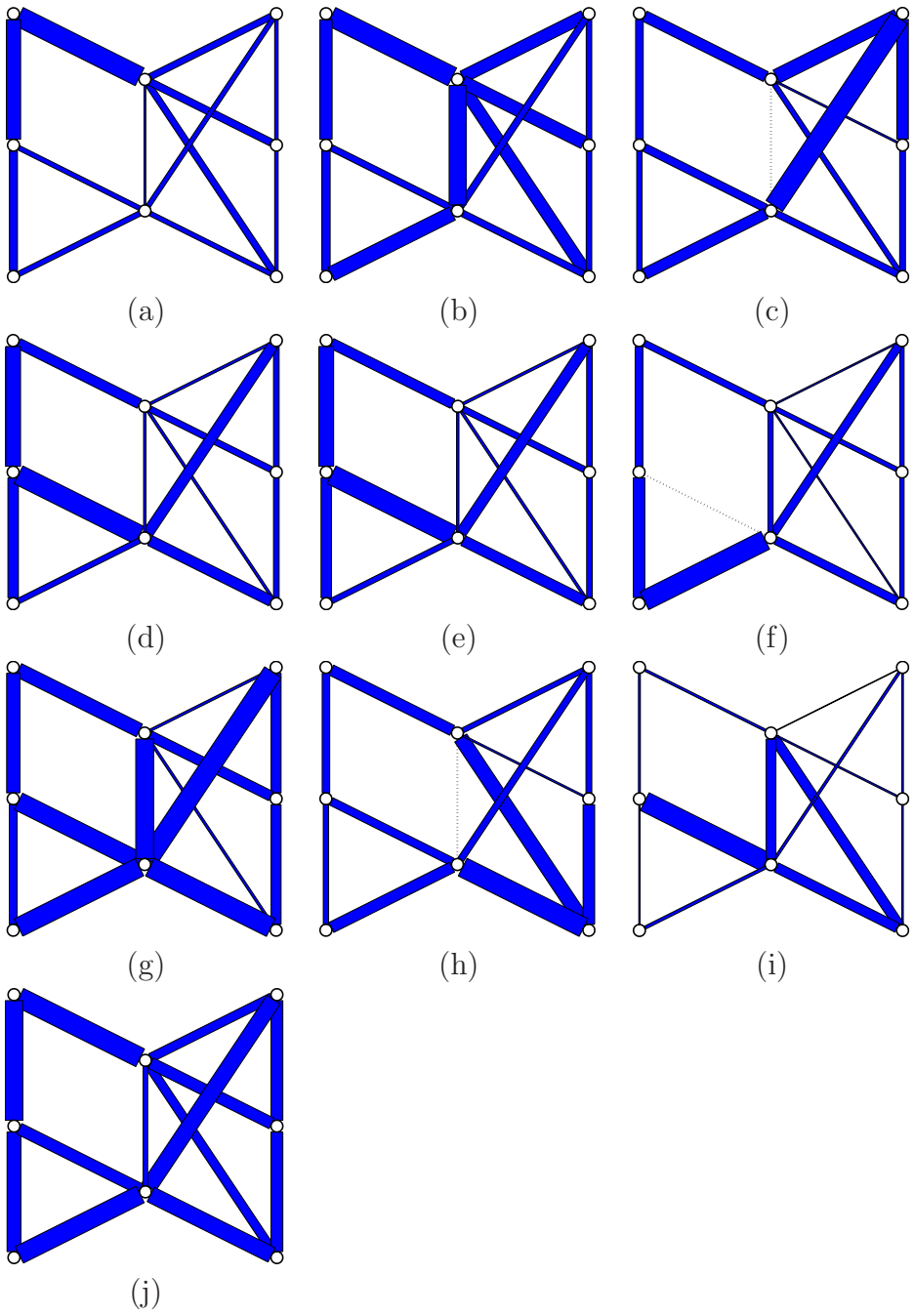


Figure 4.6: Edge weight allocation for ten patrolling strategies of a small graph: a)-h) Edge weight allocation for minimizing average commute time towards vertex 1 to vertex 8 respectively; i) Edge weight allocation for minimizing average commute time preferring a clique or subset of vertices, $\{2, 5, 8\}$; and j) Edge weight allocation for minimizing average commute time over all vertices.

Table 4.2: Payoff matrices for a small graph

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
P_1	8.0000	1.5938	0.1543	0.1318	3.5625	0.3672	0.7344	0.2148
Q_1	1.0000	0.8438	0.2324	0.1865	1.0625	0.4297	0.6094	0.3086
P_2	0.2148	8.0000	1.5938	0.7344	0.1543	0.1318	0.3672	3.5625
Q_2	0.3086	1.0000	0.8438	0.6094	0.2324	0.1865	0.4297	1.0625
P_3	0.2148	1.5938	8.0000	3.5625	0.1543	0.1318	0.7344	0.3672
Q_3	0.3086	0.8438	1.0000	1.0625	0.2324	0.1865	0.6094	0.4297
P_4	0.3672	0.7344	3.5625	8.0000	0.1543	0.1318	0.2148	1.5938
Q_4	0.4297	0.6094	1.0625	1.0000	0.2324	0.1865	0.3086	0.8438
P_5	0.7344	0.3672	0.1543	0.1318	8.0000	3.5625	1.5938	0.2148
Q_5	0.6094	0.4297	0.2324	0.1865	1.0000	1.0625	0.8438	0.3086
P_6	0.3672	0.1543	0.7344	0.1318	3.5625	8.0000	1.5938	0.2148
Q_6	0.4297	0.2324	0.6094	0.1865	1.0625	1.0000	0.8438	0.3086
P_7	0.1318	1.5938	0.7344	0.1543	0.2148	0.3672	8.0000	3.5625
Q_7	0.1865	0.8438	0.6094	0.2324	0.3086	0.4297	1.0000	1.0625
P_8	0.2148	1.5938	0.3672	3.5625	0.1318	0.1543	0.7344	8.0000
Q_8	0.3086	0.8438	0.4297	1.0625	0.1865	0.2324	0.6094	1.0000
P_{MAECT}	0.2148	3.5625	0.7344	0.1543	1.5938	0.1318	8.0000	0.3672
Q_{MAECT}	0.3086	1.0625	0.6094	0.2324	0.8438	0.1865	1.0000	0.4297
P_{Clique}	0.7344	3.5625	0.1318	0.2148	1.5938	0.1543	0.3672	8.0000
Q_{Clique}	0.6094	1.0625	0.1865	0.3086	0.8438	0.2324	0.4297	1.0000

4.5.3 Result of Visibility-Based Distributed Patrolling Policies

We implemented our Algorithm 4.3 in a simulation using Python. We take a 2-D environment which is discretized into a 15 unit $\times$ 15 unit grid including obstacles. The free space of the environment has n=188 cells. We consider the limited visibility range $l = 6$ cell units for $m = 2$ autonomous robots. We utilized a C++ library for visibility computations [OC08] to find the infinite visibility polygon and modified it to find the limited visibility polygon for each of the 188 cells of the environment. We found a set of limited visibility polygons for a minimum size subset of cells in

Table 4.3: Optimal mixed strategy result for the small graph

Patrolling Strategy No., M_i	Proportion of using Strategy α_i
1	0
2	0
3	0
4	0
5	0
6	0
7	0.9012
8	0
9	0.0988
10	0

the environment for the first robot that covers the whole environment as illustrated in Figure 4.7(a)–(i). It calculates the highest number of covered cells by a limited visibility polygon of a cell center. For the second robot, it calculates the second highest number of covered cells by a limited visibility polygon of a cell center. Thus, we found a set of limited visibility polygons for another minimum size subset of cells in the environment for the second robot that also covers the whole environment as shown in Figure 4.8(a)–(i). Since the limited visibility polygons that cover the highest and the second highest number of cells in the environment are close to each other, these limited visibility polygons look similar but cover the environment separately.

We extended the convex optimization method implemented in Matlab [GBY08]. As an illustration, we computed two visibility-based distributed and randomized patrolling policies for two autonomous robots. The randomized patrolling policy for the first robot is shown in Figure 4.9(a), based on prioritizing the minimum size subset of vertices (cells) for the visibility polygons of Figure 4.7 and the second randomized patrolling policy for the second robot is shown in Figure 4.9(b), based on prioritizing the minimum size subset of vertices (cells) for the visibility polygons

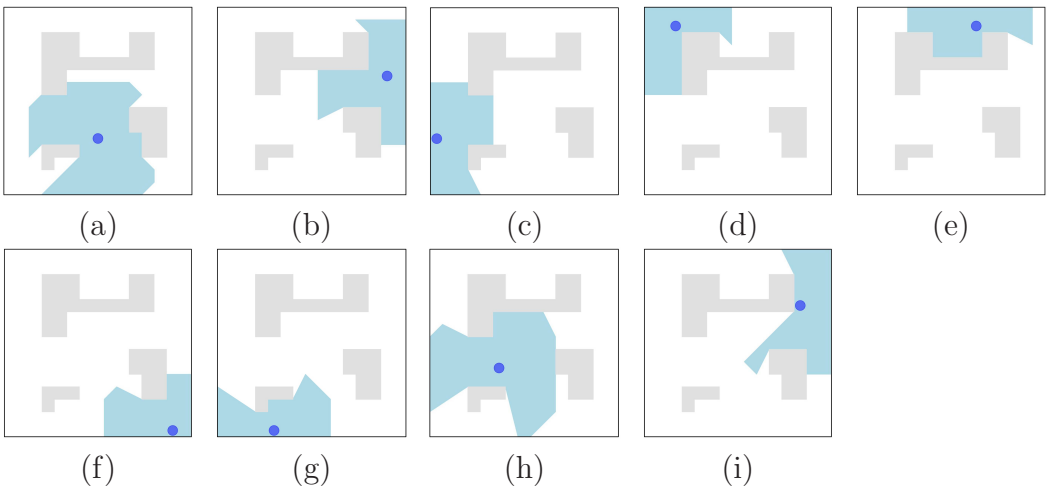


Figure 4.7: Limited visibility polygons (light blue region) for a minimum size subset of cells (blue circles) that cover the whole environment.

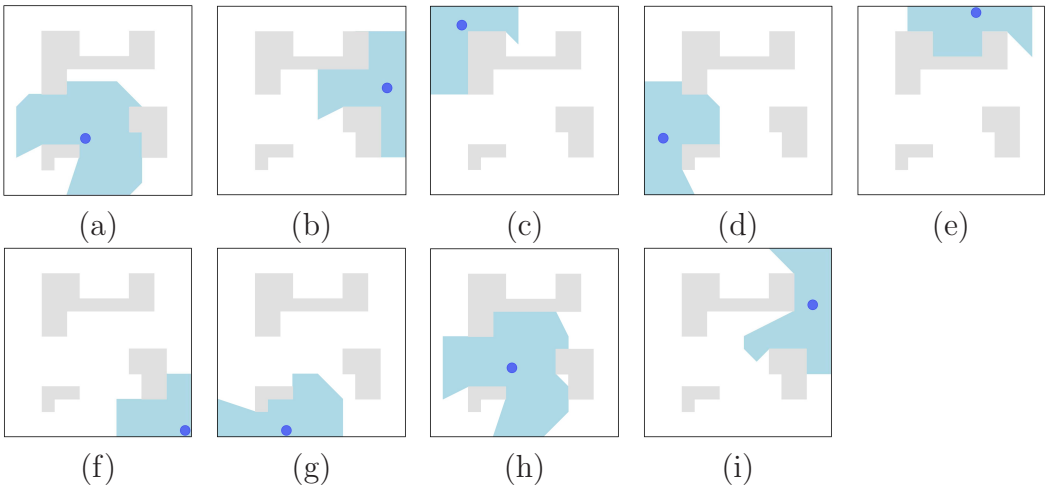
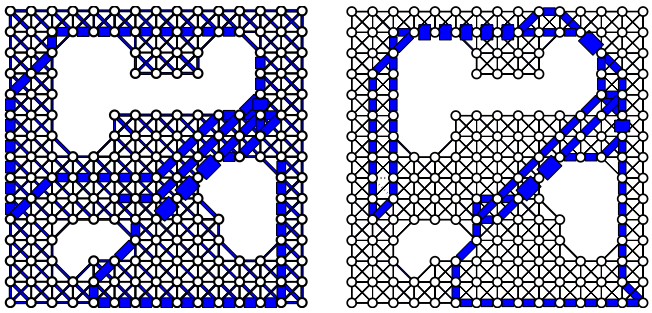


Figure 4.8: Limited visibility polygons (light blue region) for a minimum size subset of cells (blue circles) that cover the whole environment.

of Figure 4.8. In these graphs, we have $n = 188$ vertices and $p = 1046$ edges. In Figure 4.9, the width and color saturation of edges are proportional to the optimal edge weight value or the probability of that edge being chosen by a patroller. In other words, the optimal edge weight is larger on edges with more paths passing through them than edges near the leaves, which have fewer paths passing through

them. Therefore, the probability of choosing the shortest path among the minimum size subset of vertices for each robot is very high in both patrolling policies and these policies also choose other paths with low probabilities, to deceive the adversary.



(a) (b)

Figure 4.9: Distributed and randomized policies for two patrollers.

4.5.4 Result of Visibility-Based Centralized Patrolling Policies

For finding the visibility-based centralized patrolling policies, we take another 2-D environment discretized into an 18 unit $\times$ 16 unit grid with obstacles. We also provide the communication range $r = 6$ cell units for the central base station and the limited visibility range $l = 6$ cell units for $m = 3$ autonomous robots. Initially, there are 253 free cells in the environment. We found the limited visibility polygons with the communication range r in the same way as before, around the center of the environment. We placed the base station in the green circle (circle in the middle of the cyan region) of Figure 4.10– 4.12. In these figures, the communication or visibility region of the base station cell is shown in cyan color, and it covers the highest number of free cells. We partitioned the environment into 3 the angular regions ($0^\circ \leq \theta < 120^\circ$, $120^\circ \leq \theta < 240^\circ$, $240^\circ \leq \theta < 360^\circ$) based on the angular

distance between the center of base station cell and the center of other free cells. After removing the non-overlapping free cells of the communication region as they are visible from the base station, the number of free cells for patrolling in 3 regions is $n = 188$ cells. We found a minimum size subset of cells for the limited visibility polygons in each of three regions, covering each region completely using the same approximate set cover Algorithm 4.3. The sets of limited visibility polygons for minimum size subsets of cells (blue circles) in three regions are shown in Figure 4.10–4.12 respectively. The contact cells (blue circles) for each region are illustrated by the final picture in each of those figures.

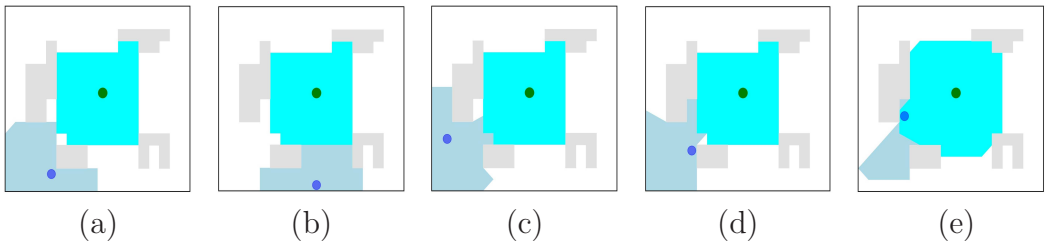


Figure 4.10: Limited visibility polygons (light blue region) for a subset of cells (blue circles) covering the first region of the environment and a contact cell (blue circle) in the base station (green circle) communication region (cyan region).

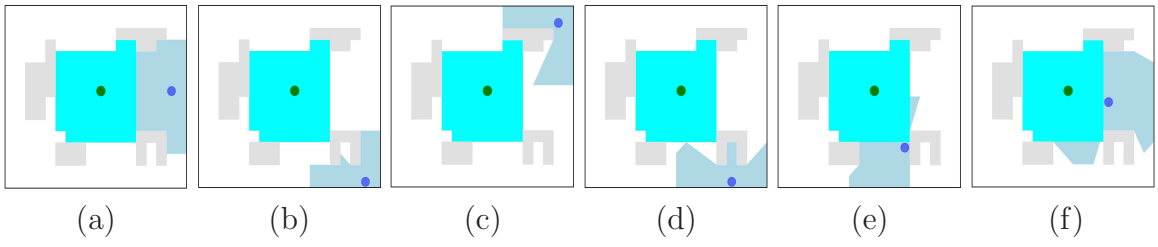
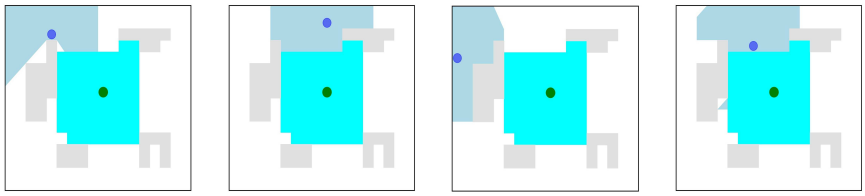


Figure 4.11: Limited visibility polygons (light blue region) for a subset of cells (blue circles) covering the second region of the environment and a contact cell (blue circle) in the base station (green circle) communication region (cyan region).

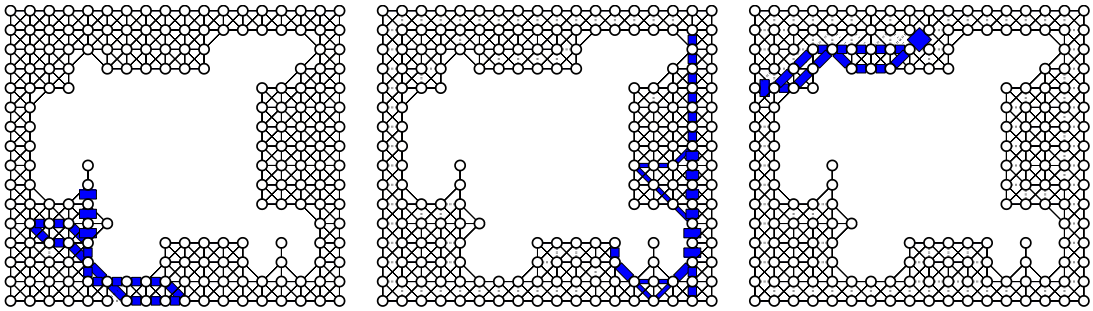
We computed three visibility-based centralized patrolling policies, again using the MAECT method in Matlab, for the respective minimum size subset of vertices



(a) (b) (c) (d)

Figure 4.12: Limited visibility polygons (light blue region) for a subset of cells (blue circles) covering the third region of the environment and a contact cell (blue circle) in the base station (green circle) communication region (cyan region).

(cells) and the contact vertex (cell) of the corresponding region. In these graphs, we have $n = 188$ vertices and $p = 989$ edges. Centralized and randomized patrolling policies for three autonomous robots are illustrated in Figure 4.13, with the robots contacting the base station to communicate. The number of paths along the edges of the shortest cycle of the minimum size subset of vertices and the contact vertex of each region of the environment is much higher than the number of paths on any other edge.



(a) (b) (c)

Figure 4.13: Centralized and randomized policies for three patrollers.

4.5.5 Computation Time Estimation of Visibility-Based Patrolling Policies

We ran our simulation experiments on a GNU/Linux computer with Intel Core i7 3.6 GHz processor and 16 GB memory. The running time (averaged over 10 repetitions) of Algorithm 4.3 for finding the minimum size subset of cells in our simulation runs is shown in Table 4.4. In Table 4.5, we provide the computation time (averaged over 10 repetitions) for determining randomized policies in both visibility-based distributed and centralized patrolling.

Table 4.4: Running time of Algorithm 4.3.

	No. of cells n	Time (milisec.)	
		Mean	Std. dev.
Algorithm 4.3 for Figure 4.7	188	2.7928	0.0634
Algorithm 4.3 for Figure 4.8	188	2.8072	0.0452
Algorithm 4.3 for Figure 4.10	188	2.7715	0.0845
Algorithm 4.3 for Figure 4.11	188	2.9198	0.0674
Algorithm 4.3 for Figure 4.12	188	2.8001	0.0855

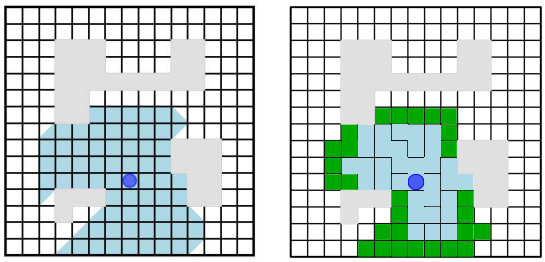
Table 4.5: Computation time for determining visibility-based patrolling policies.

	No. of vertices n	Time (sec.)	
		Mean	Std. dev.
Distributed policy of Figure 4.9(a)	188	733.2722	22.3968
Distributed policy of Figure 4.9(b)	188	676.6925	12.1284
Centralized policy of Figure 4.13(a)	188	779.4053	9.6480
Centralized policy of Figure 4.13(b)	188	744.9866	10.0691
Centralized policy of Figure 4.13(c)	188	752.8397	14.4261

4.5.6 Result of Vulnerability Evaluation of Patrolling Policies

For evaluating the distributed patrolling policies, we found a limited visibility polygon as illustrated in Figure 4.14(a) for the target blue encircled cell with the visibility range $l = 6$ cells. The boundary region of the limited visibility polygon is shown in Figure 4.14(b). This boundary region consists of green cells that are first reachable from the outside region of the limited visibility polygon. We simulated two Markov chain based distributed patrolling policies of Figure 4.9 starting from two random states (cells) in the outside region of the limited visibility polygon. We recorded the number steps required (hitting time) for one of the Markov chains that reached the boundary region of the limited visibility polygon from its starting state. This simulation of Markov chains was repeated for $n_o = 5000$ times, and we found the distribution of hitting times. We used the maximum likelihood method of the *gld* package in R [KDK16] to fit the distribution of hitting times. The histogram of recorded number steps required or the distribution of hitting times for 5000 Markov chain simulations and the result of fitting the maximum likelihood method of generalized lambda distribution to the hitting times are illustrated in Figure 4.15. Based on the result of the generalized lambda distribution, we calculated the probability of capturing an adversary p_d through the visibility of patrollers at the blue encircled cell of Figure 4.14 following two distributed patrolling policies of Figure 4.9 for different required time steps of a successful attack t_a. The result of the probability of capturing an adversary p_d through the visibility of patrollers is shown in Table 4.6.

For evaluating the centralized patrolling policies, we found a limited visibility polygon as illustrated in Figure 4.16(a) for the target blue encircled cell in one of three regions as described before with the visibility range $l = 6$ cells. The boundary



(a) (b)

Figure 4.14: (a) The light blue limited visibility polygon for the target blue encircled cell; (b) the boundary region of the limited visibility polygon consisting of green cells that are first reachable from white cells in the outside region of the limited visibility polygon.

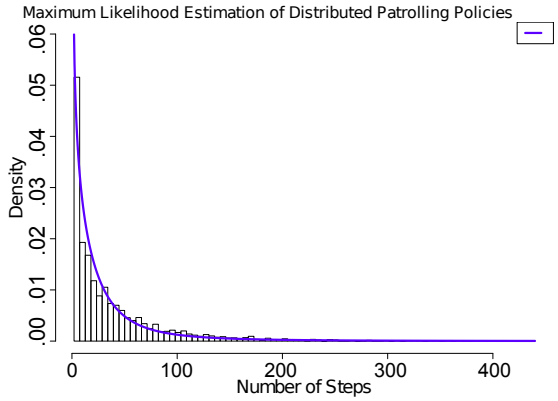


Figure 4.15: The histogram of the number of steps required (the hitting times) from 5000 Markov chain simulations for reaching the green boundary region of the limited visibility polygon starting from random white cells in the outside region of the limited visibility polygon of Figure 4.14 following two distributed patrolling policies of Figure 4.9 and fitting the maximum likelihood method of generalized lambda distribution to the hitting times.

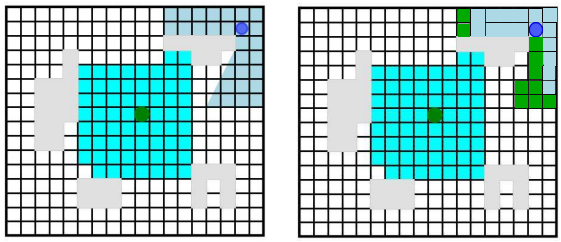
Table 4.6: Probability of capturing an adversary p_d through the visibility of patrollers at the blue encircled cell of Figure 4.14 following two distributed and randomized patrolling policies of Figure 4.9 for different required time steps of a successful attack.

	$t_a = 20$	$t_a = 30$	$t_a = 40$	$t_a = 50$	$t_a = 100$	$t_a = 500$
p_d	0.5139118	0.642008	0.726392	0.7844395	0.9109307	0.9926246

region of the limited visibility polygon is shown in Figure 4.16(b). This boundary region consists of green cells that are first reachable from the outside region of the limited visibility polygon. We simulated three Markov chain based centralized patrolling policies of Figure 4.13 starting from three random states (cells) in the outside region of the limited visibility polygon in three allocated regions of patrollers. Again, we recorded the number steps required (hitting time) for one of the Markov chains that reached the boundary region of the limited visibility polygon from its starting state. This simulation of Markov chains was also repeated for $n_o = 5000$ times, and we found the distribution of hitting times which is fitted again by the generalized lambda distribution. The histogram of recorded number steps required or the hitting times for 5000 Markov chain simulations and the result of fitting the maximum likelihood method of generalized lambda distribution to the distribution of the hitting time are illustrated in Figure 4.17. Based on the result of the generalized lambda distribution, we calculated the probability of capturing an adversary p_d through the visibility of patrollers at the blue encircled cell of Figure 4.16 following three centralized patrolling policies of Figure 4.13 for different required time steps of a successful attack t_a. The result of the probability of capturing an adversary p_d through the visibility of patrollers is shown in Table 4.7.

Table 4.7: Probability of capturing an adversary p_d at the blue encircled cell of Figure 4.16 following three centralized and randomized patrolling policies of Figure 4.13.

	$t_a = 20$	$t_a = 30$	$t_a = 40$	$t_a = 50$	$t_a = 100$	$t_a = 500$
p_d	0.1902747	0.2696536	0.3383983	0.3988624	0.6166325	0.9656368



(a) (b)

Figure 4.16: (a) The light blue limited visibility polygon for the target blue encircled cell and the cyan visibility region of a base station; (b) the boundary region of the limited visibility polygon consisting of green cells that are first reachable from white cells in the outside region of the limited visibility polygon.

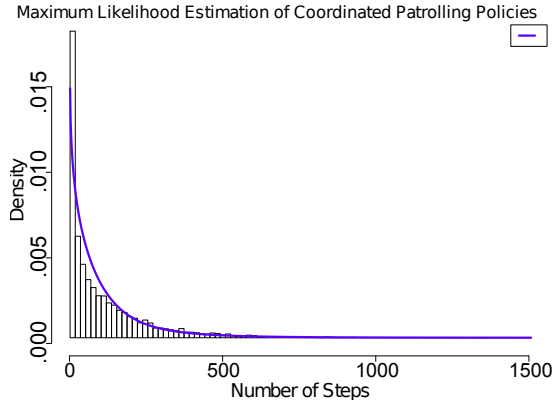


Figure 4.17: The histogram of the number of steps required (the hitting times) from 5000 Markov chain simulations for reaching the green boundary region of the limited visibility polygon starting from random white cells in the outside region of the limited visibility polygon of Figure 4.16 following three centralized patrolling policies of Figure 4.13 and fitting the maximum likelihood method of generalized lambda distribution to the hitting times.

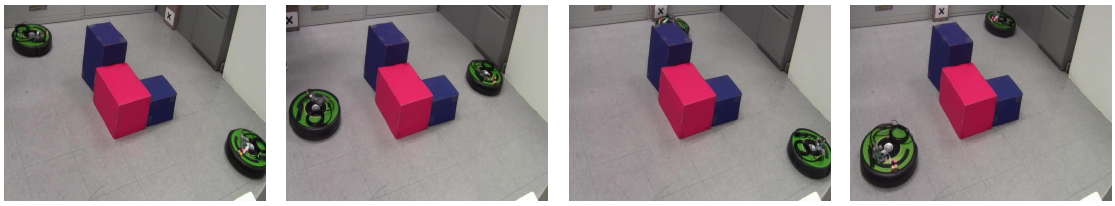
4.5.7 Hardware Implementation

To demonstrate the applicability of our method, we validated it through the physical implementation using two iRobot Create differential drive platforms as shown in Figure 4.18(a)–(b). We added two Raspberry Pi 3 for the motion control of the robots along with two separate rechargeable battery power sources and two unidirectional camera modules to these robots for taking their visibility into account.

The internal clock and a particular linear velocity of iRobot Create robots are used for their forward movements. The robots rotate with an angle from their current orientation by commanding a constant angular velocity and using the same clock to rotate for some fixed period of time. We used an indoor environment as a 6×6 grid environment including obstacles as illustrated in Figure 4.18(c) and the robots had to repeatedly visit five different locations $(A - E)$ in the environment. The visibility length of robots $l = 2$ cells in the grid environment was considered. In the physical experiment, a cross $(\times)$ marker was used to be detected by a patroller as a dummy adversary. The wave routing algorithm [Lee61] was implemented for path planning and avoiding obstacles. The software component of the robot is a Python program which is running on the Raspberry Pi to execute the randomly generated path for the robot, record the video of its patrolling, and detect the adversary ($\times$ sign). Two robots patrol the whole grid environment independently in the distributed fashion and do not communicate with each other. The snapshots at different time steps of the physical implementation of distributed patrolling policies using two iRobot create platforms are shown in Figure 4.19(a)–(d). The video of the experiment can be found at https://youtu.be/wSw9oBat6fE.

(a) (b) (c)

Figure 4.18: Two iRobot Create platforms along with added camera sensors and Raspberry Pis as microcontrollers and an artificial lab environment for patrolling with five different locations $(A - E)$ to visit repeatedly.



(a) t = 0 steps (b) t= 8 steps (c) t = 19 steps (d) t= 26 steps

Figure 4.19: Snapshots at different time steps of a distributed implementation of the patrolling task using two iRobot create platforms.

4.6 Summary

In this contribution, we introduced methods for finding distributed patrolling strategies for multiple robots based on Markov chains which minimize the average commute time towards (1) a specific vertex, (2) a subset of vertices, and (3) the average commute time for all pairs of vertices in the graph. Methods (1) and (2) use convex optimization and method (3) seek the shortest path in graphs. Even though we have used simple Markov chain based strategies for patrollers, one surprising result of our work seems to indicate that the performance does not degrade despite the lack of communication and synchronization.

We also presented both visibility-based distributed and centralized patrolling policies to monitor an adversarial environment with limited visibility and communication constraints. We found the limited visibility polygons for a subset of locations that cover the whole environment or a region of the environment. In distributed patrolling policies, every robot patrols the whole environment separately with a randomized patrolling policy. In centralized patrolling policies, each robot patrols a region of the whole environment and contacts a central base station also using a nondeterministic approach. We evaluated the vulnerability of our proposed visibility-based patrolling policies determining the capturing probability of an adversary at a given location of the environment. Our experimental results of stochastic

distributed and centralized patrolling policies for autonomous robots show that they have different probabilities on the edges of their patrolling paths, making the policies difficult for an adversary to predict.

The offline computation time of estimating visibility-based patrolling policies for large graphs entails a matter of minutes on a conventional computer. However, the online execution time of these policies is efficient since it only requires choosing a neighbor at each vertex in the graph following the Markov chains. The simulation of the Markov chains in the evaluation of patrolling policies is also very fast. Furthermore, the advantage of our method is that it is unpredictable for an adversary due to the randomization. Our patrolling policies can be implemented and executed on physical robots in real-time as demonstrated in our preliminary experiment. Since our method is randomized, the limitation of our method is that it can not give the exact guarantee of the completion time of the whole patrolling route, but can only provide the expected times. This also applies to the probability of detecting an adversary.

We consider static obstacles in the environment. We have not explicitly considered environments with dynamic obstacles. However, one alternative can be the re-estimation of patrolling policies based on changes in the environment. Also, a malfunctioning robot can be detected with the help of the base station. If the robot fails to make contact after a threshold time, the base station can determine that the robot is not operating. The base station can be aware of a malfunctioning robot as an obstacle if the robot becomes idle within the line of sight region of the base station. We used line of sight communication because it is more secure in military settings [JDH+06], harder to jam or intercept, and robust to noise [CD17]. Other models of communication can be incorporated which will change the characteristics of the base station, and the uncertainty will need to be added.

CHAPTER 5

DEPLOYMENT AND PLANNING FOR UNDERACTUATED AQUATIC ROBOTS

A data-driven deployment and planning approach is presented in this chapter for the underactuated aquatic robots called drifters. Parts of this chapter appeared in [ARBS18][1].

5.1 Motivation and Challenges

The dynamics and underlying phenomena of marine environments vary both spatially and temporally. Thus, the sensing, modeling, sampling, and prediction of marine environments are challenging tasks. To properly analyze and understand the environmental processes, we must observe them over long periods. Aquatic robots, such as Autonomous Surface Vehicles (ASVs), Autonomous Underwater Vehicles (AUVs), and low-cost drifting nodes (*drifters*), are increasingly being used for ocean exploration [WYSH00], oceanic sampling [PZL08], long-term ocean monitoring [MLS16], large-scale coral reef assessment [LRM$^+$16], and ocean model prediction [SPC$^+$10]. Since the environmental sampling or survey task entails persistent deployments, the resources used in the actuation and sensing of aquatic robots must decrease. Hence, an increase in the utility of inexpensive drifters has recently occurred [LRM$^+$16, SH14a, BFMI$^+$15, XLR16, MBS15]. In some instances, the drifters utilize a monocular camera to obtain geo-referenced, visual data of the benthos, e.g., images over a coral reef, while floating on the surface of the water carried by currents, waves, and wind. Simple drifters that are equipped with an inertial measurement unit (IMU), GPS sensor, WiFi communication module, and Raspberry PI comput-

[1]The author also acknowledges the collaboration of Gregory Murad Reis on the development of the ideas and simulations presented in this chapter.

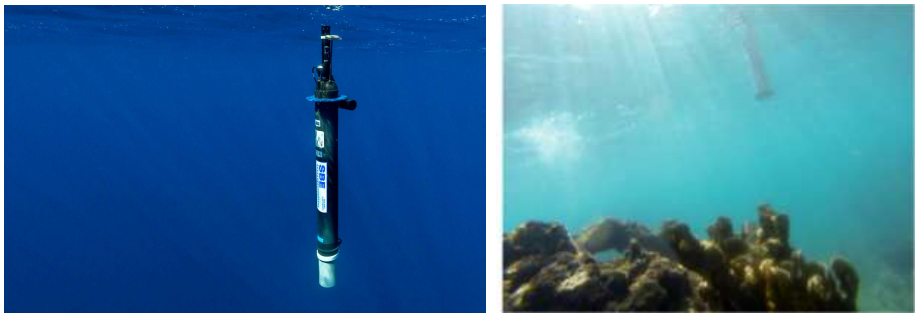


Figure 5.1: Two examples of drifters observing the marine environments at particular depths [XLR16, pfl].

ing unit, are suitable for the deployment in a marine environment for long periods of time, ranging from many days to many weeks. These drifters can collect data on various environmental attributes, such as temperature, salinity, turbidity, and chlorophyll content. Some drifters can also move vertically in the water column by controlling their buoyancy, and achieve controllability through the concept of controlled drift [SSS+11, SH14a]. These floating vehicles are referred to as Lagrangian profilers [DRDW92, JZZ13] or profiling floats [SH14b, HDS14, RJR+09] in the literature. Two examples of floating vehicles or drifters to observe marine environments are shown in Figure 5.1.

The challenges of using drifting nodes in marine environments are threefold: 1) these drifters do not have any control over their horizontal actuation; thus their planar motion depends on the ocean currents, 2) the sensing, modeling, and prediction of marine environments are hard because these environments are dynamic and their phenomena vary temporally, and 3) it can be computationally expensive to analyze the large volume of historical data from the long-term sensing of these environments. In this paper, we address Challenge 2 by utilizing predictions from an ocean model for ocean current data in a region of Southern California, illustrated in Figure 5.2. The goal of this work is to find deployment locations and long-term trajectories of

drifters from current velocity data near to the water surface implying where the drifters will travel (addresses Challenge 1). Practically, long-term trajectories of drifters will help us know the state of the environment based on the variations of spatial and temporal data (addresses Challenge 3 and feedback helps improve the models supporting Challenge 2).

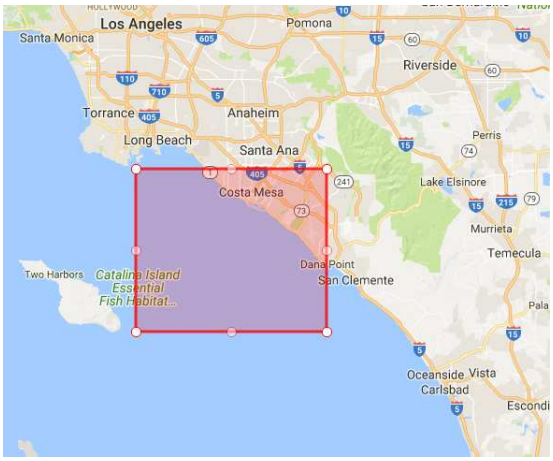


Figure 5.2: The area of interest in the Southern California Bight (SCB) region observed by the Regional Ocean Modeling Systems (ROMS) [SM05].

In our work, we investigate the problem of deploying multiple virtual drifters and the path planning and navigation policy of the drifter. However, the design and planning of a long-term drifter trajectory (i.e., a trajectory taking several days) are challenging because of the limited actuation capabilities and the dynamic behavior of the marine environment. We also examine a seafloor environment by deploying minimal drifters. This examination involves the covering of the seafloor environment through a maximum number of non-overlapping images or views from downward-facing cameras of drifters for extended periods of time. The problem of finding an optimal ordered sequence of views based on the maximal information gain for the complete coverage of an environment is known as the Next-Best View (NBV)

problem. This problem is $\mathcal{NP}$-hard because it can be reduced to a traveling salesman problem [Con85].

Considering the passive movement of drifters in a long-term deployment, we take the motion uncertainty into account caused by the disruption of ocean currents and winds. The ocean current data present the variations in spatial and temporal dimensions which are important for obtaining the persistent behavior analysis of the marine environment. To model spatiotemporal phenomena of the marine environment and the motion uncertainty, we use a Markov Chain-based state analysis of the drifters [Ste94]. More specifically, we apply the generalized cell mapping (GCM) [Hsu13] for finding the long-term behavior of drifters from their projected motion along with uncertainties in the form of a Markov Chain. Our second and third contributions [ABS17, ARBR17] are adapted to solve the deployment and planning problems in a marine environment.

5.2 Background

Many works have been proposed to study the behavior of aquatic robots and the long-term planning of their trajectories. A path planning and trajectory design for buoyancy-driven AUVs, called gliders, was proposed for analyzing dynamic features using ocean model predictions in the Southern California coastal ocean [SCL+10]. The critical assumption of their work was the ability of the glider to navigate to a given waypoint accurately. They did not consider the motion uncertainty of an AUV. Nevertheless, authors were able to propagate the errors due to the uncertainty of the motion of AUVs in [SKC+10]. The motion of different underwater vehicles was modeled as Gaussian Processes in [OLCJ14, MLS17] for path planning, as well as sensing and predicting the underlying phenomena in persistent ocean monitoring.

The long-term trajectories of the underwater vehicles were also applied to persistent monitoring [SSS+11, MLS16] of small or large marine environments. The planning method that navigates robots to achieve and maximize information gain through sampling the environment is called *informative planning* [BKS10, BKS13]. Here, we would like to have a long-term informative planning for the drifter from the persistent behavior of the marine environment.

An ensemble of heterogeneous aquatic robots was used to collect sensor data from a shallow underwater environment, e.g., a coral reef during a fixed period [LRM+16]. Several drifters were deployed for surveying Caribbean shallow coral reefs, resulting in visual mosaics of coral reefs [XLR16]. The drifters were also used to study the water dynamics via a Lagrangian approach by collecting the drifter positions as they move along the water surface over time [TCA+09, BFMI+15]. In [MKMD16], the authors used an ASV for data collection to do bathymetric mapping and visual mapping of open-water environments. The authors construct the bathymetric map from sonar sensor data using a Gaussian Process model and present a value-iteration based selective coverage algorithm which covers the entire region of interest in a prioritized fashion. They assume an underlying distribution of the phenomenon that needs to be modeled and builds an off-line trajectory to cover the high probability regions of a shallow region first and reduce the travel time and energy consumption at the same time. A coverage method for multiple AUVs with sea current disturbances was developed in [JLL+09]. A hybrid terrain coverage framework (HTCF) is presented for terrain exploration using the AUV in [LL14] that considers various surface conditions in three-dimensional environments and generates an efficient exploration path. However, a very restrictive assumption is that disturbances and external forces are not applied or considered. A vision for persistent and/or long-range seafloor exploration and monitoring utilizing ASVs and AUVs to conduct

autonomous surveys was outlined in [GJK+12]. Nevertheless, the implementation of this vision will entail careful coordination and a lot of resources whereas the deployment of drifters will be easy, distributed, and economical.

Autonomous Lagrangian profilers such as the ALACE [DRDW92] and profiling floats such as the Argo floats [RJR+09] or the PROVOR [LCMM98] have been utilized in oceanic observation and monitoring for decades. We consider profiling drifters or floats in our work. These profiling floats take measurements at different depths along vertical profiles by controlling their depth with an external bladder connected to a hydraulic pump [JZZ13]; buoyancy control. Argo floats are the most popular floating vehicles to gather data of deep ocean currents on a large scale. These data are used to study (and eventually predict) the variability in both the atmosphere and ocean. Currently, there are approximately 3800 Argo floats (as of April 3, 2018) in operation for continuous monitoring of the temperature, salinity, and velocity of the upper ocean [Bel12]. We make use of an ocean modeling and forecasting system to efficiently examine the long-term deployment of floating vehicles and increase their controllability.

The use of predictive models in path planning in the ocean is not a new concept, as ocean current models have been widely utilized, e.g., [CMN+92, KSBB07, GAO05, WD08, PPP+07, ACO04]. This approach has previously been considered to solve path planning optimization problems. Of particular interest to the ocean robotics community is utilizing ocean currents to minimize energy consumption, thus extending a vehicle's deployment time. Additionally, complex current structures experienced in a coastal region can vary significantly with time and location, making subsurface navigation difficult. The authors of this stream of research address the problem of path planning for AUVs in a complex, time-dependent, variable ocean. However, they assumed current velocities are generally coarse resolution av-

erages, as they are estimated from a compiled database, the average conditions as seen over long time periods or are provided only in two spatial dimensions. In our study, we propose not only to use high-resolution ocean models that output 4-D current velocities for path planning, but also to predict the structure and motion of the coherent structure from prior information.

5.3 Preliminary Description

In this section, we first describe the representation of the environment, and present the motion model for the specific type of aquatic robots considered. Then, we formally state the problems that are addressed.

5.3.1 Environment and Motion Model

We consider a 3-D environment where a workspace is a marine environment denoted as $\mathcal{W} \subset \mathbb{R}^3$. The workspace is divided into a set of 2-D current layers, representing different depths (third dimension) of the environment. Let L be the total number of current layers in the environment. Hence, the workspace can be defined as $\mathcal{W} = W_1 \cup W_2 \cup \cdots \cup W_L$. At each current layer, we model the workspace $W_l \subset \mathbb{R}^2$, where $l \in \{1, \ldots, L\}$, as a closed polygonal environment. Let $O_l \subset \mathbb{R}^2$ be the land and littoral region of the environment at each layer which is modeled as a polygon, and is considered an inaccessible region for obvious reasons. The free-water space of the marine environment at each current layer is composed of all navigable locations for drifters, and it is defined as $E_l = W_l \setminus O_l$. The free water space in the whole workspace is denoted by $\mathcal{E} = E_1 \cup E_2 \cup \cdots \cup E_L$.

We discretize each workspace layer W_l as a 2-D grid. This grid is also called a cell workspace as this discretized grid is a collection of cells. Each grid point

has a geographic coordinate in the form of longitude, latitude, and depth (x_t, y_t, l_t), where $x_t, y_t \in \mathbb{R}$ and $l_t \in \{1, \ldots, L\}$. The geographic coordinate of each grid point represents the center of an equal-sized cell z. Hence, each cell in the grid is represented as $z = (x_t, y_t, l_t)$, where x_t, y_t, l_t denote longitude, latitude, and depth of the center of a cell z. Let N_l be the total number of cells in the free water space at each current layer E_l.

Initially, we consider a set of drifters to be deployed to explore E_1, the water surface for our given region. We model each drifter as a point-mass and neglect the orientation. The state space of each drifter, for a given current layer, represents all the navigable locations or cells in the layer, and is defined as $X_1 = E_1$. The state space X_1 for the drifter is discretized into a set of cells which are indexed by $z \in \{1, \ldots, N_1\}$. Let $Z_1 = \{1, \ldots, N_1\}$ denote the set of all cells in the state space of the free water surface. In each cell z, except the boundary ones, we consider the simplified scenario that a drifter has a total of nine actions, based on the currents, wind, and waves at the surface. For a non-boundary cell z, the set of actions for the drifter are moving N, NE, E, SE, S, SW, W, NW and the idle operation (staying in the same cell). We assume that the drifter moves from one cell z to another cell z' in E_1 following one of the nine actions, considered as the steady motion of the drifter. We include noise and uncertainty in the movement along with the steady motion to account for the unknown disturbances, modeling error, and unmodeled dynamics. When the drifter collides with a boundary cell z, then it will either stay in the same cell or move to one of the neighboring cells. All the potential options are assumed to have uniform probability.

5.3.2 Problem Formulation

In our first problem, we consider the monitoring or patrolling problem for a given region on the surface of the water. Specifically, How many drifters and where should they be deployed, such that we achieve total coverage (in 2-D)[2] of a given region at the surface of the water. We assume that the set of drifters will be deployed for a prolonged period (days to weeks) to solve a persistent monitoring task. Hence, the trajectories of the drifters, starting from known initial locations (cells), will help us understand their explored locations (cells) on the water surface. For this purpose, we study the long-term behavior of the deployed drifters. Let $\mathcal{C}$ be a set of initial cells for the deployment of drifters. In this scenario, we formulate the problem of finding the deployment locations of drifters so that their trajectories visit all cells Z_1 contained in E_1 over the period of the deployment. Hence, we formulate the following problem statement.

Problem 1. Determine the deployment strategy for a set of drifters: *Given the water surface of a marine environment E_1 and the motion model of drifters, find a lower bound of the required number of drifters and the set of initial deployment locations (cells) $\mathcal{C}$ of them for exploring/visiting all the locations of interest on the water surface.*

In some cases, we may not need to visit all the locations (cells) in a region of interest based on the footprint of the sensor. An example of this is shallow coral reef monitoring [XLR16, LRM+16] with cameras. We can visually *cover* a seafloor environment with a camera by visiting a subset of the region on the water surface.

[2]Although we consider the drifter as a point mass, the sensing radius of the drifter is non-trivial, hence total coverage is achievable. We assume that visiting a cell achieves coverage of that cell.

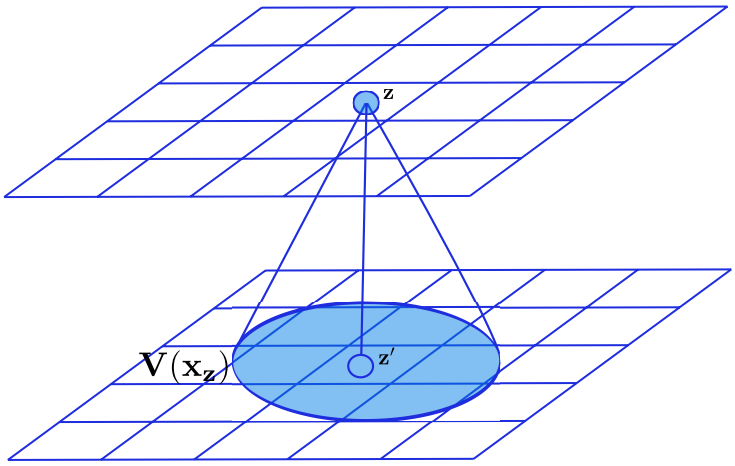


Figure 5.3: An approximated 2-D visibility polygon (blue filled region) from the water surface.

This is a result of the large footprint of the captured image from each visited location, see Figure 5.3. For covering a seafloor environment through the visibility of deployed drifters over an extended duration, we consider a shallow seafloor environment observed from a certain height h from the water surface E_1. The seafloor environment is denoted as E_L, which is the lowest layer of the environment and parallel to the water surface. Each drifter has a downward-facing, unidirectional camera to collect visual data of the seafloor environment E_L. Multiple drifters will collect visual data floating on the water surface to cover the seafloor environment. From a location on the water surface, a visibility region for the drifter creates a 2.5D visibility [KKLS10] of the seafloor environment. We approximate the 2.5-D visibility of the downward-looking camera of the drifter as a 2-D *visibility polygon* with a specific visibility range. Let the visibility range for the camera of the drifter be $d \in \mathbb{R}^+$. Therefore, the visibility polygon for a cell $z \in Z_1$, $V(x_z)$, with a visibility

range d centered at the point $x_{z'}$ corresponding to the projected cell z' at the lowest layer of the environment E_L, is defined as [EGA81]:

$$V(x_z) = \{x_r | x_r \in E_L, |\overline{x_{z'} x_r}| \leq d\}, \tag{5.1}$$

where $\overline{x_{z'} x_r}$ is the line between two points x_r and $x_{z'}$ and $|\overline{x_{z'} x_r}|$ is the distance between these two points. An example of approximated 2-D visibility polygon from a 2.5-D visibility region is illustrated in Figure 5.3.

In this context, we formulate our second problem of deploying drifters for the visibility-based coverage of a seafloor environment as below.

Problem 2. Construct a deployment policy for complete visibility-based coverage of a given region using drifters:
Given a seafloor environment E_L, a visibility range of the drifter camera d, and the motion model of drifters, construct a deployment policy for a set of drifters that covers the seafloor environment together through their visibility from a minimum number of initial deployment locations (cells) $\mathcal{C}$.

For the next set of problems, we propose to examine the full 3-D of a marine environment $\mathcal{W}$. This extends the previous problems to incorporating multiple current layers ($l > 1$). The free water space of this 3-D environment is $\mathcal{E}$. In this case, the state space of the drifter is $\mathcal{X} = X_1 \cup X_2 \cup \cdots \cup X_L$. The set of cells in all current layers is $\mathcal{Z} = Z_1 \cup Z_2 \cup \cdots \cup Z_L$. The total number of cells in all current layers of the environment is defined as $\mathcal{N} = \sum_{l=1}^{L} N_l$. Let $x_I \in X_1$ be an initial state of the drifter at the top layer ($l = 1$) of the environment or on the water surface. This x_I represents the initial deployment location of the drifter. In this work, we consider buoyancy-driven, profiling floats [SH14b, HDS14] that can control their vertical motion to achieve different depths in a marine environment,

and drift passively using different currents at different depths in the ocean. Hence, the action set of a profiling drifter is $U = \{-1, 1, 0\}$. This set of actions produces three different motions: "up" (move to the layer above), "down" (move to the layer below), and "drift" (stay at the same layer). A 3-D workspace and the actions of a profiling drifter in three different current layers of the environment are illustrated in Figure 5.4.

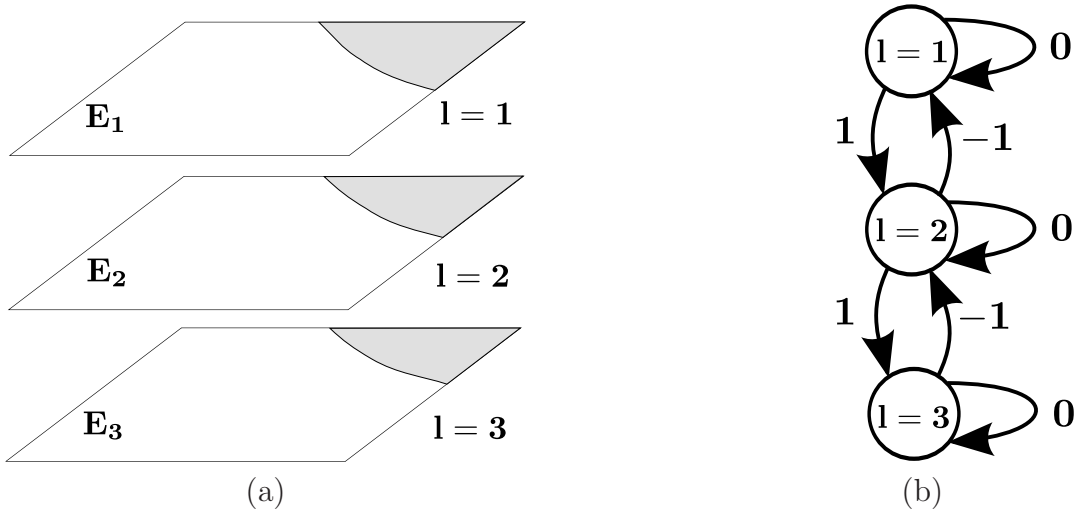


Figure 5.4: 3-D workspace and action space: (a) A multiple current layered marine environment and (b) Actions of a profiling drifter in three current layers of the marine environment.

In collecting data from a marine environment, it is crucial to find the locations where the profiling drifter can reach in the long run from its initial deployment location on the water surface; the reachability set. This information will assist in deploying the drifter in different initial locations to obtain as much as data possible of the environment. Let the set of all reachable cell locations at different current layers from the initial deployment location x_I over a long time period be $\mathcal{R} \subset \mathcal{Z}$. In this scenario, we formulate our reachability problem as follows.

Problem 3. Calculate the reachability of a long-term profiling drifter trajectory:
Given a marine environment $\mathcal{E}$, the motion model of a profiling drifter, and an initial deployment location of a drifter x_I on the water surface, characterize the long-term behavior of the environment and calculate the set of possible reachable locations $\mathcal{R}$ starting from x_I.

Let $x_G \in \mathcal{X}$ be a goal state or location of the drifter at any current layer of the environment. We define a navigation policy as $\pi : \mathcal{X} \to U$ that produces an action $u = \pi(x) \in U(x)$, for any location $x \in \mathcal{X}$, to reach the goal location. This policy considers the vertical movements of the profiling drifter along with its drift motion, which will be defined by the flow of water. Therefore, we formulate our final problem considered here as the following.

Problem 4. Develop an optimal navigation policy for a profiling drifter:
Given a marine environment $\mathcal{E}$, the motion model of a profiling drifter, and a goal location of a drifter x_G, find the optimal policy π that drives the drifter from any location of the environment to the goal location x_G.

5.4 Algorithm Description

In this section, we detail our approach for solving the four problems presented in Section 5.3.

5.4.1 Data Collection

In this work, we use the predicted oceanic current data from the Regional Ocean Modeling System (ROMS) [SM05] for the area of interest illustrated in Figure 5.2.

The geographic bounds of this region are given by $33°17'60''$ N to $33°42'$ N and $-117°42'$ E to $-118°15'36''$ E, and the region is referred to as the Southern California Bight (SCB) region, California, USA. Model predictions used here are from July 2011. ROMS is a free-surface, split-explicit, terrain-following, nested-grid mode, and an extensively used ocean modeling applications. ROMS is also an open-source ocean model that is widely accepted and supported throughout the oceanographic and modeling communities. Furthermore, the model was developed to study ocean processes along the western U.S. coast which is our area of interest. ROMS primarily assimilates surface velocities from HF radar data, and it is assumed that the forecasting for near-surface velocities is reasonable.

The four dimensions of the 4-D ROMS current velocity prediction data consist of three spatial dimensions, e.g., longitude, latitude, and depth, associated with time. Each ROMS current velocity prediction is given at depths from 0 m to 4000 m, with a 12-hour hindcast, a 12-hour nowcast, and a 48-hour forecast each day. In this work, we utilize a concatenation of the earliest 24 hours of each prediction for each day to create a simulated ocean over an entire month. The three velocity components of oceanic currents are the northing current (u), the easting current (v), and the vertical current (w). These velocity components are given based on the four dimensions (time, depth, longitude, and latitude). For the first two problems, we use the current velocity prediction data of the water surface at a depth or current layer of 0 m for a particular time. For the last two problems, we are using the current velocity prediction data of the water at depths or current layers between 0 m and 10 m. For all problems, we make the simplifying assumption that the vertical current velocity (w) is zero; no vertical currents are affecting the drifters.

5.4.2 Generation of a Vector Field and Flow Lines

Given the oceanic current prediction data of a given 2-D current layer of the environment, we generate a vector field from these data and obtain flow lines from them in this step. Ocean current velocity prediction data for a specific time and at a particular current layer can be characterized as a vector field. This vector field has the data variation in different geographic locations. Let the vector field on a cell z, or a geographic location on the environment at the processing layer E_l, be $F_l(z)$. For a cell z, the easting velocity component or the velocity along the latitude axis is denoted by $u(z)$, the northing velocity component or the velocity along the longitude axis is denoted by $v(z)$, and the vertical velocity component at the corresponding layer is denoted by $w(z)$. The vertical velocity component of the ocean current $w(z)$ is considered to be zero. Consequently, the vector field at the processing layer based on three velocity components for a cell z is specified as

$$F_l(z) = [u(z), v(z), w(z)]. \tag{5.2}$$

The vector field in Equation (5.2) can be interpreted as a set of ordinary differential equations. The solution to this system is the flow of water within the region. Flow lines of the water over the vector field F_l are the trajectories or paths traveled by a small particle, whose velocity field is given by the vector field for each location and time. This provides the following definition.

Definition 5.4.1 *(Flow Line): The vector-valued function c is a flow line of the vector field F_l if and only if, for all time t, we have that $c'(t)$ is parallel to $F_l(c(t))$. This is also referred to in the literature as a streamline.*

The vectors in the vector field are tangent to the flow lines. A flow line or a streamline for the given vector field from a particular location or a cell z, for all

time t, is illustrated in Figure 5.5(a). The numerical integration of a flow line over time produces the endpoint of the trajectory or path that a drifting particle would follow subjected to that vector field.

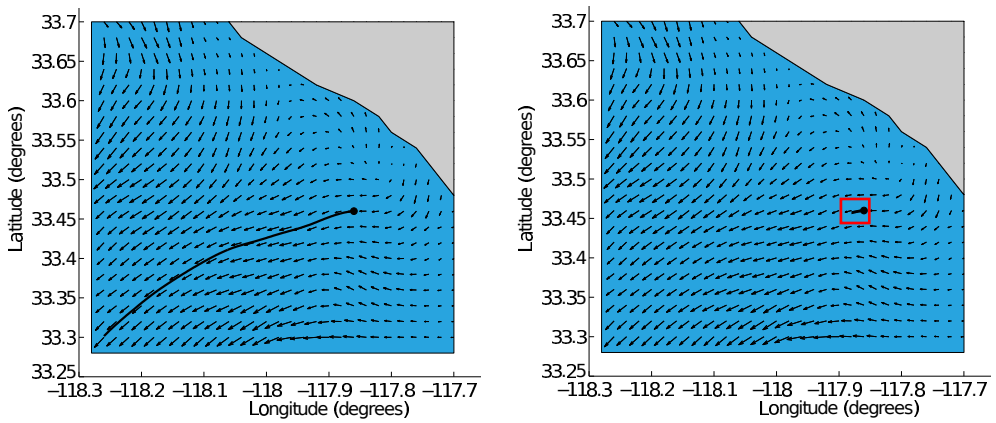


Figure 5.5: A flow line from a location: (a) The flow line of the given 2-D vector field for all time t; (b) The same flow line of the given 2-D vector field for a small time step Δt.

Let Δt be a small time step that is sufficient to cross a cell. For our study, we require all flow lines from cells in Z_l at the processing layer for a small time step so that we can map one cell to another cell based on these flow lines. The flow line for the given vector field from the same location or cell z as considered before for a small time Δt is illustrated in Figure 5.5(b). To calculate the next mapped cell $z(\Delta t)$ after a small time interval Δt from each initial cell at time zero $z(0)$, we use the Euler integration method following Equation (5.3)

$$z(\Delta t) = z(0) + \Delta t\, F_l(z(0)). \tag{5.3}$$

Equation (5.3) provides the endpoint of the flow line from the initial cell z after the small time Δt. After that, we use the Euclidean distance for locating the nearest cell from this endpoint. This nearest cell z' becomes the next mapped cell of the

initial cell z. Following this iterative process, we obtain all flow lines for the small time step Δt and calculate the next mapped cell for each cell at the processing layer of the environment E_l.

5.4.3 Finding the Long-term Behavior of the Water Flow

Given the cell mapping from the vector field and flow lines for a given 2-D current layer in the previous step, we use the GCM method [Hsu13, HX99] for finding the long-term behavior of the flow of water at that layer. Let ρ be the probability of the steady or perfect motion of the water flow. Once we add the uncertainty in the water flow, we get a set of mapped cells for each cell z at the processing layer. Let $A(z) \subset Z_l$ represent the set of mapped cells of a cell z and $p_{zz'}$ denote the mapping probability of cell z being mapped into one of the mapped cells z'. The mapping probability $p_{zz'}$ has the following properties:

$$p_{zz'} \geq 0, \quad \sum_{z' \in A(z)} p_{zz'} = 1. \tag{5.4}$$

For a non-boundary cell i, the mapping probability for the perfect motion $p_{ij} = \rho$ from cell i to cell j, and the mapping probability for imperfect motion $p_{ij} = \frac{(1-\rho)}{(|A(i)|-1)}$ from cell i to cell j. The calculated next mapped cell from the previous step identifies the mapped cell for the perfect motion. For a boundary cell i, we select all neighboring cells including the same boundary cell as a set of potential mapped cells with a uniform probability. Due to the nature of this cell mapping, the system evolution of GCM is expressed as

$$p(n+1) = Pp(n) \text{ or } p(n) = P^n p(0), \tag{5.5}$$

where P is the one-step transition probability matrix, P^n is the n-step transition probability matrix, $p(0)$ is the initial probability distribution vector, and $p(n)$ is the n-step probability distribution vector. Let p_{ij} be the (i,j)-th element of P; the one-step transition probability from cell i to cell j. Let p_{ij}^n be the (i,j)-th element of P^n; the n-step transition probability from cell i to cell j. If it is possible, through the mapping, to go from cell i to cell j, then we call cell i leads to cell j, symbolically $i \Rightarrow j$. Analytically, cell i leads to cell j if and only if there exists a positive integer m such that $p_{ij}^m > 0$. If cell i leads to cell j and cell j leads to cell i, then it is said that cell i communicates with cell j or cell j communicates with cell i. This will be denoted by $i \Leftrightarrow j$.

This system evolution of GCM leads to a homogeneous finite Markov chain which determines the long-term behavior of the system. To understand the properties of GCM based on the theory of Markov chain [HX99], some pertinent definitions are discussed in Section 3.3.3.

Let g be the total number of persistent groups in the system at a layer. Let B_i be an i-th persistent group or a set of persistent cells in i-th group where $B_i \subset Z_l$ and $i \in \{1, \ldots, g\}$. Thus, the set of all persistent groups or attractors is denoted as $\mathcal{B} = \{B_1, \ldots, B_g\}$. If a transient cell j leads to the i-th persistent group B_i, then we call B_i a *domicile* of cell j. A transient cell can have several domiciles. The definition of domiciles for the GCM method is provided below.

Definition 5.4.2 *(Single-Domicile and Multiple-Domicile): Those transient cells that have only one domicile are called single-domicile transient cells, and those that have more than one are called multiple-domicile transient cells.*

All the single-domicile, transient cells having one particular persistent group as their common domicile form the *domain of attraction* of that persistent group. A

multiple-domicile transient cell having two or more persistent groups as its domicile is a cell in the *boundary region* between the domains of attraction of these persistent groups. Transient cells are further divided into transient groups according to the number of domiciles they have. Let $B(j)$ where $j = \{1, 2, \ldots, g\}$ be the set of all single-domicile transient cells having j-th persistent group as its domicile. We call this j-th *single-domicile transient group*. It populates the domain of attraction of j-th persistent group. Let $B(i, j)$ where $i, j = \{1, 2, \ldots, g\}$, and $i < j$, be the set of all multiple-domicile transient cells having i-th and j-th persistent groups as their domiciles. We call this (i, j)-th *two-domicile transient group*. The region populated by this group is called the boundary regions of i-th and j-th domains of attractions. Hence, we define the set of transient groups at a layer as $\mathcal{T}$.

Algorithm 5.1: PERSISTENTBEHAVIOR(E_l, F_l)

Input: E_l, F_l – A 2-D layer of the environment, Corresponding vector field
Output: $\mathcal{B}$, $\mathcal{T}$, C, a – Set of persistent groups, Set of transient groups, Connectivity matrix, Expected absorption times of transient cells

$G.V \leftarrow \emptyset, \quad G.E \leftarrow \emptyset, \quad S \leftarrow \emptyset, \quad C \leftarrow \emptyset, \quad \mathcal{B} \leftarrow \emptyset, \quad \mathcal{L} \leftarrow \emptyset, \quad \mathcal{M} \leftarrow \emptyset$
for $i \leftarrow 1$ **to** N_l **do**
    $z \leftarrow i$
    $x, y \leftarrow$ CELLLOCATION(z)
    $x', y' \leftarrow$ MAPPEDCELL(x, y, F_l)
    $Z' \leftarrow$ MAPPEDCELLSET(x', y') `// Add uncertainty`
    $G.V \leftarrow G.V \cup Z' \cup \{z\}$
    $G.E \leftarrow G.E \cup \{(z, z') \mid z' \in Z'\}$
$S \leftarrow$ STRONGLYCONNECTEDCOMPONENT(G)
$C \leftarrow$ TRANSITIVECLOSURE(G)
$\mathcal{B} \leftarrow$ FINDPERSISTENTGROUPS(S, C)
$\mathcal{L} \leftarrow$ FINDUNION($\mathcal{B}$)
$\mathcal{M} \leftarrow Z_l \setminus \mathcal{L}$
$\mathcal{T} \leftarrow$ FINDTRANSIENTGROUPS($\mathcal{B}, \mathcal{M}, C$)
$a \leftarrow$ CALCULATEABSORPTIONTIME($\mathcal{M}$)
return $\mathcal{B}$, $\mathcal{T}$, C, a

First, we find the set of all persistent groups $\mathcal{B}$ and the set of all transient groups $\mathcal{T}$ from all the cells Z_l at the processing layer of the environment E_l. Let N_p be the total number of persistent cells at a layer and N_t be the total number of transient cells at a layer. Hence, the total number of cells at the processing layer N_l can be defined as $N_l = N_p + N_t$. Let $\mathcal{L}$ be the set of all persistent cells at a layer and $\mathcal{M}$ be the set of all transient cells at a layer. Therefore, the set of all cells at the processing layer of the environment can be specified as

$$Z_l = \mathcal{L} \cup \mathcal{M}. \tag{5.6}$$

In our approach, we correlate between the system evolution of GCM and directed graph theory. From this association, we find the properties of GCM. To achieve this, Algorithm 5.1 takes as input the geometric description of a 2-D layer of the environment E_l, and the corresponding vector field F_l. In Algorithm 5.1, we create a directed graph G without adding weights to the edges of G from the set of cells at the processing layer Z_l. Additionally, for each cell $z \in Z_l$, it finds the geographic location (x, y) (line 4). From this geographic location (x, y), it gets the location (x', y') of the next mapped cell as explained before (line 5). Taking the motion uncertainty into consideration, it computes the set of mapped cells Z' where $Z' \subset Z_l$ (line 6). All cells z, Z' are added to the set of vertices and their ordered pairs (z, z'), where $z' \in Z'$, are added the edges set of G (lines 7–8).

Next, we find the set of strongly connected components S from G using Tarjan's strongly connected component algorithm [Tar72] (line 9). We define the connectivity matrix as C and it is calculated from the transitive closure of G (line 10). From S and C, Algorithm 5.1 finds the set of g persistent groups $\mathcal{B}$ using the function FindPersistentGroups (line 11). In this function, if each vertex in a strongly connected component communicates to all other vertices in the strongly connected

component then this strongly connected component is found as a persistent group and each cell of this persistent group is classified as a persistent cell. Otherwise, each cell of this strongly connected component is classified as a transient cell. Taking the union of g persistent group sets, we get the set of all persistent cells $\mathcal{L}$ (line 12). Aside from all the persistent cells, the remaining cells from Z_l represent the set of transient cells $\mathcal{M}$ (line 13). Thus, it classifies all the cells Z_l in the processing layer of the environment E_l into the set of persistent cells $\mathcal{L}$ and the set of transient cells $\mathcal{M}$. To determine the set of single-domicile and multiple-domicile transient groups $\mathcal{T}$ at the processing layer using the function FindTransientGroups (line 14), we check if there is any path or connectivity from each transient cell to cells in all g persistent groups according to the connectivity matrix C of graph G.

To study the evolution of the system from transient cells, we create one sub-stochastic matrix Q, having order $N_t \times N_t$. This is a sub-stochastic matrix because the sum of every row is not equal to one. Based on the following theorem, we evaluate one property of transient cells that is the expected absorption time using Q.

Theorem 5.4.3 *(Isaacson and Madsen [IM76]): Let $Y = (I_t - Q)^{-1}$, where I_t is a unit matrix of order N_t. Then, the sum of the elements of the j-th row of Y gives the expected absorption time a_j of the j-th transient cell to be absorbed in the persistent groups, i.e.,*

$$a_j = \sum_{m=1}^{N_t} Y_{jm}, \tag{5.7}$$

where Y_{jm} denotes the (j,m)-th element of Y.

Physically, a_j provides the statistical time, it takes for the system to settle into its long-term stable motions if it starts from cell j. The expected absorption times of all transient cells a are calculated using the function CalculateAbsorptiontime

(line 15). Finally, Algorithm 5.1 returns the set of persistent groups or attractors $\mathcal{B}$, the set of transient groups $\mathcal{T}$, the connectivity matrix C, and the expected absorption times of transient cells a for the given layer. The set of persistent groups and the set of transient groups for the given layer represent the long-term behavior of the water flow at that layer.

5.4.4 Determining Deployment Locations of Drifters

For solving our first problem, we determine the initial deployment locations of drifters on the water surface E_1 and the minimum number of required drifters for total coverage from the long-term behavior of the water flow. Prior to this, we found the long-term behavior of the water flow on the surface E_1 at the top layer ($l = 0$) based on its vector field F_1 using Algorithm 5.1. Then, Algorithm 5.2 determines a random cell (location) as an initial deployment cell on the water surface in each group of the set of persistent groups $\mathcal{B}$ (line 3). It is determined in this way because each cell in a persistent group is connected to every other cell in that group. Algorithm 5.2 also selects a set of minimum number of transient cells $\mathcal{H}$ on the water surface for each transient group using the function MinCellsinTransientGroup (line 5). For each transient group, this function chooses $\mathcal{H}$ according to the descending order of the expected absorption times of transient cells in this group and the highest number of connected transient cells in the same transient group based on the connectivity matrix C. The cells in $\mathcal{H}$ are determined as the deployment cells (locations) for the associated transient group (line 6). In the end, Algorithm 5.2 returns the set of initial cells for the long-term deployment of drifters $\mathcal{C}$, and the cardinality of this set $|\mathcal{C}|$ defining the minimum number of drifters required for exploring the region represented by E_1.

Algorithm 5.2: DeploymentLocations($\mathcal{B}, \mathcal{T}, C, a$)

Input: $\mathcal{B}, \mathcal{T}, C, a$ – Set of persistent groups, Set of transient groups, Connectivity matrix, Expected absorption times of transient cells

Output: $\mathcal{C}$ – Set of cells on the water surface for deployment

```
1 C ← ∅
2 for i ← 1 to |B| do
3     C ← C ∪ {RandomCell(B_i)}
4 for j ← 1 to |T| do
5     H ← MinCellsinTransientGroup(T_j, C, a)
6     C ← C ∪ {H}
7 return C
```

5.4.5 Locating the Visibility-Based Deployment Locations for Drifters

For solving the second problem of interest, we first construct the complete 2-D visibility of a shallow seafloor environment through the visibility of the camera sensor of drifters. Then, we find the minimum deployment locations of the drifter on the water surface for the long-term visibility-based coverage of the seafloor environment. The visible circular region with a radius d of the seafloor environment is considered as the approximated 2-D visibility polygon for a cell z of all cells Z_l at the processing layer as illustrated in Figure 5.3. In our second problem, the processing layer is the top layer or the water surface (E_1) and its set of cells is Z_1.

To develop the long-term deployment policy for the 2-D visibility-based coverage of the seafloor environment E_L, Algorithm 5.3 takes as input a 2-D layer of the environment E_l, the visibility range d, and the set of persistent groups $\mathcal{B}$ from the long-term behavior of the system. It provides the minimum size subset of cells at the processing layer for the visibility-based coverage and the set of initial cells for the long-term deployment $\mathcal{C}$ as output. First, we initialize a set of visible cells $\mathcal{Y}$ that will contain the set of cells observed by all 2-D visibility polygons (line 1).

Therefore, we find the subset of cells that are completely visible by each 2-D visibility polygon $V(x_z)$ using the function VisibleCells and add this subset of cells to $\mathcal{Y}$ (lines 2–3). So, the set of visible cells $\mathcal{Y}$ is a collection of subsets of Z_l, such that every element of Z_l belongs to at least one subset of $\mathcal{Y}$, as follows:

$$Z_l = \bigcup_{\mathcal{S} \in \mathcal{Y}} \mathcal{S}. \tag{5.8}$$

For the drifter, it is an optimal solution to find the minimum size subset of cells $\mathcal{D} \subseteq \mathcal{Y}$, whose elements designated by visibility polygons (they can be overlapping) cover all the elements of Z_l at the processing layer as follows:

$$Z_l = \bigcup_{\mathcal{S} \in \mathcal{D}} \mathcal{S}. \tag{5.9}$$

However, the problem of finding the minimum size subset of cells $\mathcal{D}$ is an instance of the set-covering problem which is $\mathcal{NP}$-hard [CLRS01]. Also, our problem is similar to the set-covering problem, and it does not have an optimal solution using a polynomial-time algorithm.

Algorithm 5.3: Visibility-BasedDeployment($E_l, d, \mathcal{B}$)

Input: $E_l, d, \mathcal{B}$ – A 2-D layer of the environment, Visibility range, Set of persistent groups

Output: $\mathcal{D}, \mathcal{C}$ – Minimum size subset of cells for the visibility-based coverage, Set of cells for deployment

1 $\mathcal{Y} \leftarrow \emptyset$
2 **for** $i \leftarrow 1$ **to** N_l **do**
3 $\quad \mathcal{Y} \leftarrow \mathcal{Y} \cup \{\text{VisibleCells}(V(x_z))\}$
4 $\mathcal{D} \leftarrow \emptyset$
5 $\mathcal{U} \leftarrow Z_l$
6 **while** $\mathcal{U} \neq \emptyset$ **do**
7 $\quad$ select an $\mathcal{S} \in \mathcal{Y}$ that maximizes $|\mathcal{S} \cap \mathcal{U}|$
8 $\quad \mathcal{U} \leftarrow \mathcal{U} - \mathcal{S}$
9 $\quad \mathcal{D} \leftarrow \mathcal{D} \cup \{\mathcal{S}\}$
10 $\mathcal{C} \leftarrow \text{FindDeploymentCells}(\mathcal{D}, \mathcal{B})$
11 **return** $\mathcal{D}$, $\mathcal{C}$

Thus, we apply a greedy approximation set cover algorithm, and the best feasible solution we get is an $O(\log n)$ approximation solution [Chv79]. Let $\mathcal{D}$ be the minimum size subset of cells at the processing layer for visibility polygons that cover the seafloor environment completely by the drifter which is empty initially (line 4). At first, all the cells in Z_l at the processing layer are uncovered and the set of uncovered cells $\mathcal{U}$ is Z_l (line 5). For finding the minimum size subset of cells at the processing layer, it selects a subset $\mathcal{S} \in \mathcal{Y}$ that covers the maximum number of cells not yet covered (line 7). Let $\mathcal{U}$ maintain the set of remaining uncovered cells. At each iteration, we remove the covered cells by selected subset $\mathcal{S}$ from $\mathcal{U}$ and add the subset to $\mathcal{D}$ (lines 8–9). We continue this process until the set of uncovered cells becomes empty through the visibility. Thus, the minimum size subset of cells $\mathcal{D}$ is computed and these cells cover the seafloor environment completely through the visibility from the cells at the processing layer. We further reduce the number of locations or cells for the long-term deployment utilizing the function FindDeploymentCells. This function selects only the first cell of each persistent group and removes the others if $\mathcal{D}$ has more than one cells of this persistent group because the long-term trajectory of the deployed drifter will reach to other cells in the same persistent group. Thus, the set of initial cells $\mathcal{C}$ is calculated, and the cardinality of this set $|\mathcal{C}|$ defines the minimum number of locations. Eventually, Algorithm 5.3 returns $\mathcal{D}$ and $\mathcal{C}$ for visibility-based coverage of a seafloor environment through the long-term deployment of drifters on the water surface.

5.4.6 Calculating the Reachability of a Long-term Drifter Trajectory

To tackle our third problem, we consider a 3-D, multiple current layered marine environment $\mathcal{E}$. Let a set of vector fields at all current layers be $\mathcal{F}$. Algorithm 5.4 computes a set of all possible reachable locations $\mathcal{R}$ of a profiling drifter within different current layers of the environment. Algorithm 5.4 takes as input a 3-D environment $\mathcal{E}$, a set of vector fields $\mathcal{F}$, and an initial state or deployment location of the drifter on the water surface x_I. A computed component is the long-term behavior of the water flow for each current layer of the environment from Algorithm 5.1 (line 3). In this case, the vertical movements of the profiling drifter are considered only at the initial deployment location x_I. The corresponding initial state x_I at the lower layer is the projected location at that layer from x_I on the water surface. Then, we calculate all possible reachable locations (cells) at the processing layer using the function FindReachableCells from the connectivity matrix C and the corresponding initial state x_I and add these cells to $\mathcal{R}$ (line 4). Following a similar process, we calculate the set of all possible reachable locations $\mathcal{R}$ at all current layers from Algorithm 5.4.

Algorithm 5.4: PersistentReachability($\mathcal{E}, \mathcal{F}, x_I$)

Input: $\mathcal{E}, \mathcal{F}, x_I$ – A 3-D environment, Set of vector fields, Initial deployment location

Output: $\mathcal{R}$ – Set of all possible reachable locations

1 $\mathcal{R} \leftarrow \emptyset$
2 **for** $l \leftarrow 1$ **to** L **do**
3 $\quad \mathcal{B}, \mathcal{T}, C, a \leftarrow$ PersistentBehavior(E_l, F_l)
4 $\quad \mathcal{R} \leftarrow \mathcal{R} \cup \{$FindReachableCells$(C, x_I)\}$
5 **return** $\mathcal{R}$

Algorithm Analysis

The running time of the Algorithm 5.4 is $O(\mathcal{N})$, where $\mathcal{N}$ is the total number of cells in all current layers of a 3-D marine environment, since it iterates over all the cells $\mathcal{Z}$ at different layers of the environment, processing each cell exactly once.

5.4.7 Developing an Optimal Navigation Policy for a Drifter

To address our fourth problem, we consider a 3-D marine environment $\mathcal{E}$. A set of vector fields at all current layers $\mathcal{F}$ is generated from $\mathcal{E}$. The "drift" action (represented as 0) at each location is determined from these vector fields and the "up" (-1) and "down" (1) actions in different current layers are also included with the action set of a profiling drifter $U = \{-1, 1, 0\}$ for a location in the environment. We also consider a framework of a stochastic environment with fully-observable states which is known as a Markov Decision Process (MDP) [TBF05]. In an MDP, the policy $\pi : \mathcal{X} \to U$ maps from each state $x \in \mathcal{X}$ to a possible action $u \in U$ when the state is observable. The goal of the MDP framework is to identify the policy π that maximizes the potential reward. Hence, a reward function of the state and the action is denoted as r. For instance, the reward function for reaching the goal state x_G can be initialized as follows:

$$r(x, u) = \begin{cases} 100, & \text{if } u \text{ leads to } x_G, \\ -1, & \text{otherwise.} \end{cases} \tag{5.10}$$

Every policy has an associated value function, which measures the expected value of the policy. Let the value function be $\hat{V}$. Algorithm 5.5 initializes the value function $\hat{V}$ with $r_{\min}$, which represents the minimum possible immediate reward (lines 1–2). Then, it implements the recursive calculation of $\hat{V}$ using the value iteration method (lines 4–7). Once the value iteration converges after a number of

Algorithm 5.5: PolicyfromValueIteration $(\mathcal{E}, r, \gamma, x_G)$

Input: $\mathcal{E}, r, \gamma, x_G$ – A 3-D environment, Reward function, Discount factor, Goal location

Output: $\hat{V}, \pi$ – Value function, Optimal navigation policy

1 **for** $i \leftarrow 1$ **to** $\mathcal{N}$ **do**

2 $\quad \hat{V}_0(x_i) \leftarrow r_{\min}$

3 $k \leftarrow 0$

4 **while** $\hat{V}_k \neq \hat{V}_{k-1}$ **do**

5 $\quad k \leftarrow k + 1$

6 $\quad$ **for** $i \leftarrow 1$ **to** $\mathcal{N}$ **do**

7 $\quad\quad \hat{V}_k(x_i) \leftarrow \gamma \max_u \Big[r(x_i, u) + \sum_{j=1}^{\mathcal{N}} \hat{V}_{k-1}(x_j) p(x_j | x_i, u) \Big]$

8 **for** $i \leftarrow 1$ **to** $\mathcal{N}$ **do**

9 $\quad \pi(x_i) \leftarrow \operatorname{argmax}_u \Big[r(x_i, u) + \sum_{j=1}^{\mathcal{N}} \hat{V}_k(x_j) p(x_j | x_i, u) \Big]$

10 **return** $\hat{V}, \pi$

iterations k, the resulting value function $\hat{V}_k$ that maximizes the expected value of the function, induces the optimal navigation policy. The factor γ is the discount factor. The value iteration usually converges if $\gamma < 1$, and in some special cases, even for $\gamma = 1$. The final value function $\hat{V}_k$ after the convergence of the value iteration is the optimal value function. Thus, Algorithm 5.5 calculates the optimal navigation policy π from the optimal value function maximizing the expected reward for reaching the goal location (lines 8–9). This navigation policy π produces an optimal action from any location of the environment $\mathcal{E}$ to the given goal location x_G.

5.5 Results

5.5.1 Software Simulation

We validated our proposed approach using the ROMS [SM05] ocean current predictions for the SCB region. The two-dimensional ocean surface was considered as the simulation environment for drifter movements, and we discretized the ocean surface into a 2-D grid. The resolution of the grid was 21×29, which corresponds to approximately 1 km grid cells. A vector field from the ocean current predictions on the water surface was calculated and is shown in Figure 5.6(a). The flow lines of the current data were generated through the Euler numerical integration method from all locations of the water surface for a small time Δt which is the time to pass a cell. These flow lines are shown in Figure 5.6(b). We implemented the Algorithm 5.1 to study the long-term behavior of the water flow in the simulated environment. We found two persistent groups and three transient groups, where two of them are single-domicile transient groups and one is a multiple-domicile transient group. The result of the long-term behavior of the water flow is illustrated in Figure 5.7(a). The initial locations or cells for the long-term deployment of drifters from the result of the simulation of Algorithm 5.2 are shown with black circles in Figure 5.7(b). The minimum number of drifters required for exploring the environment $|\mathcal{C}|$ is 27. Each of the two persistent groups has only one deployment cell and needs one drifter to explore the specified region. Three transient groups have remaining 25 deployment cells and require 25 drifters to explore their designated regions. In all following figures, the gray region indicates the inaccessible land region.

We also calculated the expected absorption times for transient cells in the single-domicile and multiple-domicile transient groups. The result is shown in Figure 5.8, where the mean expected absorption times for transient cells in the multiple-domicile

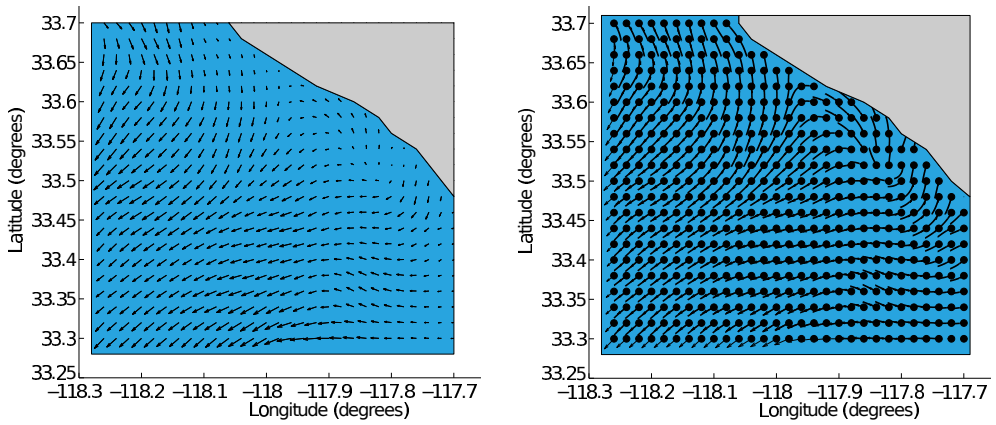
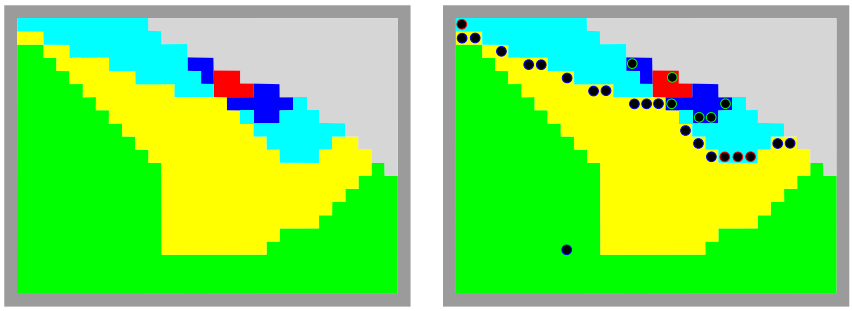


Figure 5.6: A Vector field and flow lines: (a) A 2-D Vector field for the water current of ocean-surface; (b) Flow lines of one step-size from all locations of ocean-surface.

transient group $B(1,2)$ are higher as compared with those of the transient cells in the single-domicile transient groups. Also, the transient cells of the single-domicile transient group of the first persistent group $B(1)$ take less time on average to get absorbed into their persistent group compared with the transient cells of the single-domicile transient group of the second persistent group $B(2)$.

We tested Algorithm 5.3 in simulation on the same environment and the visibility range $d = 6$ cell units. This simulation gave the minimum size subset of cells $\mathcal{D}$ of 16 cells on the water surface that covered the seafloor environment completely through the visibility. This subset $\mathcal{D}$ is illustrated in Figure 5.9(a). This $\mathcal{D}$ was further reduced to find the initial deployment locations or cells because the long-term deployment requires fewer drifters. This $\mathcal{D}$ has 9 persistent cells that are cells in the first persistent group of two groups calculated before. Keeping only the first persistent cell out of the 9 cells along with the remaining 7 transient cells, the simulation of Algorithm 5.3 provided the number of initial deployment locations or cells $|\mathcal{C}| = 8$ for the visibility-based coverage through the long-term trajectories of



(a) (b)

Figure 5.7: Simulation results of the long-term behavior of the water flow and the long-term deployment strategy of drifters: (a) Two persistent groups (the first one is composed of persistent cells in the green region and the second one is composed of persistent cells in the red region) and two single-domicile transient groups (the single-domicile transient group of the first persistent group consists of transient cells in the yellow region and the single-domicile transient group of the second persistent group consists of transient cells in the blue region.). Also, one multiple-domicile transient group of both persistent groups is shown through the cyan region where both persistent groups are the domiciles of transient cells in this region; (b) Initial deployment locations or cells (filled in black circles) for regions of these persistent and transient groups.

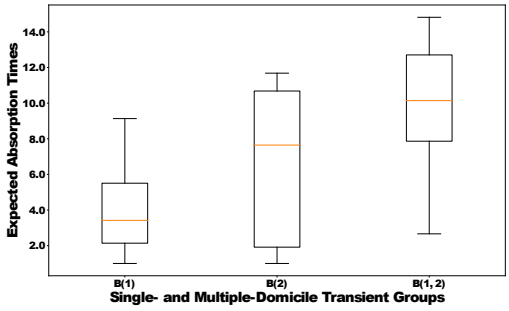
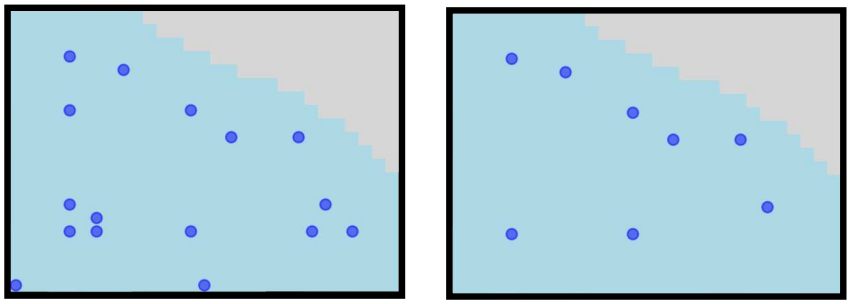


Figure 5.8: The expected absorption times for single- and multiple-domicile transient groups.

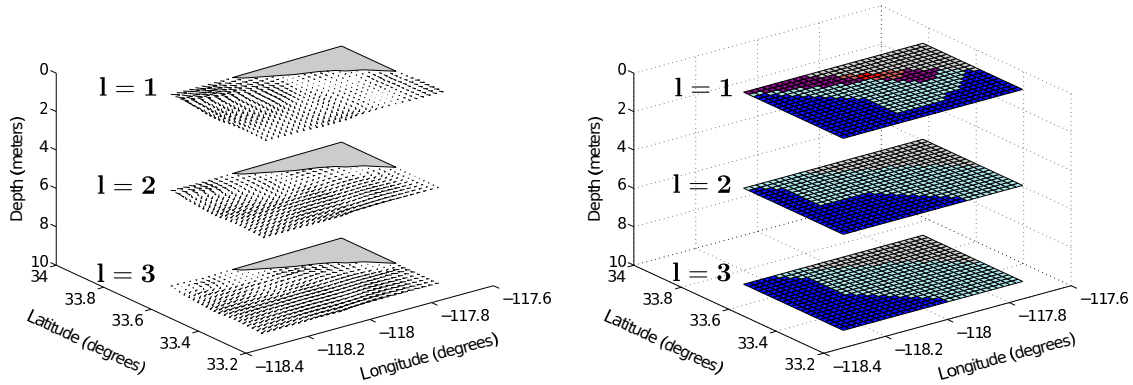
drifters. The set of initial deployment locations or cells $\mathcal{C}$ on the water surface are shown in Figure 5.9(b).

We implemented Algorithm 5.4 to study the persistent behavior of the water flow in a 3-D simulated environment consisting of three current layers for several



(a) (b)

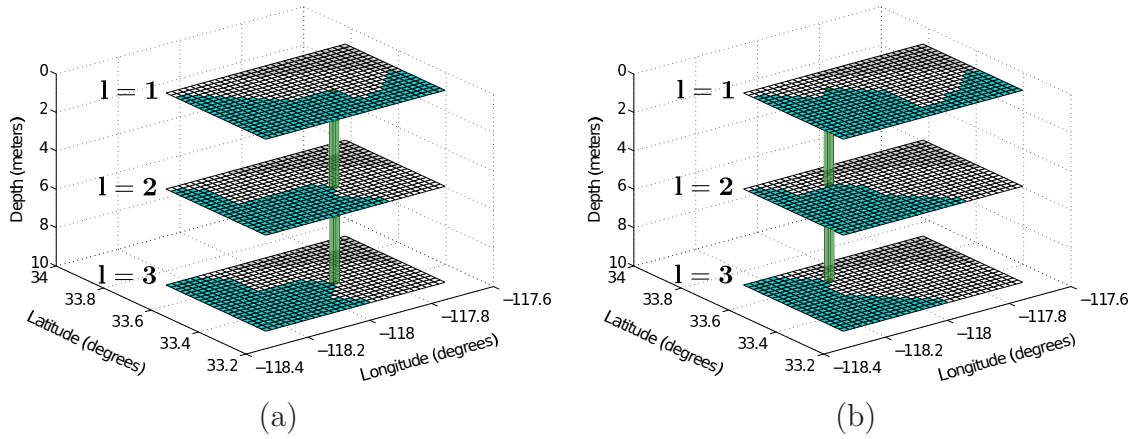
Figure 5.9: Simulation results of the long-term deployment policy for the visibility-based coverage: (a) The minimum size subset of cells (in blue filled circles) for the complete visibility (in the light blue region) of the environment; (b) The reduced set of cells (in blue filled circles) as the initial deployment locations or cells for drifters.



(a) (b)

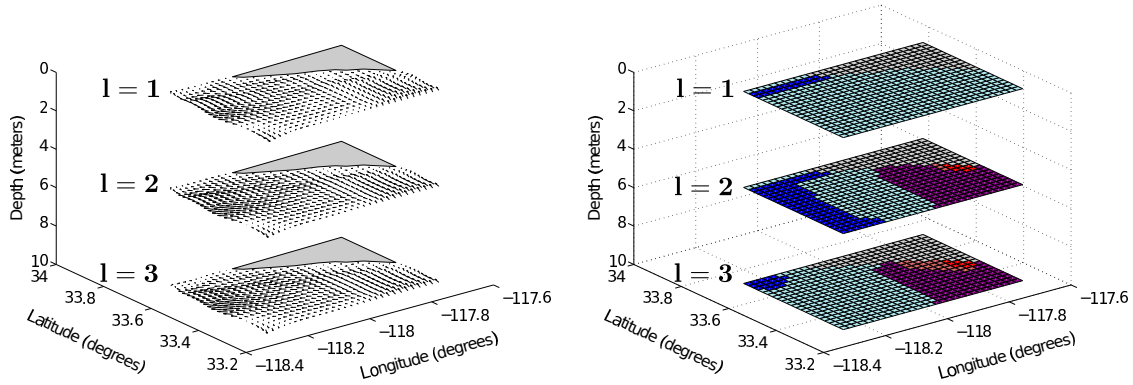
Figure 5.10: Persistent behavior results after time $t = 1$ hour: (a) Vector fields at three water layers; (b) Persistent groups (blue and red regions) and associated transient groups (the light blue region for the blue persistent group, the light red region for the red persistent group, and the purple region for both persistent groups) at the three current layers as the long-term behavior of the water flow.

times and initial deployment locations on the water surface. The resolution of each discretized 2-D layer of this environment is the same as considered before. We generated a set of vector fields at three current layers of the environment after time $t = 1$ hour, $t = 12$ hours, and $t = 24$ hours as generated above for the top layer of the environment. We found different numbers of persistent groups and their associated



(a) (b)

Figure 5.11: Long-term reachability results after time $t = 1$ hour: The light blue reachable locations at three current layers in the long run from the two green initial deployment locations of the drifter on the water surface.



(a) (b)

Figure 5.12: Persistent behavior results after time $t = 12$ hours: (a) Vector fields at three water layers; (b) Persistent groups (blue and red regions) and associated transient groups (the light blue region for the blue persistent group, the light red region for the red persistent group, and the purple region for both persistent groups) at the three current layers as the long-term behavior of the water flow.

transient groups after time $t = 1$ hour, $t = 12$ hours, and $t = 24$ hours at three current layers. The set of generated vector fields and the persistent behavior results after these times are illustrated in Figure 5.10, 5.12, and 5.14. We also calculated the set of all possible reachable locations at three current layers for two different

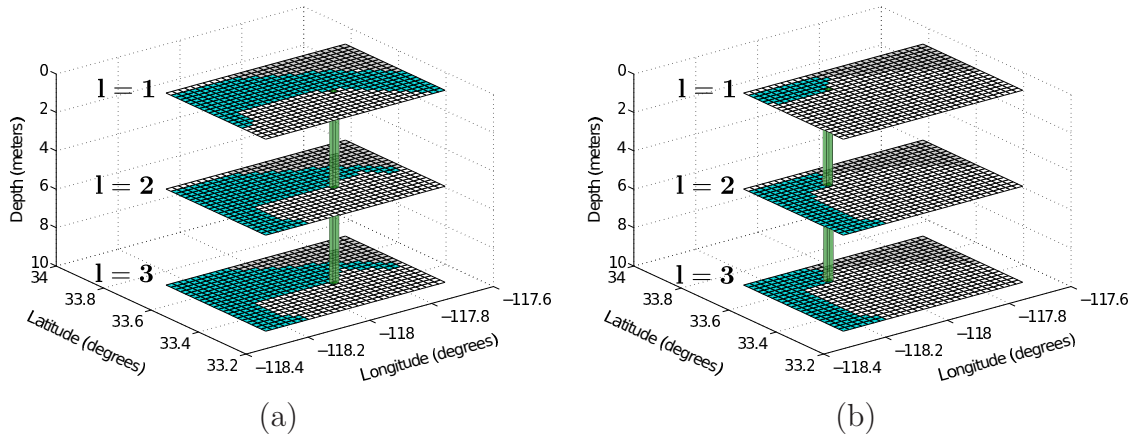


Figure 5.13: Long-term reachability results after time $t = 12$ hours: The light blue reachable locations at three current layers in the long run from the same green initial deployment locations of the drifter on the water surface.

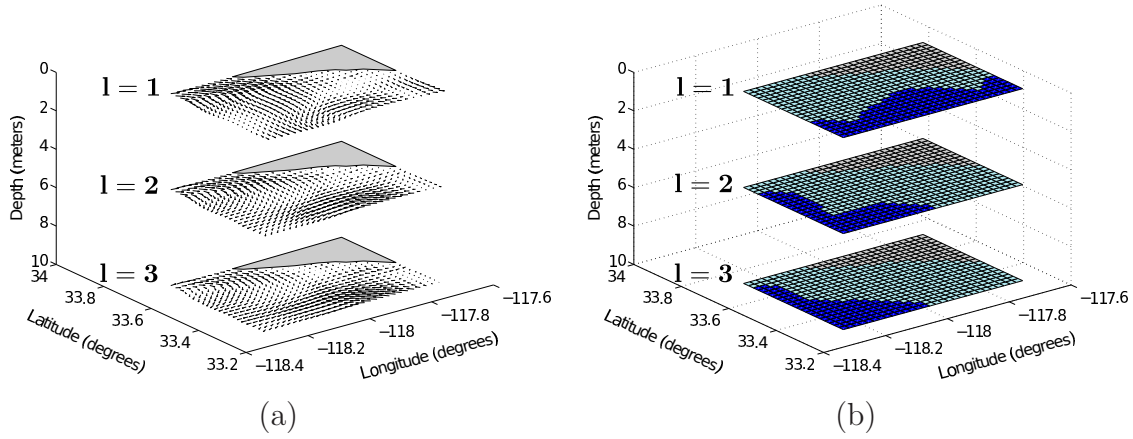


Figure 5.14: Persistent behavior results after time $t = 24$ hours: (a) Vector fields at three water layers; (b) Persistent groups (blue regions) and associated transient groups (the light blue region for the blue persistent group) at the three current layers as the long-term behavior of the water flow.

initial deployment locations of the drifter on the water surface after time $t = 1$ hour, $t = 12$ hours, and $t = 24$ hours which are shown in Figure 5.11, 5.13, and 5.15. In these figures, the vertical movements of the drifter are shown in green voxels at its initial deployment locations.

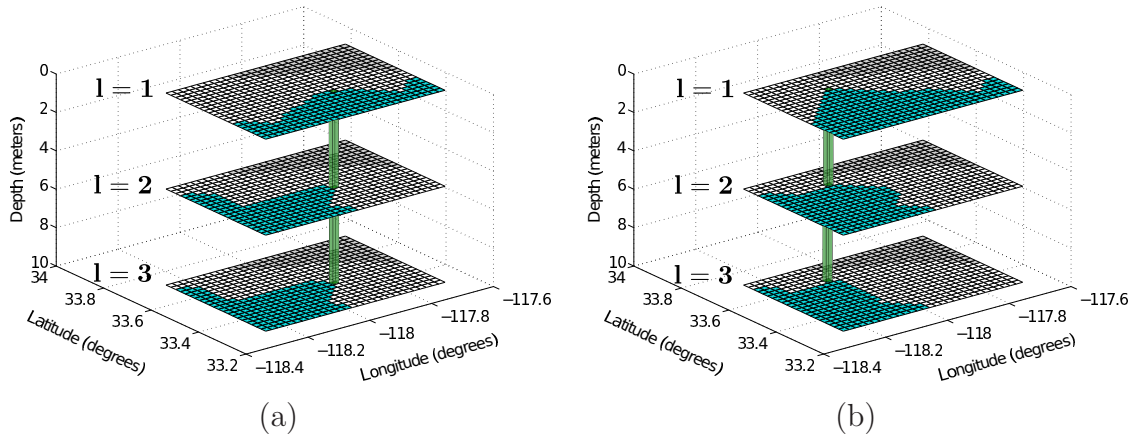


Figure 5.15: Long-term reachability results after time $t = 24$ hours: The light blue reachable locations at three current layers in the long run from the same green initial deployment locations of the drifter on the water surface.

We tested our implementation of Algorithm 5.5 on the 3-D simulated environment considered above with a goal location, as illustrated in Figure 5.16(a). We generated a set of vector fields for time $t = 1$ hour which is shown in Figure 5.10(a). Based on these vector fields, the action set, and the goal location, the optimal value function is estimated using the value iteration method. The optimal value function after the convergence of the value iteration method is demonstrated in Figure 5.16(b). The optimal navigation policy π is calculated based on the optimal value function making use of the MDP framework, which maximizes the expected reward for reaching the goal location. The optimal navigation policy is shown in Figure 5.16(c). This policy shows the best possible actions at all locations to reach the goal location that include "up", "down", and "drift" (at the same layer) movements.

5.6 Summary

In the final contribution, we presented a data-driven approach for solving the long-term deployment and planning problems using drifters in marine environments.

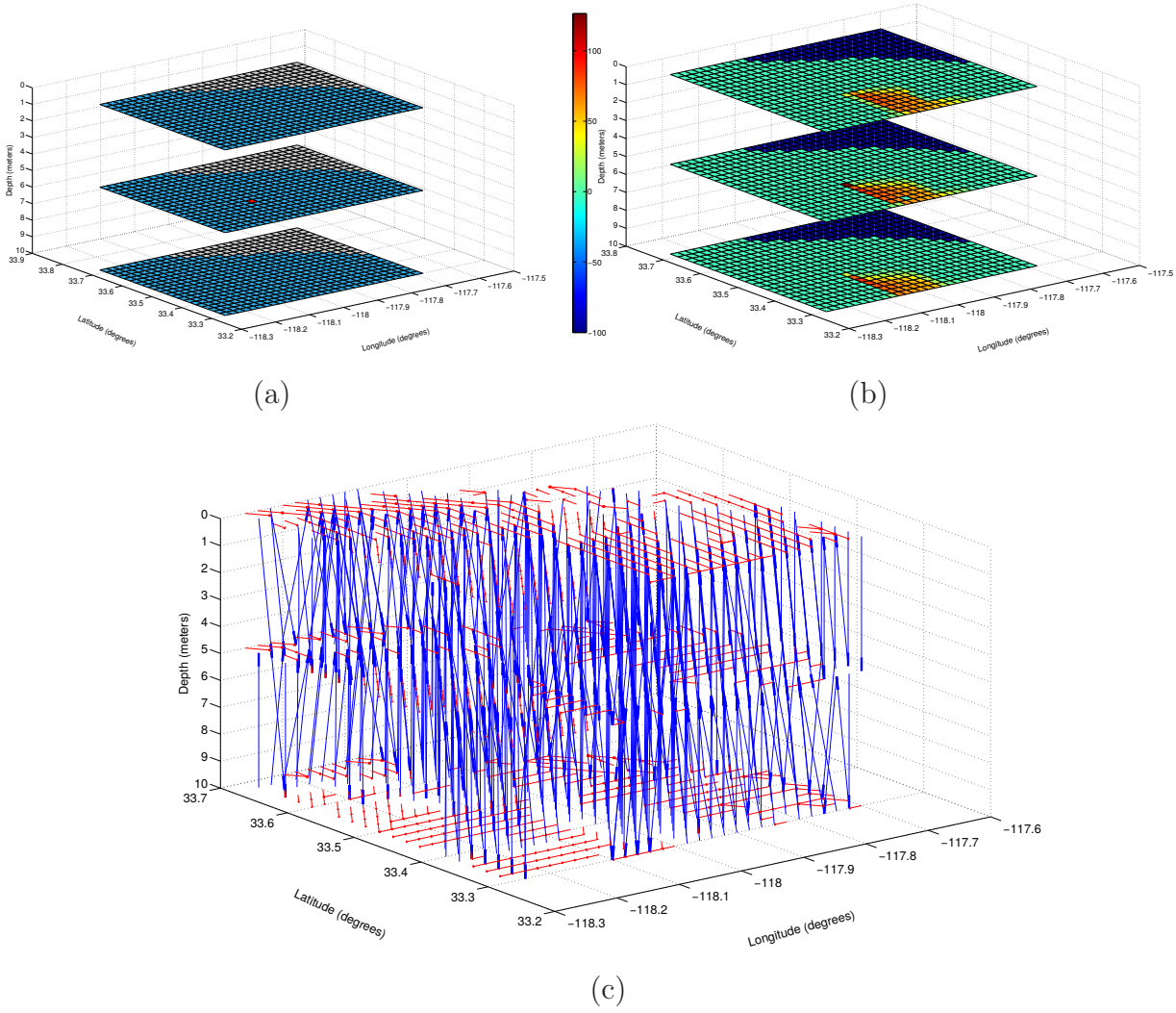


Figure 5.16: Optimal navigation policy result: (a) A 3-D environment with a red goal location; (b) The optimal value function after the value iteration convergence; (c) The optimal navigation policy showing active vertical actions with blue arrows and passive horizontal drifts with red arrows.

These deployed drifters can explore the whole water surface starting from minimum deployed locations and cover a seafloor environment through the visibility of the cameras of minimum deployed drifters while they traverse. We generated a vector field to understand the water flow on the 2-D water surface from a concatenated ROMS ocean model prediction data. From these data, the long-term behavior of the

system was studied employing the generalized cell mapping. Based on the long-term behavior of the system, the minimum deployment locations on the water surface for drifters were estimated for the whole water surface exploration and the visibility-based seafloor environment coverage. Our simulation results of the deployment of drifters showed the promising and potential application of our approach.

Further, we generated a set of vector fields in a 3-D marine environment from the ocean current prediction data and analyzed the persistent behavior of the 3-D environment. Based on the persistent behavior of the water flow, we also find all possible reachable locations at different current layers of the environment for the long-term trajectory of a profiling drifter starting from its initial deployment location. Our simulation results also show the variations both in the persistent behavior of the water flow and the long-term planning of drifter trajectory for different locations and times. We also developed an optimal navigation policy in the 3-D marine environment that generates the best possible action in the simulated policy from any location of the environment to the goal location considering the passive horizontal drift and the active vertical movements of the profiling drifter.

CHAPTER 6

DISCUSSION AND CONCLUSIONS

In this chapter, we present a brief summary of the dissertation, discuss open problems that remain to be solved, and outline some potential directions for future research.

6.1 Dissertation Summary

In this dissertation, we used simple robots, which we called bouncing robots, in many resource-constrained scenarios. Using the approaches in this dissertation, we made them capable of solving the fundamental robotic tasks such as localization, navigation, coverage, planning, patrolling, and deployment with limited sensing, actuation, computation, storage, and communication requirements. We utilized one dynamical system technique called the simple cell-to-cell mapping (SCM) to model the deterministic motion of the simple robots and another dynamical system technique called the generalized cell-to-cell mapping (GCM) to characterize the movement of the simple robots with uncertainty. These dynamical system techniques provide the attractors and domains of attraction from the system behavior which allowed us to develop the filters, controllers, and algorithms for the solutions to the tasks mentioned before. These solutions were verified on physical robots that enables them to operate in application domains that previously could not utilize autonomous systems in resource-constrained scenarios due to the task complexity. In our approach, we also optimized the properties of Markov chain which is the originality of our work. The defining aspect of this dissertation is its emphasis on using the dynamical system approach for simple system modeling which is unique in the robotics computing community. One striking feature of this dynamical system

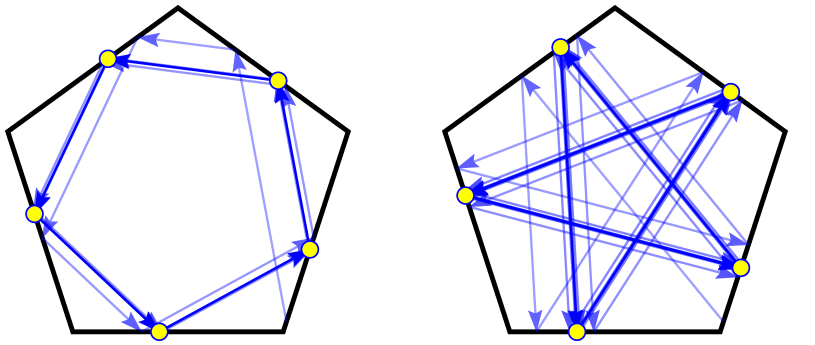
approach is that it is very general. As such, we can apply this approach in tackling problems to different domains as long as we have a consistent motion model of the robots. Consequently, the works presented in this dissertation leave possible avenues to solve several interesting open problems.

6.2 Open Problems

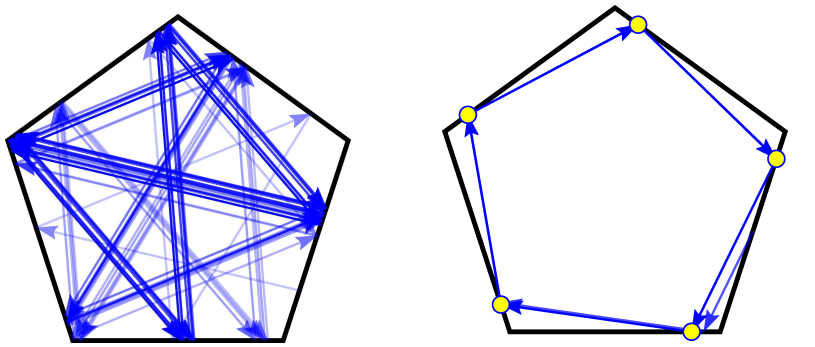
The open problems to solve using the dynamical system techniques and the simple robot behavior are discussed here.

6.2.1 Design and Planning for Simple Robots

One possible line of inquiry is how to solve design and planning problems for simple robots. In the planning problem, a robot will decide what to do to reach its goals. Using our dynamical system techniques, we find the properties of the system. Based on these properties, we can address the planning problem on the system with passive dynamics by tweaking the dynamics at specific points according to the associated costs. In this case, some notion of costs needs to be formulated too. In the design problem, robots can form a stable cycle around, for example, a crucial target area in the environment. We can also supplement the design to add some maximally useful additional modes of the dynamics. Furthermore, our concept based on limit cycles is similar to the concept of feedback motion planning with *funnels* presented in [Ted09, MT13, MT17] for simple underactuated systems. It will be an interesting problem to study our limit cycles with respect to funnels, which take a broad set of initial conditions to a goal region [BRK99].



(a) $\theta = 1.07$ rad, 20 bounces (b) $\theta = 0.57$ rad, 50 bounces



(c) $\theta = -0.08$ rad, 75 bounces (d) $\theta = -1.4$ rad, 20 bounces

Figure 6.1: Different behaviors of bounce trajectories in a regular pentagon [NBL17].

6.2.2 Comparison with Analytical Solutions

We have focused our attention on devising numerical solutions to diverse tasks for bouncing robots. A closely related analytical solution was recently proposed in [NBL17] to find periodic trajectories for bouncing robots. These bouncing trajectories in a regular polygonal environment are illustrated in Figure 6.1. We can connect our numerical results with this study of bouncing with geometric models. It is also an open problem how to characterize non-periodic dynamics.

6.2.3 Minimalist Communication Protocol

In the case of the resource-constrained robotics, it is another open problem to design a minimalist communication protocol for coordinating multiple robots. Once limited sensing and actuation requirements of simple mobile robots are met, they also need to minimize the amount of information that needs to be shared with them as power constraints can limit their communication capabilities. Therefore, a minimalist communication protocol has the potential for robustness even for a team of simple robots.

6.3 Future Directions and Extensions

In the future, we are interested in addressing some potential shortcomings of the approaches presented in this dissertation. Therefore, we conclude by mentioning future directions and extensions for each key contribution of this dissertation.

For the first contribution of this dissertation, we have several interesting directions for future work. Our localization filter does not always provide a single configuration (singleton) of the robot in the environment. We wish to reduce the uncertainty of the robot's configuration in the environment by further analyzing the structure of the localization filter. In order to do this, the robot may have to switch its bouncing angle and we can find the minimum number of changes to reduce the uncertainty. Since we discretize the state space and use a finite abstraction, our ideas connect naturally with *finite bisimulations* [VdS04], [TP03]. We assume that our cell-to-cell abstraction is deterministic and also that the system resets to the cell center in each step. Alternatively, we can model the nondeterministic cell-to-cell transitions by using GCM which uses a probabilistic cell transition map. We have implemented and used GCM to model the imperfect rotation of the robot for solving

the coverage problem [ABS17], and we believe that it can be used to account for the non-determinism in our localization method.

In the second contribution, it will be a more practical approach to incorporate a bounded uncertainty for both rotation and translation of the robots which will lead to the development of a probabilistic variant of our deterministic SCM based algorithm in the extended version of our navigation work. More reliable bouncing angles can be added to the set of bouncing angles for getting additionally minimal navigation plans. Nevertheless, it is an interesting problem to solve a complete navigation method for both rectilinear and non-rectilinear environments using the bouncing robot. Furthermore, we can do the more quantitive analysis of our coverage method in terms of efficiency by comparing with the boustrophedon coverage method for simple robots [CP98].

In the third contribution, further experimental tests and analytical techniques can be used to quantify the differences in performance between our approach and existing nondeterministic multi-robot patrolling methods. In this case, one area of interest is extending our ideas to 2.5D or 3D environments, which will extend the scope of applications of our work. Our estimation of randomized policies will work not only in planar graphs but also in graphs representing 2.5D and 3D environments. However, one challenge is the calculation of the visibility polyhedra [DDP02] in this case which is not a straightforward extension of our visibility polygon calculation. A potential solution is to approximate the visibility polyhedra with a simple shape (a sphere or a cone) or to use a 3D ray tracing approximation [BGZ07]. This will be one of our future extensions of this work. We also plan to solve the problem of dynamically reassigning patrolling policies to the robots when the central base station determines that one of the multiple robots is failing to patrol the environment. An approximation algorithm for dividing the environment into k regions can be in-

cluded in our extended work. Unidirectional visibility can be considered based on the orientation of a patrolling robot. We can model the adversary and apply game theory to find the optimal policy for a patroller competing with that adversary.

In the final contribution, our proposed approach seems to have a relation to Lagrangian Coherent Structures (LCS) [SLM05]. We will investigate this relationship for further applications or extensions of the research presented in this work. Physical deployments are being planned to acquire a larger spatiotemporal dataset for drifting vehicles. Additionally, we are examining the use of data gathered by the network of Argo floats [RJR+09] as a proxy larger dataset to develop methods for learning predictive models and flow fields for use with our proposed methods. We will also find the stochastic shortest path between two locations of an environment using the motion model of drifters which will give us the predicted trajectory of a drifter from its initial deployment location to a goal location. Some autonomous surface vehicles (ASVs) should retrieve the information collected by drifters in the long-term deployments. The drifters are able to communicate with other vehicles and transmit information. In this scenario, we can design a communication network that will include realistic parameters that affect the communication quality between a transmitter of the drifter and a receiver of the ASV, for example: a) the distance between two components, and b) the presence of obstacles.

BIBLIOGRAPHY

[ABS17] Tauhidul Alam, Leonardo Bobadilla, and Dylan A Shell. Minimalist robot navigation and coverage using a dynamical system approach. In *Proceedings of the IEEE International Conference on Robotic Computing (IRC)*, pages 249–256, 2017. 40, 48, 49, 132, 170

[ABS18] Tauhidul Alam, Leonardo Bobadilla, and Dylan A Shell. Space-efficient filters for mobile robot localization from discrete limit cycles. *IEEE Robotics and Automation Letters*, 3(1):257–264, 2018. 20, 49, 52

[ACO04] Alberto Alvarez, Andrea Caiti, and Reiner Onken. Evolutionary path planning for autonomous underwater vehicles in a variable ocean. *IEEE Journal of Oceanic Engineering*, 29(2):418–429, 2004. 134

[ÅE71] Karl Johan Åström and Peter Eykhoff. System identification–a survey. *Automatica*, 7(2):123–162, 1971. 5

[AEBS15] Tauhidul Alam, Matthew Edwards, Leonardo Bobadilla, and Dylan Shell. Distributed multi-robot area patrolling in adversarial environments. In *the International Workshop on Robotic Sensor Networks (RSN)*, 2015. 86

[Agm10] Noa Agmon. On events in multi-robot patrol in adversarial environments. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 591–598, 2010. 87

[AHK08] Noa Agmon, Noam Hazon, and Gal A Kaminka. The giving tree: constructing trees for efficient offline and online multi-robot coverage. *Annals of Mathematics and Artificial Intelligence*, 52(2):143–168, 2008. 51

[AKK08] Noa Agmon, Sarit Kraus, and Gal A Kaminka. Multi-robot perimeter patrol in adversarial settings. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2339–2345, 2008. xvi, 87, 89, 110, 111

[AKK11] Noa Agmon, Gal A Kaminka, and Sarit Kraus. Multi-robot adversarial patrolling: facing a full-knowledge opponent. *Journal of Artificial Intelligence Research*, 42:887–916, 2011. 51, 87

[Ala16] Tauhidul Alam. Decentralized and nondeterministic multi-robot area patrolling in adversarial environments. *International Journal of Computer Applications*, 156(2):1–8, 2016. 86

[AM98] Srinivas Akella and Matthew T Mason. Posing polygonal objects in the plane by pushing. *International Journal of Robotics Research*, 17(1):70–88, 1998. 10

[AmCA12] Ali-akbar Agha-mohammadi, Suman Chakravorty, and Nancy M Amato. On the probabilistic completeness of the sampling-based feedback motion planners in belief space. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3983–3990, 2012. 50

[ARBR17] Tauhidul Alam, Md Mahbubur Rahman, Leonardo Bobadilla, and Brian Rapp. Multi-vehicle patrolling with limited visibility and communication constraints. In *Proceedings of the IEEE Conference on Military Communications (MILCOM)*, pages 465–470, 2017. 86, 132

[ARBS18] Tauhidul Alam, Gregory Murad Reis, Leonardo Bobadilla, and Ryan N Smith. A data-driven deployment approach for persistent monitoring in aquatic environments. In *Proceedings of the IEEE International Conference on Robotic Computing (IRC)*, pages 147–154, 2018. 129

[AS06] Mazda Ahmadi and Peter Stone. A multi-robot system for continuous area sweeping tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1724–1729, 2006. 88

[BC12] Nicola Basilico and Stefano Carpin. Online patrolling using hierarchical spatial representations. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2163–2169, 2012. 89

[Bel12] Mathieu Belbeoch. Argo: Part of the integrated global observation strategy, 2012. Available at http://www.argo.ucsd.edu/, 2018. 134

[BFMI+15] Daniel Boydstun, Matthew Farich, John McCarthy III, Silas Rubinson, Zachary Smith, and Ioannis Rekleitis. Drifter sensor network for environmental monitoring. In *Proceedings of the 12th Conference on Computer and Robot Vision (CRV)*, pages 16–22, 2015. 129, 133

[BGA09] Nicola Basilico, Nicola Gatti, and Francesco Amigoni. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 57–64, 2009. 89

[BGA12] Nicola Basilico, Nicola Gatti, and Francesco Amigoni. Patrolling security games: Definition and algorithms for solving large instances with single patroller and single intruder. *Artificial Intelligence*, 184:78–123, 2012. 89

[BGZ07] Chao-ying Bai, Stewart Greenhalgh, and Bing Zhou. 3D ray tracing using a modified shortest-path method. *Geophysics*, 72(4):T27–T36, 2007. 170

[BKS10] Jonathan Binney, Andreas Krause, and Gaurav S Sukhatme. Informative path planning for an autonomous underwater vehicle. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4791–4796, 2010. 133

[BKS13] Jonathan Binney, Andreas Krause, and Gaurav S Sukhatme. Optimizing waypoints for monitoring spatiotemporal phenomena. *International Journal of Robotics Research*, 32(8):873–888, 2013. 133

[Boy06] Stephen Boyd. Convex optimization of graph laplacian eigenvalues. In *Proceedings of the International Congress of Mathematicians*, volume 3, pages 1311–1319, 2006. 88

[BRK99] Robert R Burridge, Alfred A Rizzi, and Daniel E Koditschek. Sequential composition of dynamically dexterous robot behaviors. *International Journal of Robotics Research*, 18(6):534–555, 1999. 22, 167

[BSC+12] Leonardo Bobadilla, Oscar Sanchez, Justin Czarnowski, Katrina Gossman, and Steven M LaValle. Controlling wild bodies using linear temporal logic. In *Proceedings of the Robotics: Science and Systems (RSS)*, 2012. 23

[BTI11] Deepak Bhadauria, Onur Tekdas, and Volkan Isler. Robotic data mules for collecting data over sparse sensor fields. *Journal of Field Robotics*, 28(3):388–404, 2011. 87

[CD17] Alin-Mihai Căilean and Mihai Dimian. Current challenges for visible light communications usage in vehicle applications: A survey. *IEEE Communications Surveys & Tutorials*, 19(4):2681–2703, 2017. 128

[CG94] JF Canny and KY Goldberg. "RISC" industrial robots: Recent results and current trends. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1951–1958, 1994. 10

[CG95] John F Canny and Kenneth V Goldberg. A RISC approach to sensing and manipulation. *Journal of Field Robotics*, 12(6):351–363, 1995. 10

[Cho01] Howie Choset. Coverage for robotics–a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31(1):113–126, 2001. 6, 49

[Chv79] Vasek Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979. 104, 153

[CJN99] Svante Carlsson, Håkan Jonsson, and Bengt J Nilsson. Finding the shortest watchman route in a simple polygon. *Discrete & Computational Geometry*, 22(3):377–402, 1999. 87

[CLC14] Chiu-Hung Chen, Tung-Kuan Liu, and Jyh-Horng Chou. A novel crowding genetic algorithm and its applications to manufacturing robots. *IEEE Transactions on Industrial Informatics*, 10(3):1705–1716, 2014. 1

[CLRS01] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2001. 61, 104, 152

[CMN+92] Kevin P Carroll, Stephen R McClaran, Eric L Nelson, David M Barnett, Donald K Friesen, and Glen N William. AUV path planning: an A* approach to path planning with consideration of variable vehicle speeds and multiple, overlapping, time-dependent exclusion zones. In *Proceedings of the Symposium on Autonomous Underwater Vehicle Technology (AUV)*, pages 79–84, 1992. 134

[CN88] Wei-pang Chin and Simeon Ntafos. Optimum watchman routes. *Information Processing Letters*, 28(1):39–44, 1988. 88

[Con85] Cl Connolly. The determination of next best views. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 432–435, 1985. 132

[CP98] Howie Choset and Philippe Pignon. Coverage path planning: The boustrophedon cellular decomposition. In *Proceedings of the International Conference on Field and Service Robotics (FSR)*, pages 203–209, 1998. 49, 170

[CSR04] Yann Chevaleyre, Francois Sempe, and Geber Ramalho. A theoretical analysis of multi-agent patrolling strategies. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1524–1525, 2004. 88

[DDP02] Frédo Durand, George Drettakis, and Claude Puech. The 3D visibility complex. *ACM Transactions on Graphics*, 21(2):176–206, 2002. 170

[DGA00] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and computing*, 10(3):197–208, 2000. 10

[DRDW92] R E Davis, L A Regier, J Dufour, and D C Webb. The Autonomous Lagrangian Circulation Explorer (ALACE). *Journal of Atmospheric and Oceanic Technology*, 9(3):264–285, 1992. 130, 134

[droa] Commerical drone applications on the rise. http://dronenodes.com/commercial-drone-applications/. 4

[drob] Department of homeland security built domestic surveillance tech into predator drones. https://www.cnet.com/news/dhs-built-domestic-surveillance-tech-into-predator-drones/. xiii, 2, 3

[EAK07] Yehuda Elmaliach, Noa Agmon, and Gal A Kaminka. Multi-robot area patrol under frequency constraints. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 385–390, 2007. 88

[EGA81] Hossam El Gindy and David Avis. A linear algorithm for computing the visibility polygon from a point. *Journal of Algorithms*, 2(2):186–197, 1981. 91, 139

[EKOL08] Lawrence H Erickson, Joseph Knuth, Jason M O'Kane, and Steven M LaValle. Probabilistic localization with a blind robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1821–1827, 2008. 12, 21, 22, 49

[EL13] Lawrence H Erickson and Steven M LaValle. Toward the design and analysis of blind, bouncing robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3233–3238, 2013. 12, 21, 22

[EM88] Michael A Erdmann and Matthew T Mason. An exploration of sensorless manipulation. *IEEE Journal on Robotics and Automation*, 4(4):369–379, 1988. 10

[FBT99] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999. 20

[FDM13] Pooyan Fazli, Alireza Davoodi, and Alan K Mackworth. Multi-robot repeated area coverage. *Autonomous robots*, 34(4):251–276, 2013. 52

[FDPM10] Pooyan Fazli, Alireza Davoodi, Philippe Pasquier, and Alan K Mackworth. Complete and robust cooperative robot area coverage with limited range. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5577–5582, 2010. 52, 90

[FKML88] Marshall Freimer, Georgia Kollia, Govind S Mudholkar, and C Thomas Lin. A study of the generalized tukey lambda family. *Communications in Statistics-Theory and Methods*, 17(10):3547–3567, 1988. 109

[FM12] Pooyan Fazli and Alan K Mackworth. The effects of communication and visual range on multi-robot repeated boundary coverage. In *Proceedings of the IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–8, 2012. 52

[Fox03] Dieter Fox. Adapting the sample size in particle filters through KLD-sampling. *International Journal of Robotics Research*, 22(12):985–1003, 2003. 12, 20

[GAO05] Bartolome Garau, Alberto Alvarez, and Gabriel Oliver. Path planning of autonomous underwater vehicles in current fields with complex spatial

variability: an A* approach. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 194–198, 2005. 134

[GBS08] Arpita Ghosh, Stephen Boyd, and Amin Saberi. Minimizing effective resistance of a graph. *SIAM Review*, 50(1):37–66, 2008. 15, 88, 98

[GBY08] Michael Grant, Stephen Boyd, and Yinyu Ye. CVX: Matlab software for disciplined convex programming, 2008. Available at http://cvxr.com/cvx/examples/. 112, 116

[GC13] Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258–1276, 2013. 6, 51

[GDS04] Elena Garcia and P Gonzalez De Santos. Mobile-robot navigation with complete coverage of unstructured environments. *Robotics and Autonomous Systems*, 46(4):195–204, 2004. 51

[GJK+12] Christopher R German, Micheal V Jakuba, James C Kinsey, Jim Partan, Stefano Suman, Abhimanyu Belani, and Dana R Yoerger. A long-term vision for long-range ship-free deep ocean operations: Persistent presence through coordination of autonomous surface vehicles and autonomous underwater vehicles. In *Proceedings of the IEEE/OES Autonomous Underwater Vehicles (AUV)*, pages 1–7, 2012. 134

[GKKP06] Ben Grocholsky, James Keller, Vijay Kumar, and George Pappas. Cooperative air and ground surveillance. *IEEE Robotics & Automation Magazine*, 13(3):16–25, 2006. 1

[gli] YSI ecomapper autonomous underwater vehicle (AUV). http://auvac.org/configurations/view/105. xiii, 3, 4

[GR01] Yoav Gabriely and Elon Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):77–98, 2001. 51

[Gus11] Bertil Gustafsson. Least square problems. In *Fundamentals of Scientific Computing*, pages 125–133. 2011. 67

[Hau10] Kris Hauser. Randomized belief-space replanning in partially-observable continuous spaces. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics (WAFR-IX)*, pages 193–209. 2010. 50

[HDS14] Van T Huynh, Matthew Dunbabin, and Ryan N Smith. Convergence-guaranteed time-varying RRT path planning for profiling floats in 4-Dimensional flow. In *Proceedings of the Australasian Conference on Robotics and Automation (ACRA)*, pages 1–9, 2014. 130, 139

[HH79] John E Hopcroft and John E Hopcroft. *Introduction to automata theory, languages, and computation*. Addison Wesley, MA, first edition, 1979. 33, 34

[HK08] Noam Hazon and Gal A Kaminka. On redundancy, efficiency, and robustness in coverage for multiple robots. *Robotics and Autonomous Systems*, 56(12):1102–1114, 2008. 51

[hr] Toyota third generation humanoid robot T-HR3. https://newsroom.toyota.co.jp/en/detail/19666346/. xiii, 3, 4

[Hsu80] Chieh Su Hsu. A theory of cell-to-cell mapping dynamical systems. *Journal of Applied Mechanics*, 47(4):931–939, 1980. 11, 12, 14, 25, 49

[Hsu13] Chieh Su Hsu. *Cell-to-cell mapping: a method of global analysis for nonlinear systems*, volume 64. Springer Science & Business Media, 2013. 11, 21, 26, 55, 56, 58, 132, 145

[HX99] Ling Hong and Jianxue Xu. Crises and chaotic transients studied by the generalized cell mapping digraph method. *Physics Letters A*, 262(4):361–375, 1999. 56, 145, 146

[IKK05] Volkan Isler, Sampath Kannan, and Sanjeev Khanna. Randomized pursuit-evasion in a polygonal environment. *IEEE Transactions on Robotics*, 21(5):875–884, 2005. 90

[IM76] Dean L Isaacson and Richard W Madsen. *Markov chains, theory and applications*, volume 4. Wiley New York, 1976. 149

[iro] iRobot Create 2 programmable robot. http://store.irobot.com/default/create-programmable-programmable-robot-irobot-create-2/RC65099.html?cgid=us. xiii, 8

[JDH+06] Juan C Juarez, Anurag Dwivedi, A Roger Hammons, Steven D Jones, Vijitha Weerackody, and Robert A Nichols. Free-space optical communications for next-generation military networks. *IEEE Communications Magazine*, 44(11):46–51, 2006. 128

[JLL+09] Yeun-Soo Jung, Kong-Woo Lee, Seong-Yong Lee, Myoung Hwan Choi, and Beom-Hee Lee. An efficient underwater coverage method for multi-AUV with sea current disturbances. *International Journal of Control, Automation and Systems*, 7(4):615–629, 2009. 133

[JLV99] Leopoldo Jetto, Sauro Longhi, and Giuseppe Venturini. Development and experimental validation of an adaptive extended Kalman filter for the localization of mobile robots. *IEEE Transactions on Robotics and Automation*, 15(2):219–229, 1999. 20

[JZZ13] Jerome Jouffroy, Qiuyang Zhou, and Oliver Zielinski. On active current selection for Lagrangian profilers. *Modeling, Identification and Control*, 34(1):1–10, 2013. 130, 134

[Kal60] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960. 10

[KC09] Myungsik Kim and Nak Young Chong. Direction sensing RFID reader for mobile robot navigation. *IEEE Transactions on Automation Science and Engineering*, 6(1):44–54, 2009. 50

[KDK16] Robert King, Ben Dean, and Sigbert Klinke. gld: Estimation and use of the generalised tukey lambda distribution. *R package version*, 2(4), 2016. Available at https://cran.r-project.org/web/packages/gld/. 122

[KKLS10] Andreas Kolling, Alexander Kleiner, Michael Lewis, and Katia Sycara. Pursuit-evasion in 2.5 d based on team-visibility. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4610–4616, 2010. 138

[KS12] Shawn M Kristek and Dylan A Shell. Orienting deformable polygonal parts without sensors. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 973–979, 2012. 23

[KSBB07] Dov Kruger, Rustam Stolkin, Aaron Blum, and Joseph Briganti. Optimal AUV path planning for extended missions in complex, fast-flowing

estuarine environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4265–4270, 2007. 134

[kuk] Kuka industrial robots. https://www.kuka.com/en-us/products/robotics-systems/industrial-robots. xiii, 3, 4

[LA06] Jyh-Ming Lien and Nancy M Amato. Approximate convex decomposition of polygons. *Computational Geometry*, 35(1-2):100–123, 2006. 107

[LaV06] Steven M LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/. 12, 20, 28, 54

[LaV13] Steven M LaValle. *Mobile Robotics–An Information Space Approach*. 2013. Available at http://msl.cs.uiuc.edu/mobile/. 5

[LCMM98] Gérard Loaec, Norbert Cortes, Martin Menzel, and Jacky Moliera. PROVOR: a hydrographic profiler based on MARVOR technology. In *Proceedings of the MTS/IEEE OCEANS*, volume 1, pages 42–45, 1998. 134

[LDW91] John J Leonard and Hugh F Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, 1991. 12, 20

[Lee61] Chin Yang Lee. An algorithm for path connections and its applications. *IRE Transactions on Electronic Computers*, EC-10(3):346–365, 1961. 126

[LIWY09] Ming Li, Kenji Imou, Katsuhiro Wakabayashi, and Shinya Yokoyama. Review of research on agricultural vehicle autonomous guidance. *International Journal of Agricultural and Biological Engineering*, 2(3):1–16, 2009. 1

[LL95] Anthony Lazanas and J-C Latombe. Landmark-based robot navigation. *Algorithmica*, 13(5):472–501, 1995. 50

[LL14] Tae-Seok Lee and Beom Hee Lee. A new hybrid terrain coverage method for underwater robotic exploration. *Journal of Marine Science and Technology*, 19(1):75–89, 2014. 133

[LO10] Jeremy S Lewis and Jason M O'Kane. Guaranteed navigation with an unreliable blind robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 5519–5524, 2010. 49, 50

[LO12] Jeremy S Lewis and Jason M O'Kane. Reliable indoor navigation with an unreliable robot: Allowing temporary uncertainty for maximum mobility. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 160–165, 2012. 50

[LO13] Jeremy S Lewis and Jason M OKane. Planning for provably reliable navigation using an unreliable, nearly sensorless robot. *International Journal of Robotics Research*, 32(11):1342–1357, 2013. 22, 50

[LRM+16] Alberto Quattrini Li, Ioannis Rekleitis, Sandeep Manjanna, Nikhil Kakodkar, Johanna Hansen, Gregory Dudek, Leonardo Bobadilla, Jacob Anderson, and Ryan N Smith. Data correlation and comparison from multiple sensors over a coral reef with a team of heterogeneous aquatic robots. In *Proceedings of the International Symposium on Experimental Robotics (ISER)*, pages 717–728, 2016. 129, 133, 137

[Mas93] Matthew T Mason. Kicking the sensing habit. *AI Magazine*, 14(1):58–59, 1993. 10

[MBS15] Artem Molchanov, Andreas Breitenmoser, and Gaurav S Sukhatme. Active drifters: Towards a practical multi-robot system for ocean monitoring. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 545–552, 2015. 129

[MFK11] Nathan Michael, Jonathan Fink, and Vijay Kumar. Cooperative manipulation and transportation with aerial robots. *Autonomous Robots*, 30(1):73–86, 2011. 1

[MKMD16] Sandeep Manjanna, Nikhil Kakodkar, Malika Meghjani, and Gregory Dudek. Efficient terrain driven coral coverage using Gaussian processes for mosaic synthesis. In *Proceedings of the 13th Conference on Computer and Robot Vision (CRV)*, pages 448–455, 2016. 133

[MLS16] Kai-Chieh Ma, Lantao Liu, and Gaurav S Sukhatme. An information-driven and disturbance-aware planning method for long-term ocean monitoring. In *Proceedings of the IEEE/RSJ International Conference*

on Intelligent Robots and Systems (IROS), pages 2102–2108, 2016. 129, 133

[MLS17] Kai-Chieh Ma, Lantao Liu, and Gaurav S Sukhatme. Informative planning and online learning with sparse Gaussian processes. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4292–4298, 2017. 132

[MSZ09] Fulvio Mastrogiovanni, Antonio Sgorbissa, and Renato Zaccaria. Robust navigation in an unknown environment with minimal sensing and representation. *IEEE Transactions on Systems, Man, and Cybernetics*, 39(1):212–229, 2009. 49

[MT13] Anirudha Majumdar and Russ Tedrake. Robust online motion planning with regions of finite time invariance. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics (WAFR-X)*, pages 543–558, 2013. 167

[MT17] Anirudha Majumdar and Russ Tedrake. Funnel libraries for real-time robust feedback motion planning. *International Journal of Robotics Research*, 36(8):947–982, 2017. 167

[MVH14] Sarah Mennicken, Jo Vermeulen, and Elaine M Huang. From today's augmented houses to tomorrow's smart homes: new directions for home automation research. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 105–115, 2014. 1

[NBL17] Alexandra Q Nilles, Israel Becerra, and Steven M LaValle. Periodic trajectories of mobile robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3020–3026, 2017. xix, 168

[NNF10] Basilico Nicola, Gatti Nicola, and Villa Federico. Asynchronous multi-robot patrolling against intrusions in arbitrary topologies. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2010. 89

[OC08] Karl J Obermeyer and Contributors. The visibility library, 2008. Available at http://www.visilibity.org/. 103, 115

[OL05] Jason M O'Kane and Steven M LaValle. Almost-sensorless localization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3764–3769, 2005. 21

[OL07] Jason M O'Kane and Steven M LaValle. Localization with limited sensing. *IEEE Transactions on Robotics*, 23(4):704–716, 2007. 12, 21, 22, 49

[OLCJ14] Ruofei Ouyang, Kian Hsiang Low, Jie Chen, and Patrick Jaillet. Multi-robot active sensing of non-stationary Gaussian process-based environmental phenomena. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 573–580, 2014. 132

[OS13] Jason M O'Kane and Dylan A Shell. Automatic reduction of combinatorial filters. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4082–4089, 2013. 23, 28, 36

[OS17] Jason M OKane and Dylan A Shell. Concise planning and filtering: hardness and algorithms. *IEEE Transactions on Automation Science and Engineering*, 14(4):1666–1681, 2017. 10

[pfl] Autonomous Profiling Floats. Available at http://navis.sea-birdscientific.com/. xvii, 130

[PK00] Gregory J Pottie and William J Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000. 10

[PPM+08] Praveen Paruchuri, Jonathan P Pearce, Janusz Marecki, Milind Tambe, Fernando Ordóñez, and Sarit Kraus. Playing games for security: An efficient exact algorithm for solving Bayesian Stackelberg games. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 895–902, 2008. 87, 88, 103

[PPP+07] Clement Petres, Yan Pailhas, Pedro Patron, Yvan Petillot, Jonathan Evans, and David Lane. Path planning for autonomous underwater vehicles. *IEEE Transactions on Robotics*, 23(2):331–341, 2007. 134

[pr2] Willow Garage's PR2 robot. http://www.willowgarage.com/pages/pr2/overview. xiii, 3, 4

[PR09] Samuel Prentice and Nicholas Roy. The belief roadmap: Efficient planning in belief space by factoring the covariance. *International Journal of Robotics Research*, 28(11-12):1448–1465, 2009. 50

[PR10] David Portugal and Rui Rocha. MSP algorithm: multi-robot patrolling based on territory allocation using balanced graph partitioning. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 1271–1276, 2010. 89

[PR11] David Portugal and Rui Rocha. A survey on multi-robot patrolling algorithms. In *Proceedings of Doctoral Conference on Computing, Electrical and Industrial Systems*, pages 139–146, 2011. 6, 89

[PZL08] Derek A Paley, Fumin Zhang, and Naomi Ehrich Leonard. Cooperative control for ocean sampling: The glider coordinated control system. *IEEE Transactions on Control Systems Technology*, 16(4):735–744, 2008. 129

[RBFT99] Nicholas Roy, Wolfram Burgard, Dieter Fox, and Sebastian Thrun. Coastal navigation-mobile robot navigation with uncertainty in dynamic environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 35–40, 1999. 50

[RF15] Susan Rodger and T Finley. JFLAP, 2015. Available at http://www.jflap.org/. 43

[RJR+09] Dean Roemmich, Gregory C Johnson, Stephen Riser, Russ Davis, John Gilson, W Brechner Owens, Silvia L Garzoli, Claudia Schmid, and Mark Ignaszewski. The Argo program: Observing the global ocean with profiling floats. *Oceanography*, 22(2):34–43, 2009. 130, 134, 171

[rmo] RoboMop. http://robomop.com/. xiii, 8

[rova] NASA's Mars exploration rover mission. https://mars.nasa.gov/mer/home/. xiii, 3

[rovb] NASA's next Mars rover: A life-hunting curiosity 2.0. https://www.space.com/38955-nasa-2020-mars-rover-curiosity.html. 4

[SCL+10] Ryan N Smith, Yi Chao, Peggy P Li, David A Caron, Burton H Jones, and Gaurav S Sukhatme. Planning and implementing trajectories for

autonomous underwater vehicles to track evolving ocean processes based on predictions from a regional ocean model. *International Journal of Robotics Research*, 29(12):1475–1497, 2010. 132

[sel] Google's Waymo self-driving car. https://waymo.com/. xiii, 1, 3

[SG01] Gyung Tak Sung and Inderbir S Gill. Robotic laparoscopic surgery: a comparison of the da Vinci and Zeus systems. *Urology*, 58(6):893–898, 2001. 1

[SH14a] Ryan N Smith and Van T Huynh. Controlling buoyancy-driven profiling floats for applications in ocean observation. *IEEE Journal of Oceanic Engineering*, 39(3):571–586, 2014. 129, 130

[SH14b] Ryan N Smith and Van T Huynh. Controlling buoyancy-driven profiling floats for applications in ocean observation. *IEEE Journal of Oceanic Engineering*, 39(3):571–586, 2014. 130, 139

[SKC+10] Ryan N Smith, Jonathan Kelly, Yi Chao, Burton H Jones, and Gaurav S Sukhatme. Towards improvement of autonomous glider navigation accuracy through the use of regional ocean models. In *Proceedings of the International Conference on Offshore Mechanics and Arctic Engineering (OMAE)*, pages 597–606, 2010. 132

[SLM05] Shawn C Shadden, Francois Lekien, and Jerrold E Marsden. Definition and properties of Lagrangian coherent structures from finite-time lyapunov exponents in two-dimensional aperiodic flows. *Physica D: Nonlinear Phenomena*, 212(3):271–304, 2005. 171

[SM05] Alexander F Shchepetkin and James C McWilliams. The Regional Ocean Modeling System (ROMS): a split-explicit, free-surface, topography-following-coordinate oceanic model. *Ocean Modelling*, 9(4):347–404, 2005. xvii, 131, 141, 157

[SNS04] Roland Siegwart, Illah R Nourbakhsh, and Davide Scaramuzza. *Autonomous Mobile Robots*. MIT Press, 2004. 6

[SO12] Nicholas M Stiffler and Jason M O'Kane. Shortest paths for visibility-based pursuit-evasion. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3997–4002, 2012. 90

[SO14] Nicholas M Stiffler and Jason M O'Kane. A complete algorithm for visibility-based pursuit-evasion with multiple pursuers. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1660–1667, 2014. 90

[SP12] Demetris Stavrou and Christos Panayiotou. Localization of a simple robot with low computational-power using a single short range sensor. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 729–734, 2012. 49

[SPC+10] Ryan N Smith, Arvind Pereira, Yi Chao, Peggy P Li, David A Caron, Burton H Jones, and Gaurav S Sukhatme. Autonomous underwater vehicle trajectory design coupled with predictive ocean models: A case study. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4770–4777, 2010. 129

[sro] Knightscope security robot. https://www.knightscope.com/. xiii, 2, 3

[sro14] Rise of the robot security guards, 2014. https://www.technologyreview.com/s/532431/rise-of-the-robot-security-guards/. 2

[SSS+11] Ryan N Smith, Mac Schwager, Stephen L Smith, Burton H Jones, Daniela Rus, and Gaurav S Sukhatme. Persistent ocean monitoring with underwater gliders: Adapting sampling resolution. *Journal of Field Robotics*, 28(5):714–741, 2011. 130, 133

[Ste94] Williams J Stewart. *Introduction to the numerical solutions of Markov chains*. Princeton University Press, 1994. 132

[Su07] Steve Su. Numerical maximum log-likelihood estimation for generalized lambda distributions. *Computational Statistics & Data Analysis*, 51(8):3983–3998, 2007. 110

[Tar72] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. 66, 148

[TBF05] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT press, 2005. 5, 47, 155

[TCA+09] David R Thompson, Steve Chien, Matthew Arrott, Arjuna Balasuriya, Yi Chao, Peggy Li, Michael Meisinger, Stephanie Petillo, and Oscar

Schofield. Mission planning in a dynamic ocean sensorweb. In *Proceedings of the International Conference on Planning and Scheduling (ICAPS) SPARK Applications Workshop*, 2009. 133

[TCB+14] Benjamín Tovar, Fred Cohen, Leonardo Bobadilla, Justin Czarnowski, and Steven M Lavalle. Combinatorial filters: Sensor beams, obstacles, and possible paths. *ACM Transactions on Sensor Networks*, 10(3):47, 2014. 10

[Ted09] Russ Tedrake. LQR-Trees: Feedback motion planning on sparse randomized trees. 2009. 167

[TFBD01] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141, 2001. 12, 20

[TGL04] Benjamín Tovar, Luis Guilamo, and Steven M LaValle. Gap navigation trees: Minimal representation for visibility-based tasks. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics (WAFR-VI)*, pages 425–440. 2004. 49

[TMCL07] Benjamín Tovar, Rafael Murrieta-Cid, and Steven M LaValle. Distance-optimal navigation in an unknown environment without sensing distances. *IEEE Transactions on Robotics*, 23(3):506–518, 2007. 23

[TP03] Paulo Tabuada and George J Pappas. Finite bisimulations of controllable linear systems. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pages 634–639, 2003. 169

[tur] TurtleBot. http://www.turtlebot.com/. xiii, 3, 4

[TYOL05] Benjamín Tovar, Anna Yershova, Jason M O'Kane, and Steven M LaValle. Information spaces for mobile robots. In *Proceedings of the Fifth International Workshop on Robot Motion and Control (RoMoCo)*, pages 11–20, 2005. 23

[VATS14] Yevgeniy Vorobeychik, Bo An, Milind Tambe, and Satinder P Singh. Computing solutions in infinite-horizon discounted adversarial patrolling games. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 314–322, 2014. 90

[vcS94] Josephus Antonius Wilhelmus van cler Spek. *Cell mapping methods: modifications and extensions*. PhD thesis, Eindhoven University of Technology, Netherlands, 1994. 25, 28, 56

[VdS04] AJ Van der Schaft. Equivalence of dynamical systems by bisimulation. *IEEE Transactions on Automatic Control*, 49(12):2160–2172, 2004. 169

[VKS13] Ch K Volos, Ioannis M Kyprianidis, and Ioannis N Stouboulos. Experimental investigation on coverage performance of a chaotic autonomous mobile robot. *Robotics and Autonomous Systems*, 61(12):1314–1322, 2013. 51

[way18] Waymo maintains lead in the self-driving car race, 2018. https://www.theatlantic.com/technology/archive/2018/01/waymo-maintains-huge-lead-in-self-driving-car-race/552059/. 2

[WBL+99] Charles R Weisbin, John Blitch, D Lavery, Eric Krotkov, Chuck Shoemaker, L Matthies, and Guillermo Rodriguez. Miniature robots for space and military missions. *IEEE Robotics & Automation Magazine*, 6(3):9–18, 1999. 1

[WD08] Jonas Witt and Matthew Dunbabin. Go with the flow: Optimal AUV path planning in coastal environments. In *Proceedings of the Australasian Conference on Robotics and Automation (ACRA)*, 2008. 134

[WL94] Fei-Yue Wang and Paul JA Lever. A cell mapping method for general optimum trajectory planning of multiple robotic arms. *Robotics and Autonomous Systems*, 12(1-2):15–27, 1994. 11

[WM03] Sylvia C Wong and Bruce A MacDonald. A topological coverage algorithm for mobile robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1685–1690, 2003. 51

[WYSH00] Louis Whitcomb, Dana R Yoerger, Hanumant Singh, and Jonathan Howland. Advances in underwater robot vehicles for deep ocean exploration: Navigation, control, and survey operations. In *Proceedings of the International Symposium on Robotics Research (ISRR)*, pages 439–448. 2000. 129

[XLR16] Marios Xanthidis, Alberto Quattrini Li, and Ioannis Rekleitis. Shallow coral reef surveying by inexpensive drifters. In *Proceedings of the*

MTS/IEEE OCEANS-Shanghai, pages 1–9, 2016. xvii, 129, 130, 133, 137

[Yen71] Jin Y Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971. 100

[YL12] Jingjin Yu and Steven M LaValle. Shadow information spaces: Combinatorial filters for tracking targets. *IEEE Transactions on Robotics*, 28(2):440–456, 2012. 23

[YTGL05] Anna Yershova, Benjamín Tovar, Robert Ghrist, and Steven M LaValle. Bitbots: Simple robots solving complex tasks. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, 2005. 10

[ZJKK05] Xiaoming Zheng, Sonal Jain, Sven Koenig, and David Kempe. Multi-robot forest coverage. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3852–3857, 2005. 51

[ZL90] Wen H Zhu and Ming-Chuan Leu. Planning optimal robot trajectories by cell mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1730–1735, 1990. 11

VITA

TAUHIDUL ALAM

	Born, Chittagong, Bangladesh
2013 – 2018	Ph.D., Computer Science Florida International University Miami, Florida, U.S.A.
2004 – 2008	B.Sc., Computer Science and Engineering Chittagong University of Engineering and Technology Chittagong, Bangladesh
08/2013 – 08/2018	Graduate Assistant School of Computing and Information Sciences Florida International University Miami, Florida, U.S.A.

PUBLICATIONS

[1] Tauhidul Alam, Md. Mahbubur Rahman, Pedro Carrillo, Leonardo Bobadilla, and Brian Rapp. Stochastic multi-robot patrolling with limited visibility. Accepted in *Journal of Intelligent and Robotic Systems*, 2018.

[2] Tauhidul Alam, Gregory Murad Reis, Leonardo Bobadilla, and Ryan N. Smith. An underactuated vehicle localization method in marine environments. Accepted in the *MTS/IEEE OCEANS Conference*, Charleston, SC, USA, October 2018.

[3] Gregory Murad Reis, Hector Leon, Tauhidul Alam, Jacob Anderson, Leonardo Bobadilla, and Ryan N. Smith. A whitening-based tracking algorithm for autonomous underwater vehicles. In *Proceedings of the MTS/IEEE OCEANS Conference*, Kobe, Japan, May 2018.

[4] Tauhidul Alam, Gregory Murad Reis, Leonardo Bobadilla, and Ryan N. Smith. A data-driven deployment approach for persistent monitoring in aquatic environments. In *Proceedings of the IEEE International Conference on Robotic Computing (IRC)*, Laguna Hills, CA, USA, pp. 147 – 154, February 2018.

[5] Tauhidul Alam, Leonardo Bobadilla, and Dylan A. Shell. Space-efficient filters for mobile robot localization from discrete limit cycles. *IEEE Robotics and Automation Letters*, 3(1): 257 – 264, January 2018. Also presented at the *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* in Vancouver, Canada, September 2017.

[6] Tauhidul Alam, Md. Mahbubur Rahman, Leonardo Bobadilla, and Brian Rapp. Multi-vehicle patrolling with limited visibility and communication constraints. In *Proceedings of the IEEE Conference on Military Communications (MILCOM)*, pp. 465 − 470, Baltimore, MD, USA, October 2017.

[7] Tauhidul Alam, Leonardo Bobadilla, and Dylan A. Shell. Minimalist robot navigation and coverage using a dynamical system approach. In *Proceedings of the IEEE International Conference on Robotic Computing (IRC)*, Taichung, Taiwan, pp. 249 − 256, April 2017.

[8] Tauhidul Alam. Decentralized and nondeterministic multi-robot area patrolling in adversarial environments. *International Journal of Computer Applications*, 156(2): 1 − 8, December 2016.

[9] Tauhidul Alam, Matthew Edwards, Leonardo Bobadilla, and Dylan Shell. Distributed multi-robot area patrolling in adversarial environments. In the *International Workshop on Robotic Sensor Networks (RSN)*, Seattle, WA, USA, April 2015.