Generating Informative Paths for Persistent Sensing in Unknown Environments

by

Daniel Eduardo Soltero

B.S., University of Puerto Rico, Mayaguez (2010)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of Master of Science
at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2012

© Daniel Eduardo Soltero, MMXII. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

Author .................................................................
Department of Electrical Engineering and Computer Science

Certified by ..............................................................
Daniela Rus
Professor of Electrical Engineering and Computer Science

Accepted by .............................................................
Leslie A. Kolodziejski
Chair, Department Committee on Graduate Students

This work is sponsored in part by ONR-MURI Award N00014-09-1-1051, the NSF Graduate Research Fellowship Award 0645960 & The Boeing Company.
Generating Informative Paths for Persistent Sensing in Unknown Environments

by

Daniel Eduardo Soltero

Submitted to the Department of Electrical Engineering and Computer Science on May 22, 2012, in partial fulfillment of the requirements for the degree of Master of Science

Abstract

In this thesis, we present an adaptive control law for a team of robots to shape their paths to locally optimal configurations for sensing an unknown dynamical environment. As the robots travel through their paths, they identify the areas where the environment is dynamic and shape their paths to sense these areas. A Lyapunov-like stability proof is used to show that, under the proposed adaptive control law, the paths converge to locally optimal configurations according to a Voronoi-based coverage task, i.e. informative paths. The problem is first treated for a single robot and then extended to multiple robots.

Additionally, the controllers for both the single-robot and the multi-robot case are extended to treat the problem of generating informative paths for persistent sensing tasks. Persistent sensing tasks are concerned with controlling the trajectories of mobile robots to act in a growing field in the environment in a way that guarantees that the field remains bounded for all time. The extended informative path controllers are proven to shape the paths into informative paths that are useful for performing persistent sensing tasks.

Lastly, prior work in persistent sensing tasks only considered robotic systems with collision-free paths. In this thesis we also describe a solution to multi-robot persistent sensing, where robots have intersecting trajectories. We develop collision and deadlock avoidance algorithms and quantify the impact of avoiding collision on the overall stability of the persistent sensing task. Simulated and experimental results support the proposed approach.

Thesis Supervisor: Daniela Rus
Title: Professor of Electrical Engineering and Computer Science
Acknowledgments

I would like to thank professor Daniela Rus for giving me great support, energy and enthusiasm throughout this process. Her ideas and vision guided me and positively affected the outcome of this thesis. I would also like to thank professors Mac Schwager and Stephen Smith for their excellent guidance and help. Mac Schwager was a key ingredient in this thesis in the topic of informative paths, and Stephen Smith was a key ingredient for the topic of persistent sensing.

I would also like to thank all of the Distributed Robotics Laboratory in CSAIL, MIT for their unconditional support and help in any topic. I especially thank Brian Julian for his guidance, ideas and mentorship. His insight and experience are invaluable.

I would also like to thank my father, mother, brother, sisters, grandparents, and the rest of my family for their support and understanding. My parents and grandparents have been a great source of support and inspiration and they have pushed me into becoming who I am today.
## Contents

1 Introduction 19
   1.1 Motivation and Goals .............................. 19
   1.2 Contribution to Robotics ........................... 23
   1.3 Relation to Previous Work ......................... 25
   1.4 Thesis Organization ................................ 28

2 Informative Path Controller for a Single Robot 29
   2.1 Single Robot System - Problem Setup ............... 30
      2.1.1 Single Robot System Description ............... 30
      2.1.2 Environment Structure .......................... 30
   2.2 Locational Optimization ............................ 31
   2.3 Sensory Function Approximation .................... 33
   2.4 Informative Path Controller ....................... 34
   2.5 Simulation and Results ............................. 38
      2.5.1 System Architecture .......................... 38
      2.5.2 Numerical Simulation .......................... 40
   2.6 Discussion ........................................ 45

3 Persistent Informative Controller for a Single Robot 47
   3.1 Single Robot System Description for Persistent Sensing ........... 47
   3.2 Relation to Persistent Sensing Tasks ................ 48
   3.3 Persistent Informative Controller ................... 49
   3.4 Simulation, Results and Implementation ............. 52
6.4.2 All Time Policy

6.4.3 Time Window Policy

6.5 Performance Bound

6.6 Simulation, Results and Implementation

6.6.1 Simulation Results

6.6.2 Distributed Implementation

6.7 Discussion

7 Hardware Implementation of Persistent Informative Controller for
Multiple Robots with Collision Avoidance

7.1 System Architecture

7.2 Results

8 Conclusion and Lessons Learned

A Tables of Important Symbols
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Example of path reshaping process for two robots.</td>
<td>20</td>
</tr>
<tr>
<td>1-2</td>
<td>Example of persistent sensing by two robots.</td>
<td>22</td>
</tr>
<tr>
<td>2-1</td>
<td>Architecture for single-robot system</td>
<td>38</td>
</tr>
<tr>
<td>2-2</td>
<td>Integral parameter error under the informative path controller for a single robot</td>
<td>41</td>
</tr>
<tr>
<td>2-3</td>
<td>Lyapunov-like function in learning phase under the informative path controller for a single robot</td>
<td>41</td>
</tr>
<tr>
<td>2-4</td>
<td>Simulated single robot system with informative path controller during the learning phase</td>
<td>42</td>
</tr>
<tr>
<td>2-5</td>
<td>Mean waypoint position error under the informative path controller for a single robot</td>
<td>43</td>
</tr>
<tr>
<td>2-6</td>
<td>Lyapunov-like function in path shaping phase under the informative path controller for a single robot</td>
<td>43</td>
</tr>
<tr>
<td>2-7</td>
<td>Simulated single robot system with informative path controller during the path shaping phase</td>
<td>44</td>
</tr>
<tr>
<td>3-1</td>
<td>Geometric representation for Lyapunov-like proof</td>
<td>51</td>
</tr>
<tr>
<td>3-2</td>
<td>Integral parameter error under the persistent informative controller for a single robot</td>
<td>53</td>
</tr>
<tr>
<td>3-3</td>
<td>Lyapunov-like function in learning phase under the persistent informative controller for a single robot</td>
<td>53</td>
</tr>
<tr>
<td>3-4</td>
<td>Simulated single robot system with persistent informative controller during the learning phase</td>
<td>54</td>
</tr>
</tbody>
</table>
3-5 Mean waypoint position error under the persistent informative controller for a single robot ........................................ 55
3-6 Lyapunov-like function in path shaping phase under the persistent informative controller for a single robot ..................... 55
3-7 Simulated single robot system with persistent informative controller during the path shaping phase ............................. 56
3-8 Persistent sensing task’s stability margin under the persistent informative controller for a single robot ............................. 57
3-9 Hardware implementation of the single-robot system ......................... 58
4-1 Architecture for the multi-robot system ........................................ 68
4-2 Integral parameter error under the informative path controller for multiple robots ....................................................... 71
4-3 Lyapunov-like function in learning phase under the informative path controller for multiple robots ............................... 71
4-4 Consensus error under the informative path controller for multiple robots ................................................................. 71
4-5 Simulated multi-robot system with informative path controller during the learning phase ............................................. 72
4-6 Mean waypoint position error under the informative path controller for multiple robots .................................................. 73
4-7 Lyapunov-like function in path shaping phase under the informative path controller for multiple robots ......................... 73
4-8 Simulated multi-robot system with informative path controller during the path shaping phase ......................................... 74
4-9 Initial paths for comparing neighbor distance weight ....................... 75
4-10 Final paths for comparing neighbor distance weight ....................... 76
4-11 Environment for comparing neighbor distance weight ....................... 76
4-12 Consensus error for comparing neighbor distance weight ....................... 76
4-13 Lyapunov-like in learning phase for comparing neighbor distance weight 76
4-14 Lyapunov-like function in path shaping phase for comparing neighbor distance weight .................................................. 76
4-15 Mean waypoint position error for comparing neighbor distance weight 76
4-16 Mean integral parameter error for comparing neighbor distance weight 76
4-17 Initial paths for comparing initial paths ................................. 78
4-18 Final paths for comparing initial paths ................................. 79
4-19 Environment for comparing initial paths ............................... 79
4-20 Consensus error for comparing initial paths ............................ 79
4-21 Lyapunov-like function in learning phase for comparing initial paths 79
4-22 Lyapunov-like function in path shaping phase for comparing initial paths 79
4-23 Mean waypoint position error for comparing initial paths ......... 79
4-24 Mean integral parameter error for comparing initial paths ........ 79

5-1 Integral parameter error under the persistent informative controller for multiple robots ...................................................... 85
5-2 Lyapunov-like function in learning phase under the persistent informative controller for multiple robots .......................... 85
5-3 Consensus error under the persistent informative controller for multiple robots ................................................................. 85
5-4 Simulated multi-robot system with persistent informative controller during the learning phase ................................................. 86
5-5 Mean waypoint position error under the persistent informative controller for multiple robots ................................................. 87
5-6 Lyapunov-like function in path shaping phase under the persistent informative controller for multiple robots ...................... 87
5-7 Persistent sensing task’s stability margin under the persistent informative controller for multiple robots .............................. 87
5-8 Simulated multi-robot system with persistent informative controller during the path shaping phase ............................... 88
6-1  Example of four robots performing a persistent sensing task with intersecting paths ........................................ 92
6-2  Phase portrait of two robots performing a persistent sensing task ......................................................... 94
6-3  Paths for three robots used in Example 1 ......................................................................................... 96
6-4  Collision graph for Example 1 .................................................................................................. 97
6-5  Trajectory sets for collision avoidance simulations .............................................................................. 106
6-6  Stopping policy rankings ................................................................................................................ 107
6-7  Head-to-head comparison of All-Time versus Min-Time policies .................................................... 108
6-8  Snapshots of distributed implementation of the collision avoidance algorithm .................................. 109

7-1  Architecture for the multi-robot hardware implementation ................................................................. 112
7-2  Environment for the multi-robot hardware implementation ................................................................ 113
7-3  Initial paths for the multi-robot hardware implementation ................................................................... 113
7-4  Basis coefficient error for the multi-robot hardware implementation .................................................. 114
7-5  Consensus error for the multi-robot hardware implementation ............................................................. 114
7-6  Integral parameter error for the multi-robot hardware implementation ................................................ 114
7-7  Lyapunov-like function in learning phase for the multi-robot hardware implementation .................. 114
7-8  Mean waypoint position error for the multi-robot hardware implementation ....................................... 115
7-9  Lyapunov-like function in path shaping phase for the multi-robot hardware implementation ................ 115
7-10 Persistent sensing task’s stability margin for the multi-robot hardware implementation ..................... 115
7-11 Mean accumulation function value for the multi-robot hardware implementation ............................ 115
7-12 Snapshots of the hardware implementation of the multi-robot system ............................................. 116
7-13 Path shaping phase of the multi-robot system hardware implementation ........................................... 117
**List of Algorithms**

<table>
<thead>
<tr>
<th></th>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Informative Path Controller for a Single Robot: Layer 1</td>
<td>39</td>
</tr>
<tr>
<td>2</td>
<td>Informative Path Controller for a Single Robot: Layer 2</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>Persistent Informative Controller for a Single Robot: Layer 2</td>
<td>52</td>
</tr>
<tr>
<td>4</td>
<td>Informative Path Controller for Multiple Robots: Layer 1</td>
<td>69</td>
</tr>
<tr>
<td>5</td>
<td>Informative Path Controller for Multiple Robots: Layer 2</td>
<td>70</td>
</tr>
<tr>
<td>6</td>
<td>Persistent Informative Controller for Multiple Robots: Layer 2</td>
<td>84</td>
</tr>
<tr>
<td>7</td>
<td>Collision Avoidance Framework</td>
<td>97</td>
</tr>
<tr>
<td>8</td>
<td>Collision and Deadlock Avoidance Algorithm</td>
<td>99</td>
</tr>
<tr>
<td>9</td>
<td>Min-Time Policy</td>
<td>102</td>
</tr>
<tr>
<td>10</td>
<td>All-Time Policy</td>
<td>103</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>Common symbols to all controllers</td>
<td>123</td>
</tr>
<tr>
<td>A.2</td>
<td>Symbols for single-robot controllers</td>
<td>124</td>
</tr>
<tr>
<td>A.3</td>
<td>Symbols for multi-robot controllers</td>
<td>125</td>
</tr>
<tr>
<td>A.4</td>
<td>Symbols for collision avoidance</td>
<td>126</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation and Goals

Robots operating in dynamic and unknown environments are often faced with the problem of deciding where to go to get the most relevant information for their current task. Informative path planning addresses this problem. Given a dynamic environment that is unknown, and a group of robots, each with a sensor to measure the environment, we want to find a set of paths for the robots that will maximize information gathering. To achieve this, the robots need to do two things: 1) learn the structure of the environment by identifying the areas within the environment that are dynamic and the rate of change for these areas; and 2) compute paths which allow them to sense the parts of the environment that have high rate of change. These paths are called informative paths since they focus on driving the robots through locations in the environment where the sensory information is important.

In this thesis we introduce the informative path planning problem and present a new control algorithm for generating closed informative paths in unknown environments. The key insight is inspired by [32–34]. The robots move along their paths, marking the areas they observe as dynamic or static. As the robots discover the static/dynamic structure of the environment, they reshape their paths to avoid visiting static areas and focus on sensing dynamic areas. An example of this reshaping process for two robots can be seen in Figure 1-1. We develop the theory for adaptive
Figure 1-1: Example of path reshaping process for two robots. The paths, shown as red and blue lines, connect the waypoints, shown as black circles. The blue path corresponds to robot 1, and the red path corresponds to robot 2. The robots' positions are represented by the black arrow heads. The dynamic regions of the environment, which the robots are tasked to cover, are represented by the grid of green dots. As time passes by, the robots shape their paths so they can cover the dynamic regions of the environment.
controllers that generate informative paths for a single robot and multiple robots.

The adaptive controllers have two key features. The first is an adaptation law that the robots use to learn how sensory information is distributed in the environment. This learning is done through parameter estimation. The second feature is a Voronoi-based coverage controller that performs the reshaping of the paths based on the robots’ estimates of the space. The Voronoi-based controller is an extension to previous work [34] and it consists of placing waypoints of a path in locally optimal positions that achieve an equilibrium between providing good locations for the robot to sense the environment and generating short paths. The benefit of short paths is the reduction of travel time through the static regions of the environment. The paths will drive the robots to spend more time in locations of the environment that are very dynamic, and less time in the locations that are less dynamic.

The informative path problem has many applications. For example, it can be used by a team of robots in an oil spill over a large environment. The robots would be able to learn the regions where there is oil, and generate paths such that they can better sense these oil regions while avoiding the clean regions. Another example is in a surveillance task of a city where the robots could learn the regions where crime is frequently committed and generate paths so that they can sense these crime regions more frequently and sense the safer regions less frequently. In this thesis we are interested in using it to achieve and improve persistent sensing tasks [35, 36] in unknown environments, where the robots are assumed to have sensors with finite footprints.

More specifically, in a persistent sensing task we wish for the robots with finite sensor footprints to conduct their information gathering while guaranteeing a bound on the difference between the robots’ current model of the environment and the actual state of the environment for all time and all locations. Since their sensors have finite footprints, the robots cannot collect the data about all of the environment at once. As data about a dynamic region becomes outdated, the robots must return to that region repeatedly to collect new data. In previous work [35], a persistent sensing controller calculates the speeds of the robots at each point along given paths (referred to as the
Figure 1-2: Example of persistent sensing by two robots. Each robot (black arrowhead) travels through its path, and the objective is to keep the accumulation function (green dots) low everywhere. The robots use their sensors with finite footprint (red and blue circles around the robots’ positions) to collect data at the dynamic regions and shrink the accumulation function. The size of the green dot is proportional to the value of the accumulation function at that point. On the left, a stabilizing speed profile is used to maintain the accumulation function bounded everywhere. On the right, the speed profile is not a stabilizing one and the accumulation function grows beyond control in some of the points.

The persistent sensing problem is defined in [35] as an optimization problem whose goal is to keep a time changing field as low as possible everywhere. We refer to this field as the accumulation function. The accumulation function grows where it is not covered by the robots’ sensors, indicating a growing need to collect data at that location, and shrinks where it is covered by the robots’ sensors, indicating a decreasing need for data collection. A stabilizing speed profile is one which maintains the height of the accumulation function bounded. Figure 1-2 shows an example of two robots performing a persistent sensing task where the dynamic regions of the environment are represented by the green dots. The size of a dot represents the value of the accumulation function at that location. In Figure 1-2a, a stabilizing
speed profile maintains the accumulation function bounded for all time. However, in Figure 1-2b, no stabilizing speed profile is used and the accumulation function grows unbounded for some locations. The persistent sensing problem can be applicable to a wide range of situations by simply redefining the meaning of the accumulation function. For example, the accumulation function can represent the accumulation of dust in a household, the uncertainty about measurements of an environment, or the chance of crime in a region of a city. In the case of dust, the robots’ sensors could be vacuum cleaners; and in the case of chance of crime, the robots could be UAV’s (unmanned aerial vehicles) patrolling the city and using cameras to sense it. In this thesis we show that our computed informative paths can be used in conjunction with the stabilizing speed profiles from the persistent sensing controller to locally optimize a persistent sensing task, i.e. we apply the informative path adaptive controllers to find paths in unknown environments that keep the accumulation function low everywhere.

In [35], the closed paths for robots performing persistent sensing tasks are assumed to be non-intersecting. However, most efficient monitoring paths may intersect, and when considering multiple robots traveling through intersecting paths, collisions between the robots are possible. Thus, a collision avoidance procedure is needed. In this thesis we consider this problem of multiple robots persistently sensing the environment while avoiding collision, and analyze the effect of the collision avoidance procedure on the stability of the persistent sensing task. We would like for each robot to continually follow its path using its stabilizing speed profile while avoiding collisions with the other robots. The collisions should be avoided in such a way that we can still guarantee boundedness of the accumulation function.

1.2 Contribution to Robotics

This thesis makes the following contributions:

- A novel informative path controller. We enable robots to compute online paths that are information-rich, i.e. paths that visit the regions that matter
in the unknown dynamic environment. This is done through a provably stable adaptive controller that enables the robots to learn the location of dynamic events in the environment and simultaneously compute informative paths for these events. We refer to this adaptive controller as the informative path controller. We propose an informative path controller for the single-robot case and use a Lyapunov-like proof to prove stability of the system under the proposed controller. We also extend the theory to the multi-robot case and prove stability of the multi-robot system. We simulate and analyze the informative path controllers using a MATLAB environment for both the single-robot and multi-robot system.

- **A persistent informative controller.** Previous work in persistent sensing tasks [35] requires full knowledge of the environment and a pre-designed path. These two constraints can be very restricting in practice. In this thesis, we relax these two constraints and enable robots to perform persistent sensing tasks in unknown environments when the robots’ paths are unknown ahead of time. Combining a stability metric from persistent sensing tasks with our informative path controllers for the single-robot and multi-robot cases we develop persistent informative controllers which generate informative paths for the robots while increasing the stability metric of the persistent sensing task. By doing so, we locally optimize the persistent sensing task. Lyapunov-like proofs are used to prove stability of both the single-robot and multi-robot case under this controller. We simulate and analyze the informative path controllers for persistent sensing using a MATLAB environment, as well as perform hardware implementations using quadrotor robots for both the single-robot and multi-robot system.

- **A collision avoidance strategy that is guaranteed to be deadlock free.** We enable persistent sensing tasks to operate when multiple robots have intersecting trajectories. Here, we mean intersection in the sense that the robot bodies could collide. We develop a collision avoidance procedure based on stop-
ping, and quantify its effect on the stability of the persistent sensing task. The collision avoidance operates by identifying collision zones in which collisions could occur. We then avoid collisions by stopping and restarting robot movement so that at most one robot occupies a given collision zone at any moment in time. We also design a procedure to avoid deadlocks; a situation in which a group of robots are all stopped, and are waiting for each other to move before resuming motion. We identify several different algorithms that tell us how to decide which robots stop to avoid collision while maintaining the system stable. Such algorithms are referred to as stopping policies. We perform extensive simulations and hardware implementations to test our collision and deadlock avoidance procedures and to determine the most effective stopping policy.

- **Experiments.** We evaluate the persistent informative controller with collision avoidance in simulation and hardware. We use two quadrotors as our robotic platforms, MATLAB as our main tool for calculations and ROS (robot operating system) as our messaging system for the different processes of the implementation. This hardware implementation shows the practical usage of our algorithms.

## 1.3 Relation to Previous Work

Most of the previous work in path planning/generation focuses on reaching a goal while avoiding collision with obstacles, e.g. [26], or on computing an optimal path according to some metric, e.g. [18], [28]. Other works have focused on probabilistic approaches to path planning, e.g. [16]. Prior work in adaptive path planning, e.g. [9], and [40], considers adapting the robot’s path to changes in the robot’s knowledge of the environment. In this thesis, the robots do not have to reach a final goal; our paths are closed, so the robots perpetually travel through their paths. We focus on generating paths that allow the robots to travel through regions of interest in an unknown environment. We use adaptive control tools to create a novel algorithm for computing informative paths.
In [13], the objective was to design a sampling trajectory that minimized the uncertainty of the estimated random field at the end of the time frame. A form of generalized Voronoi partitions was used to solve this optimization problem. In this thesis, we also use Voronoi partitions as the basis of our algorithms, but the field, although unknown, is not random. Also, we are not concerned with optimizing the trajectory of the agents to minimize the predictive variance in a time frame, but rather optimizing the location of agents for a coverage task in an unknown environment.

The adaptive coverage controller relevant to this thesis was first introduced in [32], where a group of agents were coordinated to place themselves in locally optimal locations to sense an unknown environment. Voronoi partitions were used to develop a control law and an adaptation law which allowed the agents to learn the environment and achieve the task. This thesis builds upon this previous work, but with some significant changes and additions. The Voronoi partitions are used by the robots to execute adaptation laws and change the location of the agents, which are now waypoints defining the robots’ paths. The paths must be closed paths and must provide good sensing locations for the robots. The robots should also be able to achieve persistent sensing along these paths.

The persistent sensing concept motivating this thesis was introduced in [35], where a linear program was designed to calculate the robots’ speeds at each point along given paths, i.e. the speed profiles, in order for them to perform a persistent sensing task, that is, maintain the height of the growing accumulation function bounded. The robots were assumed to have full knowledge of the environment and were given pre-designed paths. In this thesis we alleviate these two assumptions by having the robots learn the environment through parameter estimation, and use this information to shape their paths into useful paths. These two alleviations provide a significant step towards persistent sensing in dynamic environments.

When considering multiple robots in an environment, collision avoidance is crucial. There is a wealth of literature on collision avoidance. A common method is to use artificial potential fields [17], which repel robots from each other and from obstacles. A recently proposed method relies on velocity obstacles [37]. Such methods result
in the robots deviating from their prescribed paths and from their desired speed
profiles. For the persistent sensing application, the effect of such deviations on the
accumulation function is difficult to characterize. A more tractable approach is to
constrain each robot to remain on its path. In this context the most closely related
work is on path traversal problems where each robot is given a path. The goal is
to minimize the time until the robots reach their destination points on their paths,
while avoiding collisions [19]. The authors of [25] consider a variation on this problem
in which there are additional dynamic constraints on the robot. In [1], the authors
consider a variation in which the full trajectory (path and speed along the path) is
specified for each robot. Collisions can be avoided only by changing the start times
of each robot along its path. Our approach to collision avoidance is closely related
to [1]. However, since our paths are closed, and are repeatedly traversed by the robots,
we avoid collisions by repeatedly stopping robots. A variation of path constrained
collision avoidance is studied in [11] in the context of vehicle roundabouts, where
the speed of the robots along the paths was altered in order to avoid collisions. This
approach cannot be used in our applications because we need to use the speed profiles
from [35] to stabilize the environment.

Persistent monitoring is related to sweep coverage [6], and patrolling problems [12,
22] where robots with finite sensor footprints must sweep their sensor over every point
in the environment. The problem is also related to environmental monitoring research
such as [4, 7, 20, 21, 41]. In this prior work, the authors often use a probabilistic
model of the environment, and estimate the state of that model using a Kalman-
like filter. Then robots are controlled so as to maximize a metric on the quality
of the state estimate. The collision avoidance problem is not addressed in these
works. In addition, due to the complexity of the models used, performance guarantees
are difficult to obtain. This makes it difficult to characterize the effect of collision
avoidance on the system performance. In this thesis, we use a more tractable model,
and in doing so we can provide guarantees on the boundedness of the accumulation
function.
1.4 Thesis Organization

This thesis is divided into eight chapters. Chapter 2 presents the informative path controller for a single robot. Chapter 3 extends the informative path controller to persistent sensing. A hardware implementation is presented for this controller. Chapter 4 introduces the informative path controller for multiple robots. Chapter 5 extends the informative path controller for multiple robots to persistent sensing. Chapter 6 presents our collision and deadlock avoidance algorithm, and a distributed implementation of it. Chapter 7 shows our implementation of the multi-robot informative path controller for persistent sensing with collision avoidance. We conclude in Chapter 8 with final thoughts and lessons learned.
Chapter 2

Informative Path Controller for a Single Robot

In this chapter\(^1\) we focus on developing a controller that allows a single robot to sense an unknown environment, create an estimate of the environment based on these measurements, and reshape its path in order to cover dynamic regions in the environment. We assume we are given a robot whose task is to sense an unknown dynamic environment while traveling along a closed path consisting of a finite number of waypoints. The goal is for the robot to identify the areas within the environment that are dynamic and compute a path which allows it to sense these dynamic areas. This is done by using an adaptation law for parameter estimation, and by optimizing a coverage task based on Voronoi partitions. A path that locally optimizes this coverage task is referred to as an informative path. A formal mathematical description of the problem follows.

\(^1\)The majority of this chapter was published in [39].
2.1 Single Robot System - Problem Setup

2.1.1 Single Robot System Description

Let there be a robot, with position $p_r$, traveling along a finite number $n$ of waypoints in a convex, bounded area $Q \subset \mathbb{R}^2$. An arbitrary point in $Q$ is denoted $q$ and the position of the $i^{th}$ waypoint is denoted $p_i$. Let $\{p_1, \ldots, p_n\}$ be the configuration of the path and let $\{V_1, \ldots, V_n\}$ be the Voronoi partitions of $Q$, with waypoint positions as the generator points, defined as

$$V_i = \{q \in Q : \|q - p_i\| \leq \|q - p_j\|, \forall j \neq i\},$$

where, $\| \cdot \|$ denotes the $l^2$-norm. We assume that the robot is able to compute the Voronoi partitions based on the waypoint locations, as is common in the literature [8, 30].

Since the path is closed, each waypoint $i$ has a previous waypoint $i - 1$ and next waypoint $i + 1$ related to it, which are referred to as the neighbor waypoints of $i$. Note that $i + 1 = 1$ for $i = n$, and $i - 1 = n$ for $i = 1$. Once the robot reaches a waypoint, it continues to move to the next waypoint, in a straight line interpolation.\(^2\)

2.1.2 Environment Structure

A sensory function, defined as a map $\# : Q \mapsto \mathbb{R}_{\geq 0}$ (where $\mathbb{R}_{\geq 0}$ refers to non-negative scalars) determines the constant rate of change of the environment at point $q \in Q$. The function $\#(q)$ is not known by the robot, but the robot is equipped with a sensor to make a point measurement of $\#(p_r)$ at the robot’s position $p_r$.

Remark 1 The interpretation of the sensory function $\#(q)$ may be adjusted for a broad range of applications. It can be any kind of weighting of importance for points $q \in Q$. In this paper we treat it as a rate of change in a dynamic environment.

\(^2\)We assume that the waypoints do not outrun the robot under the action of the informative path controller, i.e. that the robot will reach a waypoint at some time and then continue moving to the next one, which it will also reach at some time, and so on.
2.2 Locational Optimization

Let $p_{i,j} = p_i - p_j$. Notice that $p_{i,j} = -p_{j,i}$. Building upon [34], we can formulate the cost incurred by the system over the region $Q$ as

$$\mathcal{H}_1 = \sum_{i=1}^{n} \int_{V_i} \frac{W_s}{2} \|q - p_i\|^2 \phi(q) dq + \sum_{i=1}^{n} \frac{W_n}{2} \|p_{i,i+1}\|^2, \quad (2.1)$$

where $\|q - p_i\|$ can be interpreted as the unreliability of the sensory function value $\phi(q)$ when the robot is at $p_i$, and $\|p_{i,i+1}\|$ can be interpreted as the cost of a waypoint being too far away from a neighboring waypoint. $W_s$ and $W_n$ are constant positive scalar weights assigned to the sensing task and neighbor distance, respectively. Note that unreliable sensing and distance between neighboring waypoints are expensive. A formal definition of an informative path for a single robot follows.

**Definition 1 (Informative Path for a Single Robot)** An informative path for a single-robot system corresponds to a set of waypoint locations that locally minimize (2.1).

Next we define three properties analogous to mass-moments of rigid bodies. The mass, first mass-moment, and centroid of $V_i$ are defined as

$$M_i = \int_{V_i} W_s \phi(q) dq \quad (2.2)$$

$$L_i = \int_{V_i} W_s q \phi(q) dq \quad (2.3)$$

$$C_i = \frac{L_i}{M_i} \quad (2.4)$$

respectively. Also, let $e_i = C_i - p_i$. 

31
From locational optimization [10] and from differentiation under the integral sign for Voronoi partitions [27] we have

$$\frac{\partial H_1}{\partial p_i} = -\int_{V_i} W_s(q - p_i) \phi(q) d\textbf{q} - W_n p_{i+1,i} - W_n p_{i-1,i}$$

$$= -(L_i - p_i M_i) - W_n p_{i+1,i} - W_n p_{i-1,i}$$

$$= -M_i e_i - W_n p_{i+1,i} - W_n p_{i-1,i}. \quad (2.5)$$

An equilibrium is reached when $\frac{\partial H_1}{\partial p_i} = 0$. Assigning to each waypoint dynamics of the form

$$\dot{p}_i = u_i, \quad (2.6)$$

where $u_i$ is the control input, we propose the following control law for the waypoints to converge to an equilibrium configuration:

$$u_i = \frac{K_i (M_i e_i + \alpha_i)}{\beta_i}, \quad (2.7)$$

where

$$\alpha_i = W_n p_{i+1,i} + W_n p_{i-1,i},$$

$$\beta_i = M_i + 2W_n,$$

and $K_i$ is a uniformly positive definite matrix.

**Remark 2** $\beta_i > 0$ has the nice effect of normalizing the weight distribution between sensing and staying close to neighboring waypoints. Also, $K_i$ could potentially be time-varying to improve performance.
2.3 Sensory Function Approximation

We assume that the sensory function $\phi(q)$ can be parameterized as an unknown linear combination of a set of known basis functions. That is,

Assumption 1 (Matching condition) \( \exists a \in \mathbb{R}^m \) and \( K : Q \rightarrow \mathbb{R}_{\geq 0}^m \), where \( \mathbb{R}_{\geq 0} \) is a vector of non-negative entries, such that

\[
\phi(q) = K(q)^T a, \tag{2.8}
\]

where the vector of basis functions \( K(q) \) is known by the robot, but the parameter vector \( a \) is unknown. Furthermore,

\[
a(j) \geq 0, \quad \forall j = 1, \ldots, m, \tag{2.9}
\]

where \( a(j) \) denotes the \( j \)th element of \( a \).

Now, let \( \hat{a}(t) \) be robot’s approximation of the parameter vector \( a \). Then, \( \hat{\phi}(q) = K(q)^T \hat{a} \) is the robot’s approximation of \( \phi(q) \). Building on this, we define the mass moment approximations as

\[
\hat{M}_i = \int_{V_i} W_i \hat{\phi}(q) dq, \tag{2.10}
\]

\[
\hat{\dot{L}}_i = \int_{V_i} W_i q \hat{\phi}(q) dq, \tag{2.11}
\]

\[
\hat{C}_i = \frac{\hat{\dot{L}}_i}{\hat{M}_i}. \tag{2.12}
\]
Additionally, we can define $\ddot{a} = \dot{a} - a$, and the sensory function error, and mass moment errors as

\[
\tilde{\phi}(q) = \hat{\phi}(q) - \phi(q) = K(q)^T \ddot{a},
\]
\[
\tilde{M}_i = \hat{M}_i - M_i = \int_{V_i} W_s K(q)^T dq \ \ddot{a},
\]
\[
\tilde{L}_i = \hat{L}_i - L_i = \int_{V_i} W_s q K(q)^T dq \ \ddot{a},
\]
\[
\tilde{C}_i = \frac{\tilde{L}_i}{\tilde{M}_i},
\]

respectively. Note that $\tilde{C}_i \neq \hat{C}_i - C_i$. Finally, in order to compress the notation, let $K_{p_r}(t)$ and $\phi_{p_r}(t)$ be the value of the basis function vector and the value of $\phi$ at the robot’s position $p_r(t)$, respectively.

### 2.4 Informative Path Controller

We design an adaptive control law and prove that it causes the path to converge to a locally optimal configuration according to (2.1), while causing the robot’s estimate of the environment to converge to the real environment description by integrating its sensory measurements along its trajectory.

Since the robot does not know $\phi(q)$, but has an estimate $\hat{\phi}(q)$, the control law from (2.7) becomes

\[
u_i = \frac{K_i(\hat{M}_i \dot{\alpha}_i + \alpha_i)}{\hat{\beta}_i},
\]

where

\[
\alpha_i = W_n p_{i+1,i} + W_n p_{i-1,i},
\]
\[
\hat{\beta}_i = \hat{M}_i + 2W_n
\]
\[
\dot{\alpha}_i = \hat{C}_i - p_i.
\]
The parameter $\dot{a}$ is adjusted according to an adaptation law which is described next. Let

$$\lambda = \int_0^t w(\tau)K_p(\tau)\phi_p(\tau)d\tau,$$

$$\Lambda = \int_0^t w(\tau)K_p(\tau)K_p(\tau)^Td\tau,$$

where

$$w(t) = \begin{cases} 
\text{positive constant scalar,} & \text{if } t < \tau_w, \\
0, & \text{otherwise,} 
\end{cases}$$

and $\tau_w$ is some positive time at which part of the adaptation shuts down to maintain $\Lambda$ and $\lambda$ bounded. $\Lambda$ and $\lambda$ can also be calculated differentially by the robot by using $\dot{\Lambda}(t) = w(t)K_p(t)K_p(t)^T$, and $\dot{\lambda}(t) = w(t)K_p(t)\phi_p(t)$, with zero initial conditions. Let

$$\hat{a}_{\text{pre}} = b - \gamma(\Lambda \dot{a} - \lambda),$$

$$\hat{a} = \Gamma(\hat{a}_{\text{pre}} - I_{\text{proj}}\hat{a}_{\text{pre}}),$$

where $\gamma > 0$ is the adaptation gain scalar. Since $a(j) \geq 0$, $\forall j$, we enforce $\dot{a}(j) \geq 0$, $\forall j$. We do this by using a projection law [34],

$$I_{\text{proj}} = \begin{cases} 
0, & \text{if } \dot{a}(j) > 0, \\
0, & \text{if } \dot{a}(j) = 0 \text{ and } \hat{a}_{\text{pre}}(j) \geq 0, \\
1, & \text{otherwise,} 
\end{cases}$$

where $(j)$ denotes the $j^{th}$ element for a vector and the $j^{th}$ diagonal element for a
Theorem 1 (Convergence Theorem for a Single Robot) Under Assumption 1, with waypoint dynamics specified by (2.6), control law specified by (2.17) and adaptive law specified by (2.23), we have

1. \( \lim_{t \to \infty} \| \dot{M}_i(t) \dot{e}_i(t) + \alpha_i(t) \| = 0 \quad \forall i \in \{1, \ldots, n\} \)

2. \( \lim_{t \to \infty} \| \bar{\phi}_{\rho_r}(\tau) \| = 0 \quad \forall \tau \mid w(\tau) > 0 \)

Proof 1 We define a Lyapunov-like function based on the robot’s path and environment estimate, and use Barbalat’s lemma to prove asymptotic stability of the system to a locally optimal equilibrium.

Let

\[
V_1 = \mathcal{H}_1 + \frac{1}{2} \tilde{a}^T \Gamma^{-1} \tilde{a}.
\] (2.25)

Taking the time derivative of \( V_1 \), we obtain

\[
\dot{V}_1 = \sum_{i=1}^{n} \frac{\partial \mathcal{H}_1}{\partial \rho_i} T \dot{p}_i + \tilde{a}^T \Gamma^{-1} \tilde{a} = \sum_{i=1}^{n} -(M_i e_i + \alpha_i) T \dot{p}_i + \tilde{a}^T \Gamma^{-1} \tilde{a}.
\] (2.26)

From (2.14), (2.15), (2.16), it is easy to check that

\[
L_i = M_i C_i = M_i \dot{C}_i + \dot{M}_i (\dot{C}_i - \ddot{C}_i).
\]

Plugging this into (2.26),

\[
\dot{V}_1 = \sum_{i=1}^{n} -(M_i \dot{e}_i + \alpha_i) T \dot{p}_i + (\ddot{L}_i - \ddot{M}_i \dot{C}_i) T \dot{p}_i + \tilde{a}^T \Gamma^{-1} \tilde{a}.
\]

Using (2.14), we have

\[
\dot{V}_1 = \sum_{i=1}^{n} -(\ddot{M}_i \dot{e}_i + \alpha_i) T \dot{p}_i + (\ddot{L}_i - \ddot{M}_i \rho_i) T \dot{p}_i + \tilde{a}^T \Gamma^{-1} \tilde{a}.
\]
Substituting the dynamics specified by (2.6) and control law specified by (2.17), we obtain

\[
\dot{V}_1 = \sum_{i=1}^{n} -\frac{1}{\hat{\beta}_i} (\hat{M}_i \hat{e}_i + \alpha_i)^T K_i (\hat{M}_i \hat{e}_i + \alpha_i) + \sum_{i=1}^{n} (\hat{L}_i - \hat{M}_i \hat{p}_i)^T \hat{p}_i + \hat{a}^T \Gamma^{-1} \hat{a}.
\]

Using (2.14) and (2.15),

\[
\dot{V}_1 = \sum_{i=1}^{n} -\frac{1}{\hat{\beta}_i} (\hat{M}_i \hat{e}_i + \alpha_i)^T K_i (\hat{M}_i \hat{e}_i + \alpha_i) + \hat{a}^T \sum_{i=1}^{n} \int_{t} \mathcal{V}_i \bar{\mathcal{K}}(\textbf{q} - \textbf{p}_i)^T d\textbf{q} \hat{p}_i + \hat{a}^T \Gamma^{-1} \hat{a}.
\]

Plugging in the adaptation law from (2.23), (2.18) and (2.19),

\[
\dot{V}_1 = \sum_{i=1}^{n} -\frac{1}{\hat{\beta}_i} (\hat{M}_i \hat{e}_i + \alpha_i)^T K_i (\hat{M}_i \hat{e}_i + \alpha_i) - \gamma \int_{0}^{t} w(\tau)(\hat{\phi}_r(\tau))^2 d\tau - \hat{a}^T I_{\text{prof}} \hat{a} \text{pre}(2.27)
\]

Denote the three terms in (2.27) as \(\xi_1(t)\), \(\xi_2(t)\) and \(\xi_3(t)\), so that \(\dot{V}_1(t) = \dot{\xi}_1(t) + \xi_2(t) + \dot{\xi}_3(t)\). Notice that \(\xi_1(t) \leq 0\) since \(K_i\) is uniformly positive definite and \(\hat{\beta}_i > 0\), \(\xi_2(t) \leq 0\) since it is the negative integral of a squared quantity, and it was proven in [34] that \(\xi_3(t) \leq 0\). Now consider the time integral of each of these three terms, \(\int_{0}^{t} \xi_k(\tau) d\tau\), \(k = 1, 2, 3\). Since each of the terms is non-positive, \(\int_{0}^{t} \xi_k(\tau) d\tau \leq 0\), \(\forall k\), and since \(V_1 > 0\), each integral is lower bounded by \(\int_{0}^{t} \xi_k(\tau) d\tau \geq -V_{i0}\), where \(V_{i0}\) is the initial value of \(V_1\). Therefore, these integrals are lower bounded and non-increasing, and hence \(\lim_{t \to \infty} \int_{0}^{t} \xi_k(\tau) d\tau\) exists and is finite for all \(k\). Furthermore, it was shown in [34] (Lemma 1) that \(\dot{\xi}_1(t)\) and \(\dot{\xi}_2(t)\) are uniformly bounded, therefore \(\xi_1(t)\) and \(\xi_2(t)\) are uniformly continuous. Hence, by Barbalat’s Lemma [14], \(\xi_1(t) \to 0\) and \(\xi_2(t) \to 0\). This implies (i) and (ii).

Property (i) from Theorem 1 implies that the path will reach a locally optimal configuration for sensing, i.e. an informative path, where the waypoints reach a stable balance between the sensing task and being close to their neighbors. Property (ii) from Theorem 1 implies that the sensory function estimate error will converge to zero for all points on the robot’s trajectory with positive weight \(w(t)\), but not necessarily for all the environment. This means that the robot will learn the true sensory function for
the environment if its trajectory is rich enough while the weight is positive. Since it is crucial for the sensory function estimate to approach the true sensory function for all the environment, the initial waypoint locations can be designed so the robot initially travels most of the environment in dynamic unknown environments (see Figure 2-4).

2.5 Simulation and Results

2.5.1 System Architecture

The system architecture for our simulations can be seen in Figure 2-1 and is divided into two layers. The first layer (top of Figure 2-1 and colored in red) corresponds to the robot traveling through its path, and sampling and estimating the environment. A practical algorithm for the first layer is shown in Algorithm 1. The second layer (bottom of Figure 2-1 and colored in blue) corresponds to the waypoints moving and placing themselves in locally optimal locations to generate an informative path. We can think of each waypoint as an agent that has its own controller detailed in Algorithm 2. This algorithm can be implemented in a distributed way by each waypoint in the sense that the waypoints can execute this algorithm independently, only sharing information with their neighboring waypoints and enough information for all waypoints to compute their Voronoi cells.

Figure 2-1: Architecture for single-robot system
Algorithm 1 Informative Path Controller for a Single Robot: Layer 1

Require: $\phi(q)$ can be parametrized as in (2.8)
Require: $a \geq 0$
Require: Waypoints cannot outrun the robot traveling the path
Require: Robot knows location of $p_i$, $\forall i \in \{1,\ldots,n\}$

1: Initially robot is moving towards $p_i$
2: Initialize $\Lambda$ and $\lambda$ to zero
3: Initialize $\hat{a}$ element-wise to some bounded non-negative value
4: loop
5:     if robot reached $p_i$ then
6:         move towards $p_{i+1}$ in a straight line from $p_i$
7:     else
8:         move towards $p_i$ in a straight line from $p_r$
9:     end if
10:    Make measurement $\phi(p_r)$
11:    Update $\hat{a}$ according to (2.23)
12:    Update $\Lambda$ and $\lambda$ according to (2.18) and (2.19)
13: end loop

Algorithm 2 Informative Path Controller for a Single Robot: Layer 2 for the $i^{th}$ waypoint

Require: Each waypoint knows $\hat{a}$ from Algorithm 1
Require: Each waypoint can calculate its Voronoi partition
Require: Each waypoint knows its neighboring waypoints

1: loop
2:     Compute the waypoint’s Voronoi partition
3:     Compute $\hat{C}_i$ according to (2.12)
4:     Obtain neighbor waypoint locations $p_{i-1}$ and $p_{i+1}$
5:     Compute $u_i$ according to (2.17)
6:     Update $p_i$ according to (2.6)
7: end loop
2.5.2 Numerical Simulation

The informative path controller for a single robot was simulated in a MATLAB environment for many test cases. Here we present a case for \( n = 59 \) waypoints. A fixed-time step numerical solver is used to integrate the equations of motion and adaptation using a time step of 0.01 seconds. The region \( Q \) is taken to be the unit square. The sensory function \( \phi(q) \) is parametrized as a Gaussian network with 25 truncated Gaussians, i.e. \( \mathcal{K} = [\mathcal{K}(1) \cdots \mathcal{K}(25)]^T \), where

\[
G(j) = \frac{1}{\sigma \sqrt{2\pi}} \exp\left\{ -\frac{(q - \mu_j)^2}{2\sigma^2} \right\},
\]

\[
G_{\text{trunc}} = \frac{1}{\sigma \sqrt{2\pi}} \exp\left\{ -\frac{\rho_{\text{trunc}}^2}{2\sigma^2} \right\},
\]

\[
\mathcal{K}(j) = \begin{cases} G(j) - G_{\text{trunc}}, & \text{if } \|q - \mu_j\| < \rho_{\text{trunc}}, \\ 0, & \text{otherwise}, \end{cases}
\]

\( \sigma = 0.4 \) and \( \rho_{\text{trunc}} = 0.2 \). The unit square is divided into an even \( 5 \times 5 \) grid and each \( \mu_j \) is chosen so that each of the 25 Gaussians is centered at its corresponding grid square. The parameters are chosen as \( a(7) = 80, a(8) = 60, a(12) = 70, \) and \( a(j) = 0 \) otherwise. The environment created with these parameters can be seen in Figure 2-4c. The parameters \( \hat{a}, \Lambda \) and \( \lambda \) are initialized to zero. The parameters for the controller are \( K_i = 50, \forall i, \Gamma = \text{identity}, \gamma = 2000, W_n = 5, W_s = 150, w = 30 \). The spatial integrals are approximated by discretizing each Voronoi region into a \( 7 \times 7 \) grid and summing contributions of the integrand over the grid. Voronoi regions are computed using a decentralized algorithm similar to the one in [8].

The initial path is designed in a snake-like configuration “zig-zagging” across the environment (the blue line in Figure 2-4). We first allowed the robot to go through the initial path without reshaping it so that it could sample most of the space and learn the distribution of sensory information in the environment. Therefore, we present results in two separate phases: 1) learning phase, and 2) path shaping phase. The
learning phase corresponds to the robot traveling through its whole path once, without reshaping it, in order to learn the environment. The path shaping phase corresponds to when (2.6) is used to reshape the path into an informative path, and starts when the learning phase is done. In the path shaping phase, \( w = 0 \).

**Learning Phase**

The robot travels its entire path once, measuring the environment as it travels and using the adaptation law (2.23) to estimate the environment. This process can be seen in Figure 2-4. As the robot travels its path, the adaptation law causes \( \hat{\phi}(q) \to 0 \), \( \forall q \in Q \), as can be seen from Figures 2-4a, 2-4b, 2-4c, where the robot’s estimate of the environment (solid colored environment) converges to the real environment description (translucent environment) for all of the space. This means that the robot’s trajectory was rich enough to generate accurate estimates for all of the environment. Figure 2-2 shows that \( \lim_{t \to \infty} \int_0^t w(\tau) (\hat{\phi}_p(\tau))^2 d\tau = 0 \), in accordance with (ii) from Theorem 1. Finally, for this learning phase, we see in Figure 2-3 that the Lyapunov-like function \( V_1 \) is monotonically non-increasing, which supports our theory.

![Figure 2-2: Integral parameter error](image)

![Figure 2-3: Lyapunov-like function in learning phase](image)
Figure 2-4: Simulated single robot system with informative path controller during the learning phase. Left: 1) the path, shown as the blue line, connects all the waypoints, shown as black circles; 2) the black arrowhead represents the simulated robot. Right: the environment description, where the translucent environment represents the true environment and the solid environment represents the estimated environment.
Path Shaping Phase

Once the robot travels through its path once (and learns the environment), the controller from Section 2.4 is activated. We can see how the path evolves under this controller in Figure 2-7. The robot learns the location of the points of interest in the learning phase and, consequently, shapes its path to cover these points. After 200 iterations (Figure 2-7d), the path already tends to go through all dynamic regions of the environment, while avoiding all static regions. At 1000 iterations (Figure 2-7f), the path clearly only goes through the dynamic regions.

Figure 2-5 shows the true and estimated mean waypoint position errors, where the estimated error refers to the quantity $\|\hat{\alpha}(t)\|$. We can see that $\lim_{t \to \infty} \|\hat{\alpha}(t)\| = 0$, in accordance to (i) from Theorem 1. Figure 2-6 shows the Lyapunov-like function $V_1$ monotonically non-increasing during the path morphing phase and approaching a limit near 900 iterations. The initial value of this function in the path morphing phase is the final value of the function in the learning phase.
Figure 2-7: Simulated single robot system with informative path controller during the path shaping phase. The path, shown as the blue line, connects all the waypoints, shown as black circles. The black arrowhead represents the simulated robot.
2.6 Discussion

The selection of the controller parameters $W_s$ and $W_n$, can vary depending on the desired behavior of the system. For example, higher $W_n$ will pull waypoints closer together, which might drive the system to a more sub-optimal configuration for sensing. If, on the contrary, $W_s$ is very high, the system focuses more on the sensing task, and the waypoints will be more scattered. Although in this thesis we do not consider vehicle dynamics, one way to indirectly impose vehicle dynamics restrictions is increasing the neighbor distance weight $W_n$ and increase the number of waypoints in the path. This will result in smoother paths.

This adaptive controller is applicable to any sensing task done by a robot without full knowledge of the environment. In the next chapter we present an extension to persistent sensing tasks performed by a single robot.
Chapter 3

Persistent Informative Controller for a Single Robot

The main motivation behind this chapter\(^1\) is to generate an informative path that can be used by a robot tasked with persistent sensing. We now extend the adaptive controller from Chapter 2 and prove that it enables the robot to learn the environment dynamics, in the form of growth rates of the field over the environment, and generate an informative path for a robot with a finite sensor footprint to follow and sense the growing field, known as the accumulation function, to maintain the height of the field bounded. This extended controller is referred to as the persistent informative controller for a single robot.

3.1 Single Robot System Description for Persistent Sensing

The system description for persistent sensing by a single robot is identical to that in Section 2.1.1. Additionally, the robot is equipped with a sensor with a finite footprint. Examples of sensors with finite footprint are a camera for a surveillance task and a vacuum cleaner for a cleaning task. Although any footprint shape can be used, we use a

\(^1\)The majority of this chapter was published in [39].
constant circular footprint for simplicity, defined as \( F(p_r) = \{ q \in Q : \| q - p_r \| \leq \rho \} \), when the robot is at location \( p_r \), where \( \rho \) is a constant positive scalar.

### 3.2 Relation to Persistent Sensing Tasks

We use the stability criterion for a persistent sensing task, defined in [35], and plug it into the adaptive controller, such that the control action increases the stability margin of the persistent sensing task through time. Specifically, the stability criterion for a persistent sensing task when given the speed of the robot at each point along the path, i.e. the speed profile, is

\[
\phi(q) - \frac{\tau_c(q,t)}{T(t)} c(q) = s(q,t) < 0, \quad \forall q \mid \phi(q) > 0, \tag{3.1}
\]

where the constant scalar \( \phi(q) \) (the sensory function) is the rate at which the field grows at point \( q \), the constant scalar \( c(q) \) is the rate at which the field shrinks (is consumed) when the robot’s sensor is covering point \( q \), and \( c(q) > \phi(q), \forall q \). Also, \( T(t) \) is the time it takes the robot to complete the path at time \( t \), and \( \tau_c(q,t) \) is the time the robot’s sensor covers point \( q \) along the path at time \( t \). These last two quantities are calculated using the speed profile for the path at time \( t \) and the robot’s sensor footprint \( F(p_r) \). The stability margin of the system is \( S(t) = -(\max_q s(q,t)) \). A stable persistent task at time \( t \) is one in which \( S(t) > 0 \), which means the robot is able to maintain the height of the accumulation function bounded at all points \( q \). If \( S(t) > 0, \forall t \geq \tau \), for some \( \tau \), then the persistent sensing task is stable for all \( t \geq \tau \). Note that only points \( q \) that satisfy \( \phi(q) > 0 \) are considered in a persistent sensing task since it is not necessary to persistently sense a point that has no sensory interest. Points that satisfy this condition are referred to as points of interest.\(^2\)

Since the robot does not know the environment, but has an estimate of it, the

\(^2\)We assume the environment contains a finite number of points of interests. These finite points could be the discretization of a continuous environment.
robot uses the estimated version of (3.1), i.e.

\[ \hat{\phi}(q, t) - \frac{\tau_c(q, t)}{T(t)} c(q) = \dot{s}(q, t) < 0, \quad \forall q \mid \hat{\phi}(q, t) > 0, \]  

(3.2)

where \( c(q) \) is known by the robot, and since the robot knows the path and the speed profile, it knows \( T(t) \) and \( \tau_c(q, t), \forall(q) \). The robot also knows the estimated stability margin at time \( t \), defined as \( \hat{S}(t) = -(\max_q \dot{s}(q, t)) \).

In [35], a linear program was given which can calculate the speed profile for the path at time \( t \) that maximizes \( \hat{S}(t) \) (or \( S(t) \) for ground-truth). With the speed profiles obtained with this linear program, and using (3.2), we can formulate a new controller to drive the robot’s path in a direction to perform stable persistent sensing tasks. Hence, from this point on, we assume that this maximizing speed profile\(^3\) is known and used to obtain \( \dot{s}(q, t) \), \( \forall q \), \( \forall t \).

### 3.3 Persistent Informative Controller

Let the waypoints have new dynamics of the form

\[ \dot{p}_i = I_i u_i, \]  

(3.3)

where \( u_i \) is defined in (2.17),

\[ I_i = \begin{cases} 0, & \text{if } \frac{\partial \dot{s}}{\partial p_i} u_i < 0 \text{ and } t - t_u^i > \tau_{dwell}, \\ 1, & \text{otherwise,} \end{cases} \]  

(3.4)

\( \tau_{dwell} \) is a design parameter, and \( t_u^i \) is the most recent time at which \( I_i \), switched from zero to one (switched “up”). Equations (3.3) and (3.4) look at how each waypoint movement affects the estimated stability margin through time, and ensure that this quantity does not decrease.

**Remark 3** For positive \( \epsilon \to 0 \), \( \frac{\partial \dot{s}}{\partial p_i}(t) \) is not always defined when \( \arg \max_q \dot{s}(q, t - \epsilon) \neq \).

\(^3\)From [35], for a given path, this speed profile makes the robot’s trajectory a periodic one.
arg max_q \dot{s}(q, t + \epsilon). In such cases \frac{\partial \dot{s}}{\partial \dot{p}_i}(t) refers to \frac{\partial \dot{s}}{\partial \dot{p}_i}(t + \epsilon).

**Theorem 2 (Convergence Theorem for Persistent Sensing - Single Robot)**

Under Assumption 1, with waypoint dynamics specified by (3.3), control law specified by (2.17), and adaptive law specified by (2.23), we have

1. \( \lim_{t \to \infty} I_i(t) \| \dot{M}_i(t) \dot{e}_i(t) + \alpha_i(t) \| = 0, \quad \forall i \in \{1, \ldots, n\} \),

2. \( \lim_{t \to \infty} \| \dot{\phi}_p(\tau) \| = 0, \quad \forall \tau : w(\tau) > 0. \)

**Proof 2** We define a Lyapunov-like function based on the robot's path and environment estimate, and prove asymptotic stability of the system to a locally optimal equilibrium. However, contrary to the previous section, a bit more work is needed to prove property (i) due to the piecewise differentiability of the new Lyapunov-like function caused by the new waypoint dynamics.

Let \( V_2 \) be the new Lyapunov-like function, and let \( V_2 = V_1 \) from (2.25). Then, following the procedure from Section 2.4, but with \( p_i \) defined by (3.3), we get

\[
\dot{V}_2 = \sum_{i=1}^{n} -\frac{1}{\beta_i} (\dot{M}_i \dot{e}_i + \alpha_i)^T I_i K_i (\dot{M}_i \dot{e}_i + \alpha_i) - \gamma \int_0^t w(\tau) (\dot{\phi}_p(\tau))^2 d\tau - \dot{a}^T I_{proj} \dot{a}_{proj}. (3.5)
\]

Using a similar analysis as in Section 2.4, denote the three terms in (3.5) as \(-\xi_1(t), -\xi_2(t)\) and \(-\xi_3(t)\), so that \( \dot{V}_2(t) = -\xi_1(t) - \xi_2(t) - \xi_3(t) \). In Section 2.4 we showed that \( \xi_1(t) \geq 0, \xi_2(t) \geq 0, \) and \( \xi_3(t) \geq 0 \). Additionally, the time integral of each of these three terms, \( \lim_{t \to \infty} \int_0^t \xi_k(\tau) d\tau \) exists and is finite for all \( k \). It was shown in [34] (Lemma 1) that \( \xi_2(t) \) is uniformly bounded, therefore \( \xi_2(t) \) is uniformly continuous. Hence, by Barabat's Lemma [14], \( \xi_2(t) \to 0. \) This implies (ii).

Now, as proven before, \( \int_0^\infty \xi_1(\tau) d\tau \) exists and is finite, and we want to show that \( \lim_{t \to \infty} \xi_1(t) = 0. \) Let \( \xi'_1 \) be defined such that \( \xi_1 = \sum_{i=1}^{n} \xi'_1. \) Let us assume that \( \lim_{t \to \infty} \xi'_1(t) \neq 0, \) that is, \( \forall t \exists j \geq t, \epsilon > 0 \), such that \( \xi'_1(t_j) \geq \epsilon. \) Let \( \{t_j\}_{j=1}^\infty \) be the infinite sequence of \( t_j \) 's separated by more than \( 2\tau_{\text{dwell}} \) such that \( \xi'_1(t_j) \geq \epsilon. \) That is, \( |t_j - t_{j'}| > 2\tau_{\text{dwell}}, \forall j \neq j', \) and \( t_j, t_{j'} \in \{t_j\}_{j=1}^\infty. \) Since, from [34] (Lemma 1), \( \dot{\xi}'_1(t) \) is uniformly bounded by some value \( B \) when \( I_i = 1 \) (i.e. \( |\dot{\xi}'_1(t)| \leq B), \) and whenever
Figure 3-1: Geometric representation

$I_i = 1$, it remains with this value for at least $\tau_{dwell}$, then we have that $\forall t_j \in \{t_j\}_{j=1}^{\infty}$,

$$\int_{t_j-\tau_{dwell}}^{t_j+\tau_{dwell}} \xi_1^i(\tau) d\tau \geq \epsilon \delta > 0,$$

(3.6)

where

$$\delta = \min\{\frac{\epsilon}{2B}, \frac{\tau_{dwell}}{2}\} > 0.$$

(3.7)

This can be see graphically in Figure 3-1, where the red colored region has an area of $\epsilon \tau_{dwell}/2$ and the union of the blue and red colored regions has an area of $\epsilon^2/(2B)$. Then

$$\int_0^\infty \xi_1^i(\tau) d\tau \geq \sum_{t_j \in \{t_j\}_{j=1}^{\infty}} \int_{t_j-\tau_{dwell}}^{t_j+\tau_{dwell}} \xi_1^i(\tau) d\tau \geq \sum_{t_j \in \{t_j\}_{j=1}^{\infty}} \epsilon \delta,$$

(3.8)

which is infinite, and contradicts that $\int_0^\infty \xi_1(\tau) d\tau = \sum_{i=1}^n \int_0^\infty \xi_1^i(\tau) d\tau$ exists and is finite, which has already been proven. Therefore, by contradiction, $\lim_{t \to \infty} \xi_1^i(t) = 0$, hence property (i) holds.

**Remark 4** The stability margin can theoretically worsen while $I_i$, for some $i$, cannot switch from one to zero because it is waiting for $t - t_u > \tau_{dwell}$. However, $\tau_{dwell}$ can be selected arbitrarily small and, in practice, any computer will enforce a $\tau_{dwell}$ due to discrete time steps. Therefore, it is not a practical restriction. As a result,
intuitively, (i) from Theorem 2 means that \( \lim_{t \to \infty} \| \tilde{M}_i(t) \hat{e}_i(t) + \alpha_i(t) \| = 0 \) only if this helps the persistent sensing task. Otherwise \( \lim_{t \to \infty} I_{i}(t) = 0 \), meaning that the persistent sensing task will not benefit if the \( i^{th} \) waypoint moves.

3.4 Simulation, Results and Implementation

3.4.1 System Architecture

The system architecture is the same as in Figure 2-1 from Section 2.5.1, but the algorithm for layer 2 changes slightly. The new algorithm for layer 2 can be seen in Algorithm 3.

3.4.2 Numerical Simulation

The persistent informative controller for a single robot was simulated in a MATLAB environment for many test cases. Here we present a case for \( n = 70 \) waypoints. A time step of 0.01 seconds was used, and \( \tau_{\text{dwell}} = 0.009 \). The parameters for the environment where \( \sigma = 0.18 \) and \( \rho_{\text{trunc}} = 0.1 \), \( a(6) = 20 \), \( a(7) = 10 \), \( a(8) = 16 \), \( a(14) = 10 \), \( a(17) = 16 \) and \( a(j) = 0 \) otherwise. The environment created with these parameters can be seen in Figure 3-4c. The parameters \( \hat{a} \), \( \hat{A} \) and \( \lambda \) were initialized to zero. The

---

**Algorithm 3** Persistent Informative Controller for a Single Robot:

Layer 2 for the \( i^{th} \) waypoint

1. Initially compute the value of \( \hat{S} \)
2. **loop**
   3. Compute the waypoint’s Voronoi partition
   4. Compute \( \hat{C}_i \) according to (2.12)
   5. Obtain neighbor waypoint locations \( p_{i-1} \) and \( p_{i+1} \)
   6. Compute \( u_i \) according to (2.17)
   7. Compute \( I_i \) according to (3.4)
   8. Update \( p_i \) according to (3.3)
3. **end loop**
controller parameters were $K_i = 90$, $\forall i$, $\Gamma = \text{identity}$, $\gamma = 2000$, $W_n = 3$, $W_s = 50$, $w = 1$, and $\rho = 0.05$. The environment was discretized into a $12 \times 12$ grid and only points in this grid that satisfied $\tilde{\phi}(\mathbf{q}) > 0$ were used as points of interest in (3.2). By only using this discretized version of the environment, the running time for experiments is greatly reduced. For more sensitive systems, this grid can be refined.

The initial path was designed in a snake-like configuration "zig-zagging" across the environment (the blue line in Figure 3-4). Again, we present results in two separate phases: 1) learning phase, and 2) path shaping phase, and described in Section 2.5.2. However, in the path shaping phase (3.3) is used to reshape the path into an informative path.

**Learning Phase**

The robot travels its entire path once, measuring the environment as it travels and using the adaptation law (2.23) to estimate the environment. This process can be seen in Figure 3-4. As the robot travels its path, the adaptation law causes $\tilde{\phi}(\mathbf{q}) \to 0$, $\forall \mathbf{q} \in Q$, as can be seen from Figures 3-4a, 3-4b, 3-4c, where the robot’s estimate of the environment converges to the real environment description for all of the space. Figure 3-2 shows that $\lim_{t \to \infty} \int_0^t w(\tau)(\tilde{\phi}_{r_e}(\tau))^2d\tau = 0$, in accordance with (ii) from Theorem 2. Finally, we see in Figure 3-3 that the Lyapunov-like function $V_2$ is monotonically non-increasing.

---

**Figure 3-2:** Integral parameter error

**Figure 3-3:** Lyapunov-like function in learning phase
Figure 3-4: Simulated single robot system with persistent informative controller during the learning phase. Left: 1) the path, shown as the blue line, connects all the waypoints, shown as black circles; 2) the points of interest are shown as green dots; and 3) the black arrowhead represents the simulated robot, and its sensor footprint is represented by the blue circle around the robot's position. Right: the environment description, where the translucent environment represents the true environment and the solid environment represents the estimated environment.
Path Shaping Phase

We can see how the path evolves under this controller in Figure 3-7. The robot learns the location of the points of interest in the learning phase and, consequently, shapes its path to cover these points. At 300 iterations (Figure 3-7f), the path enables the robot to perform a locally optimal persistent sensing task.

Figure 2-5 shows the true and estimated versions of the mean waypoint errors for persistent sensing, referring to the mean of the quantity \( I_t(t) \| \hat{M}_t(t) \hat{e}_t(t) + \alpha_t(t) \|. \) We can see that \( \lim_{t \to \infty} I_t(t) \| \hat{M}_t(t) \hat{e}_t(t) + \alpha_t(t) \| = 0, \) in accordance to (i) from Theorem 2. Figure 2-6 shows the Lyapunov-like function \( V_2 \) monotonically non-increasing during the path morphing phase and settling to a local minimum, meaning that the path is driven to a locally optimal configuration that is helpful for the persistent sensing task. The initial value of this function in the path morphing phase is the final value of the function in the learning phase.

Finally, Figure 3-8 shows the persistent sensing task’s stability margin evolving through time. Both the true and estimated stability margins are shown. The true plot provides ground-truth, showing that the robot’s estimates were a good representation of true values. The stability margin starts off with a negative value, indicating that the persistent sensing task is initially unstable, and increases through time, indicating that the path is morphing to make the persistent sensing task “more stable”. By the end of the simulation, the persistent sensing task is stable, i.e. the stability margin is positive. The discrete jumps in the stability margin are due to the discrete time steps. These jumps can be minimized by performing smaller steps.
Figure 3-7: Simulated single robot system with persistent informative controller during the path shaping phase. The path, shown as the blue line, connects all the waypoints, shown as black circles. The points of interest are shown as green dots. The black arrowhead represents the simulated robot, and its sensor footprint is represented by the blue circle around the robot's position.
3.4.3 Implementation of Single-Robot System

The persistent informative controller for a single robot was implemented with a quadrotor robot. We tested a number of different cases for different environments, and all generated practically identical results to their simulated counterparts. Here we present the hardware implementation with the same simulated environment as described in Section 3.4.2 and a path consisting of 70 waypoints. The environment is simulated and, hence, the measurements by the robot are simulated, but the locations of these measurements correspond to the locations of the physical quadrotor. The path shaping phase was run for 375 iterations. Results for the implementation are practically identical to simulated results and, thus, are not presented. Figures 7-12a, 7-12b, 7-12c show snapshots of the implementation at different iteration values. Notice in Figure 7-12c that the final informative path is practically identical to that in Figure 3-7f from the simulations.

3.5 Discussion

This persistent informative controller is identical to the informative path controller from Section 2.4 while the path shaping is beneficial to the persistent sensing task. Only when this condition is not satisfied is when the control action becomes different by not allowing the waypoints to move. In some scenarios, the informative path that

---

4A more detailed description of the hardware implementation is found in Chapter 7.
Figure 3-9: Hardware implementation of the single-robot system using a quadrotor robot. Three snapshots of the path shaping phase at different iteration values are shown. The path, shown as the black line, connects all the waypoints. The points of interest in the environment are shown as purple dots. The robot is the green-lit quadrotor, and its sensor footprint is represented by the green circle under the robot’s position.

The path was generated by the controller in Section 3.3 was very similar to the informative path generated by the controller from Section 2.4. On others, the informative paths from both controllers were very different. This is due to the additional restriction of the non-decreasing stability margin. This restriction may cause the system to get stuck on a local minimum early in the path shaping phase because otherwise it would generate a decrease in the stability margin of the persistent task. One possible way to not get stuck early on a local minimum for persistent sensing is to initially have the robot use the informative path controller and, after some time, use the persistent informative controller to improve the persistent sensing task. The controller weights $W_s$ and $W_n$ can also impact how the controllers behave, and should be tuned to achieve...
the desired behavior. In the next chapters, we focus on generating informative paths when the system is comprised of multiple robots.
Chapter 4

Informative Path Controller for Multiple Robots

In many scenarios, using a team of robots working together to achieve a task generates better results than using a single robot. Therefore, for the following chapters, we are interested in generating informative paths for multiple robots. The problem description and notation are very similar to that in Chapter 2. In this chapter we redefine some term in order to extend the informative path controllers from Chapter 2 to multiple robots.

We assume we are given multiple robots whose tasks are to sense an unknown dynamical environment while traveling along their individual closed paths, each consisting of a finite number of waypoints. The goal is for the robots to identify the areas within the environment that are dynamic and compute paths that allow them to jointly sense the dynamic areas. A formal mathematical description of the problem follows.

4.1 Multi-Robot System - Problem Setup

Let there be $N$ robots identified by $r \in \{1, \ldots, N\}$. Robot $r$ is located at position $p_r \in Q$ and travels along its closed path consisting of a finite number $n(r)$ of waypoints. The position of the $i^{th}$ waypoint in robot $r$’s path is denoted $p_{ri}$. 

Let \{p^r_1, \ldots, p^r_{n(r)}\} be the configuration of robot \(r\)'s path and let \(V^r_i\) be a Voronoi partitions of \(Q\), with the \(i^{th}\) waypoint position in robot \(r\)'s path as the generator point, defined as

\[
V^r_i = \{q \in Q : ||q - p^r_i|| \leq ||q - p^r_{i'}||, \forall (r', i') \neq (r, i)\},
\]

where \(r, r' \in \{1, \ldots, N\}, i \in \{1, \ldots, n(r)\}\) and \(i' \in \{1, \ldots, n(r')\}\). (4.1)

Since each path is closed, each waypoint \(i\) along robot \(r\)'s path has a previous waypoint \(i - 1\) and next waypoint \(i + 1\) related to it, which are referred to as the neighbor waypoints of \(i\). Note that for each \(r\), \(i + 1 = 1\) for \(i = n(r)\), and \(i - 1 = n(r)\) for \(i = 1\). Once a robot reaches a waypoint, it continues to move to the next waypoint along its path, in a straight line interpolation.

We assume that the network of robot's in the system is a connected network, i.e. that the graph where each robot is a node and an edge represents communication between two nodes is a connected graph. This connected network corresponds to the robots’ abilities to communicate with each other. Finally, the environment structure is identical to that described in Section 2.1.2.

### 4.2 Locational Optimization

We can formulate the cost incurred by the multi-robot system over the region \(Q\) as

\[
\mathcal{H}_2 = \sum_{r=1}^{N} \sum_{i=1}^{n(r)} \int_{V^r_i} \frac{W_s}{2} ||q - p^r_i||^2 \phi(q) dq + \sum_{r=1}^{N} \sum_{i=1}^{n(r)} \frac{W_n}{2} ||p^r_i - p^r_{i+1}||^2.
\] (4.2)

A formal definition of informative paths for multiple robots follows.

**Definition 2 (Informative Paths for Multiple Robots)** A collection of informative paths for a multi-robot system corresponds to a set of waypoint locations for each robot that locally minimize (4.2).

The mass, first mass-moment, and centroid of the Voronoi partitions for the multi-robot system are represented by \(M^r_i\), \(L^r_i\) and \(C^r_i\), respectively, and are defined by
(2.2), (2.3) and (2.4), respectively, but integrating over the new Voronoi partition $V_i^r$. Letting $e_i^r = C_i^r - p_i^r$ and, following the same procedure as in Section 2.2, we have that

$$\frac{\partial H_2}{\partial p_i^r} = -M_i^r e_i^r - W_n(p_{i+1}^r + p_{i-1}^r - 2p_i^r). \quad (4.3)$$

Assigning to each waypoint dynamics of the form

$$\ddot{p}_i^r = u_i^r, \quad (4.4)$$

where $u_i^r$ is the control input, we propose the following control law for the waypoints to converge to an equilibrium configuration:

$$u_i^r = \frac{K_i^r(M_i^r e_i^r + \alpha_i^r)}{\beta_i^r}, \quad (4.5)$$

where

$$\alpha_i^r = W_n(p_{i+1}^r + p_{i-1}^r - 2p_i^r),$$

$$\beta_i^r = M_i^r + 2W_n > 0,$$

and $K_i^r$ is a uniformly positive definite matrix.

### 4.3 Sensory Function Approximation

We assume the matching condition from Assumption 1. Let $\hat{a}_r(t)$ be robot $r$’s approximation of the parameter vector $a$. Then, $\hat{\phi}_r(q) = \mathcal{K}(q)^T \hat{a}_r$ is robot $r$’s approximation of $\phi(q)$. The mass moment approximations are represented by $\hat{M}_i^r$, $\hat{L}_i^r$ and $\hat{C}_i^r$, and are defined by (2.10), (2.11) and (2.12), respectively, but integrating over the new Voronoi partition $V_i^r$ and using the respective robot’s parameter vector estimate $\hat{a}_r$. 

63
Similarly, let $\ddot{a}_r = \hat{a}_r - a$ and

\[
\begin{align*}
\ddot{\phi}_r(q) &= \hat{\phi}_r(q) - \phi(q) = K(q)^T \dot{a}_r, \\
\ddot{M}_i^r &= \hat{M}_i^r - M_i^r = \int_{V_i^r} W_s K(q)^T dq \dot{a}_r, \\
\ddot{L}_i^r &= \hat{L}_i^r - L_i^r = \int_{V_i^r} W_s qK(q)^T dq \dot{a}_r, \\
\ddot{C}_i^r &= \hat{C}_i^r - C_i^r.
\end{align*}
\] (4.6, 4.7, 4.8, 4.9)

### 4.4 Informative Path Controller

We design an adaptive control law very similar to the one in Section 2.4 and prove that it causes the multiple paths to converge to a locally optimal configuration according to (4.2), while causing all of the robots’ estimates of the environment to converge to the real environment description by integrating their sensory measurements along their trajectories and by using a consensus term between the robots. All of the robots’ estimates of the environment converge to a same estimate due to a consensus term [34] used in their adaptive laws.

Since the robots do not know $\phi(q)$, but have estimates $\hat{\phi}_r(q)$, the control law from (4.5) becomes

\[
u_i^r = \frac{K_i^r (\hat{M}_i^r \hat{e}_i^r + \alpha_i^r)}{\hat{\beta}_i^r}.
\] (4.10)

where

\[
\begin{align*}
\alpha_i^r &= W_n (p_{i+1}^r + p_{i-1}^r - 2 p_i^r), \\
\hat{\beta}_i^r &= \hat{M}_i^r + 2 W_n, \\
\hat{e}_i^r &= \hat{C}_i^r - p_i^r.
\end{align*}
\]
The parameter $\hat{a}_r$ is adjusted according to

$$\Lambda_r = \int_0^t w_r(\tau)\mathcal{K}_{p_\rho}(\tau)\mathcal{K}_{p_\rho}(\tau)^T d\tau,$$  \hspace{1cm} (4.11)

$$\lambda_r = \int_0^t w_r(\tau)\mathcal{K}_{p_\rho}(\tau)\phi_{p_\rho}(\tau) d\tau,$$  \hspace{1cm} (4.12)

where

$$w_r(t) = \begin{cases} \text{positive constant scalar}, & \text{if } t < t_{wr}, \\ 0, & \text{otherwise}, \end{cases}$$  \hspace{1cm} (4.13)

and $t_{wr}$ is some positive time at which part of the the adaptation for robot $r$ shuts down to maintain $\Lambda_r$ and $\lambda_r$ bounded. Let

$$b_r = \sum_{i=1}^{n(r)} \int_{V_i} W_s\mathcal{K}(q)(q - p_i)^T dq \dot{p}_i,$$  \hspace{1cm} (4.14)

$$\dot{\hat{a}}_{pre} = -b_r - \gamma(\Lambda_r\hat{a}_r - \lambda_r) - \zeta \sum_{r'=1}^{N} l_{r,r'}(\hat{a}_r - \hat{a}_{r'}),$$  \hspace{1cm} (4.15)

where $\zeta > 0$ is a consensus scalar gain, and $l_{r,r'}$ can be interpreted as the strength of the communication between robots $r$ and $r'$ and is defined as

$$l_{r,r'} = \begin{cases} D_{\max} - \|p_r - p_{r'}\|, & \text{if } \|p_r - p_{r'}\| \leq D_{\max} \\ 0, & \text{otherwise}. \end{cases}$$  \hspace{1cm} (4.16)

Since $a(j) \geq 0, \forall j$, we enforce $\hat{a}_r(j) \geq 0, \forall r, \forall j$, by the projection law [34],

$$\dot{a}_r = \Gamma(\hat{a}_{pre} - I_{proj,\hat{a}_{pre}}),$$  \hspace{1cm} (4.17)

where $\Gamma \in \mathbb{R}^{m \times m}$ is a diagonal positive definite adaptation gain matrix, and the
diagonal matrix $I_{proj}$ is defined element-wise as

$$
I_{proj}\,(j) = \begin{cases} 
0, & \text{if } \hat{a}_r(j) > 0, \\
0, & \text{if } \hat{a}_r(j) = 0 \text{ and } \hat{a}_{pres}(j) \geq 0, \\
1, & \text{otherwise}, 
\end{cases}
$$

(4.18)

where $(j)$ denotes the $j^{th}$ element for a vector and the $j^{th}$ diagonal element for a matrix.

**Theorem 3 (Convergence Theorem for Multiple Robots)** Under Assumption 1, with waypoint dynamics specified by (4.4), control law specified by (4.10) and adaptive law specified by (4.17), we have

1. $\lim_{t \to \infty} \|\hat{M}^r_i(t)\dot{e}^r_i(t) + \alpha^r_i(t)\| = 0$, $\forall r \in \{1, \ldots, N\}, \forall i \in \{1, \ldots, n(r)\}$,

2. $\lim_{t \to \infty} \|\tilde{\phi}_{pre}(\tau)\| = 0$, $\forall r \in \{1, \ldots, N\}, \forall \tau \mid w_r(\tau) > 0$,

3. $\lim_{t \to \infty}(\hat{a}_r - \hat{a}_{r'}) = 0$, $\forall r, r' \in \{1, \ldots, N\}$.

**Proof 3** We define a Lyapunov-like function based on the robots' paths and environment estimates, and use Barbalat's lemma to prove asymptotic stability of the system to a locally optimal equilibrium.

Let

$$
\mathcal{V}_3 = \mathcal{H}_2 + \frac{1}{2} \sum_{r=1}^{N} \ddot{a}_r^T \Gamma^{-1} \dot{a}_r. 
$$

(4.19)

Following a similar procedure to the one in Section 2.4, we have that

$$
\dot{\mathcal{V}}_3 = \sum_{r=1}^{N} \sum_{i=1}^{n(r)} - \frac{1}{\beta^r_i} (\dot{M}^r_i \dot{e}^r_i + \alpha^r_i)^T K^r_i (\dot{M}^r_i \dot{e}^r_i + \alpha^r_i) - \gamma \sum_{r=1}^{N} \int_{0}^{t} w_r(\tau) (\dot{\phi}_{pre}(\tau))^2 \, d\tau \\
- \sum_{r=1}^{N} \ddot{a}_r^T I_{proj} \ddot{a}_{pres} - \zeta \sum_{r=1}^{N} \ddot{a}_r^T \sum_{r'=1}^{N} l_{r,r'}(\dot{a}_r - \dot{a}_{r'}). 
$$

(4.20)
Let \( \mathbf{1} = [1, \ldots, 1]^T \). Then, from [34] we can represent the last term in (4.20) as

\[
-\zeta \sum_{j=1}^m \hat{\Omega}_j^T L(t) \hat{\Omega}_j,
\]

where \( \Omega_j = \alpha(j) \mathbf{1} \), \( \hat{\Omega}_j = [\hat{\alpha}_1(j), \ldots, \hat{\alpha}_N(j)]^T \) and \( \hat{\Omega}_j = \hat{\Omega}_j - \Omega_j \). Also, \( L(t) \) is the weighted graph Laplacian of the system at time \( t \) and is defined element-wise by

\[
L(r, r') = \begin{cases} 
-\ell_{r, r'}, & \text{for } r \neq r', \\
\sum_{r' = 1}^N \ell_{r, r'}, & \text{for } r = r'.
\end{cases}
\]

From [34] we know that for a connected network like ours, \( x^T L x \geq 0 \forall x \), and \( x^T L x = 0 \) implies that \( x = 0 \) or \( x = v \mathbf{1} \), for some \( v \in \mathbb{R} \). Additionally,

\[
\Omega_j^T L = \alpha(j) \mathbf{1}^T L = 0, \forall j.
\]

Therefore,

\[
-\zeta \sum_{j=1}^m \hat{\Omega}_j^T L \hat{\Omega}_j = -\zeta \sum_{j=1}^m \hat{\Omega}_j^T L \hat{\Omega}_j,
\]

and we have that

\[
\dot{\mathbf{v}}_3 = \sum_{r=1}^N \sum_{i=1}^{n(r)} -\frac{1}{\beta_i^r} (\dot{M}_i^r \dot{e}_i^r + \alpha_i^T (\dot{M}_i^r \dot{e}_i^r + \alpha_i^r) - \gamma \sum_{r=1}^N \int_0^t w_r(\tau)(\dot{\phi}_p(\tau))^2 d\tau \\
- \sum_{r=1}^N \hat{\alpha}_r^T I_{\text{proj}} \dot{\alpha}_r - \zeta \sum_{j=1}^m \hat{\Omega}_j^T L \hat{\Omega}_j. \tag{4.21}
\]

Denote the four terms in (4.21) as \( \xi_1(t) \), \( \xi_2(t) \), \( \xi_3(t) \) and \( \xi_4(t) \), so that \( \dot{\mathbf{v}}_3(t) = \xi_1(t) + \xi_2(t) + \xi_3(t) + \xi_4(t) \). It was shown in Section 2.4 that \( \xi_k(t) \leq 0 \), for \( k = 1, 2, 3 \). For the fourth term, \( L(t) \) is, in general, a time varying function since communication between robots can change. However, since we assume a connected network, we have that \( L(t) \geq 0 \forall t \). Therefore, \( \xi_4(t) \leq 0 \).

Following the reasoning in Section 2.4, we have that \( \lim_{t \to \infty} \int_0^t \xi_k(\tau) d\tau \) exists and
is finite for all $k$. Furthermore, it was shown in [34] (Lemma 1) that $\dot{\xi}_1(t)$ and $\dot{\xi}_2(t)$ are uniformly bounded, and in [34] (Lemma 2) that $\dot{\xi}_4(t)$ is uniformly bounded. Therefore $\xi_1(t)$, $\xi_2(t)$ and $\xi_4(t)$ are uniformly continuous. Hence, by Barbalat's Lemma [14], $\xi_1(t) \to 0$, $\xi_2(t) \to 0$ and $\xi_4(t) \to 0$. The first two statements imply (i) and (ii). The third statement implies that $\hat{\Omega}_j^T L \hat{\Omega}_j \to 0$, $\forall j$. From the definition of the weighted graph Laplacian, this means that $\hat{\Omega}_j \to \hat{a}_{\text{final}}(j)$, $\forall j$, where $\hat{a}_{\text{final}} \in \mathbb{R}^{m}_{>0}$ is the final common parameter estimate vector shared by all robots. This implies (iii).

Properties (ii) and (iii) from Theorem 3 together imply that the sensory function estimate error for all robots will converge to zero for all points on any robot's trajectory with positive weight $w_r(t)$. This means that the robots will learn the true sensory function for the environment if the union of their trajectories while their weights are positive is rich enough. Therefore, we can design the initial waypoint locations such that, between all robots, most of the dynamic unknown environment is explored (see Figures 4-5 and 5-4).

4.5 Simulation and Results

4.5.1 System Architecture

The system architecture for our simulations can be seen in Figure 4-1 and is divided into two layers. The first layer (top of Figure 4-1 and colored in red) corresponds to
Algorithm 4 Informative Path Controller for Multiple Robots: 
Layer 1 for robot $r$

Require: $\phi(q)$ can be parametrized as in (2.8)
Require: $a \geq 0$
Require: Waypoints move slow enough that the robot can reach them and travel its path
Require: The network of robots is connected
Require: Robot knows location of $p_i^r$, $\forall i \in \{1, \ldots, n(r)\}$

1: Initially robot is moving towards $p_i^r$
2: Initialize $A_r$ and $\lambda_r$ to zero
3: Initialize $\hat{a}_r$ to some non-negative value
4: loop
5: if robot reached $p_i^r$ then
6: move towards $p_{i+1}^r$ in a straight line from $p_i^r$
7: else
8: move towards $p_i^r$ in a straight line from $p_r$
9: end if
10: Make measurement $\phi(p_r)$
11: Obtain $\hat{a}_r'$, $\forall r'$ that can communicate to $r$
12: Update $\hat{a}_r$ according to (4.17)
13: Update $A_r$ and $\lambda_r$ according to (4.11) and (4.12)
14: end loop

The robots traveling through their paths, and sampling and estimating the environment. A practical algorithm for this first layer is shown in Algorithm 4 and can be executed in a distributed way by each robot, in the sense that each robots can execute this algorithm independently, only sharing their current parameter estimate vector.

The second layer (bottom of Figure 4-1 and colored in blue) corresponds to the waypoints moving an placing themselves in locally optimal locations to create informative paths. We can think of each waypoint as an agent that has its own controller detailed in Algorithm 5. This algorithm can be implemented in a distributed way by each waypoint in the sense that the waypoints can execute this algorithm independently, only sharing information with their neighboring waypoints and enough information for all waypoints to compute their Voronoi cells.
Algorithm 5 Informative Path Controller for Multiple Robots:
Layer 2 for robot $r$’s $i^{th}$ waypoint

Require: Each waypoint knows $a_r$ from Algorithm 4
Require: Each waypoint can calculate its Voronoi partition
Require: Each waypoint knows its neighboring waypoints

1: loop
2: Compute the waypoint’s Voronoi partition
3: Compute $\tilde{C}_i^r$ according to (2.12), but integrating over $V_i^r$ from (4.1)
4: Obtain neighbor waypoint locations $p_{i-1}^r$ and $p_{i+1}^r$
5: Compute $u_i$ according to (4.10)
6: Update $p_i^r$ according to (4.4)
7: end loop

4.5.2 Numerical Simulation

The informative path controller for multiple robots was simulated in a MATLAB environment for many test cases. Here we present a case for $N = 10$ robots, $n(r) = 28$ waypoints, $\forall r$. The same fixed-time step numerical solver is used with a time step of 0.01 seconds. The environment parameters are $\sigma = 0.4$, $\rho_{\text{trunc}} = 0.2$, and $a = 60(1)$. The environment created with these parameters can be seen in Figure 4-5c. The parameters $a_r$, $\Lambda_r$ and $\lambda_r$, for all $r$ are initialized to zero. The parameters for the controller are $K_i^r = 90$, $\forall i, r$, $\Gamma = $ identity, $\gamma = 2000$, $W_n = 70$, $W_s = 500$, and $w_r = 10$, $\forall r$. In addition, $D_{\text{max}}$ is assumed to be very large, so that $l_{r,r'}(t) = 10$, $\forall r, r'$, $\forall t$.

All robots initially have approximately the same path, which is designed in a snake-like configuration “zig-zagging” across the environment (colored lines in Figure 4-5). All robots initially sweep most of the environment. Again, we present results in the two separate phases: 1) learning phase, and 2) path shaping phase, as described in Section 2.5.2. The path shaping phase starts after the learning phase is done by all robots. In the path shaping phase, $w_r = 0$, $\forall r$, and (4.4) is used to reshape the path.
Learning Phase

The robots travel their entire path once, measuring the environment as they travel and using the adaptation law (4.17) to estimate the environment. As the robots travel their paths, the adaptation laws cause $\tilde{\phi}_r(q) \to 0$, $\forall q \in Q$, $\forall r$. This process can be seen in Figure 4-5, where the estimated environment for two robots is presented on the right side. As can be seen, the estimated environments converge to the real environment. This means that the union of all robots' trajectories was rich enough to generate accurate estimates for all of the environment. Figure 4-2 shows that the mean over all robots of $\int_0^T w_r(\tau)(\tilde{\phi}_r(\tau))^2 d\tau$ converges to zero, in accordance with (ii) from Theorem 3. Figure 4-4 shows that the consensus error, referring to $\zeta \sum_{r=1}^N \bar{a}_r^T \sum_{r'=1}^N l_{r,r'}(\bar{a}_r - \bar{a}_{r'})$ converges to zero, in accordance with (iii) from Theorem 3, meaning that all robots have the same estimate of the environment (which converges to the real environment). Finally, for this learning phase, we see in Figure 4-3 that the Lyapunov-like function $V_3$ is monotonically non-increasing.
Figure 4-5: Simulated multi-robot system with informative path controller during the learning phase. Left: 1) the paths, shown as colored lines, connect all the waypoints, shown as black circles, and each robot has a different color assigned to it; 2) the black arrowheads represent the simulated robots. Right: the environment description for two of the 10 robots, where the translucent environment represents the true environment (common to all robots) and the solid environment represents the individual estimated environments.
Path Shaping Phase

Once the robots travel through their paths once, the controller from Section 4.4 is activated. We can see how the path evolves under this controller in Figures 4-8a to 4-8f. Figure 4-6 shows the true and estimated mean waypoint position errors, where the estimated error refers to the quantity $\|\hat{M}_i(t)\hat{e}_i^r(t) + \alpha_i^r(t)\|$. As shown, $\lim_{t \to \infty} \|\hat{M}_i(t)\hat{e}_i^r(t) + \alpha_i^r(t)\| = 0$, $\forall i, r$ in accordance to (i) from Theorem 3. Figure 4-7 shows the Lyapunov-like function $V_3$ monotonically non-increasing during the path morphing phase and reaching a local minimum, meaning that the robots have achieved informative paths to sense the environment. The initial value of this function in the path morphing phase is the final value of the function in the learning phase.

Note that the environment is approximately uniform in sensory information, i.e. every point in the environment has approximately the same sensory information. Because of this, the robots generate paths that, between all of them, cover all of the environment evenly. If portions of the environment were static (with no sensory information), then the informative paths would tend to not go through those regions.
Figure 4-8: Simulated multi-robot system with informative path controller during the path shaping phase. The paths, shown as the colored lines, connects all the corresponding waypoints, shown as black circles. Each robot follows a path with a unique color. The black arrowheads represent the simulated robots.
4.6 Discussion

The robots using this controller work together to cover an unknown environment due to a coupling that exists between them in the Voronoi partitions. Therefore, a robot will not tend to cover the same spot that another robot is already covering because they have different Voronoi partitions. However, it is important to point out that the informative paths generated by our controller depend on the parameters inserted into the controller and the initial paths assigned to each robot. We now compare some cases where such variations produce big differences in the final informative paths.

4.6.1 Variations of Neighbor Distance Weight $W_n$

Due to the coupling of waypoints within each path, if the value of the neighbor distance weight $W_n$ is high enough the paths will tend to shape in a way that creates short paths for each robot. This fact also applies to the single robot system from Chapter 2. In this section we present two cases that are identical with the exception of the value of $W_n$. For Case 1, $W_n = 30$ and for Case 2, $W_n = 3$. All other parameters are the same for both cases: $W_s = 150$, $N = 2$, $n(r) = 40 \forall r$, $a(j) = 80$, for $j \in \{1, 2, 3, 16, 17, 18\}$, and $a(j) = 0$ for all other $j$, $\sigma = 0.4$, $\rho_{\text{trunc}} = 0.2$, $K_i^r = 90 \forall i, r$, $\Gamma = \text{identity}$, $\gamma = 2000$, and $w_r = 30 \forall r$, and $D_{\text{max}}$ is assumed to be very large, so that $l_{r,r'}(t)\zeta = 20$, $\forall r, r'$, $\forall t$.

The environment created by the parameters can be seen in Figure 4-11. The initial

![Figure 4-9: Initial paths](image)
Figure 4-10: Final paths

Figure 4-11: Environment

Figure 4-12: Consensus error

Figure 4-13: Lyapunov-like function in learning phase

Figure 4-14: Lyapunov-like function in path shaping phase

Figure 4-15: Mean waypoint position error

Figure 4-16: Mean integral parameter error
paths for both cases are practically the same and are shown in Figure 4-9. The final paths, however, are very different for each case and are shown in Figure 4-10. The difference in final paths is caused by the difference in the neighbor distance weight $W_n$. For Case 1, $W_n$ is high, providing high attractive forces between neighboring waypoints. This causes the paths to tend to be short because the neighbor waypoints want to be close to each other. Therefore, the system evolves in a way that the environment is covered with low-length paths. On the contrary, in Case 2, there is a low $W_n$, providing low attractive forces for neighboring waypoints. This causes the waypoints to be more distant to each other and focus more on the coverage task, rather than generating short paths. Depending on the application, the weights $W_n$ and $W_s$ can be selected to generate appropriate results. As an example, if collision is an issue, one way to possibly avoid collisions is to have a high $W_n$, so that paths tend to not intersect each other.

Figure 4-12 shows the consensus error for both cases, which converges to zero in accordance to (iii) from Theorem 3. Figures 4-13 and 4-14 show the Lyapunov-like function in the learning phase and path shaping phase, respectively. In both phases, the function is monotonically non-increasing and approaching a limit. Finally, Figures 4-15 and 4-16 show the mean waypoint position error and the mean integral parameter error converging to zero, in accordance to (i) and (ii) from Theorem 3.

4.6.2 Variations of Initial Paths

For all of the controllers that have been shown in previous sections, any initial paths for the robots can be selected. However, depending on the initial paths that are given to the robots, the system may achieve a different informative paths corresponding to a different local minimum to the cost function. In this section, we provide an example to show how the initial paths can affect the local equilibrium at which the system settles to. The example consists of two cases of a system with the same environment, parameters and number of robots, but with different initial paths for the robots. The parameters are: $W_s = 50$, $W_n = 3$, $N = 2$, $a(j) = 25$, for $j \in \{7, 9, 11, 15, 16, 20\}$, and $a(j) = 0$ for all other $j$, $\sigma = 0.18$, $\rho_{\text{trunc}} = 0.15$, $K_i^r = 90 \forall i, r$, $\Gamma = \text{identity}$, $\gamma = 500$. 

77
and \( w_r = 3 \forall r \), and \( D_{\text{max}} \) is assumed to be very large, so that \( l_{r,r'}(t) \zeta = 20, \forall r, r', \forall t \).

The environment created by the parameters can be seen in Figure 4-19. The initial paths are the only difference between the two cases and these are shown in Figure 4-17. Consequently, the resulting final paths are very different for each case and are shown in Figure 4-18. The difference in final paths is caused by a different local minimum in the cost function. From this example, it is clear that the initial paths for the robots can greatly affect the outcome of the final path shapes. Therefore, if some structure of the environment is known, it can be used to design initial paths that are more beneficial to the outcome of the system.

Figure 4-20 shows the consensus error for both cases, which converges to zero in accordance to (iii) from Theorem 3. Figures 4-21 and 4-22 show the Lyapunov-like function monotonically non-increasing in the learning phase and path shaping phase, respectively. For both cases, the Lyapunov-like function approaches a limit. Finally, Figures 4-23 and 4-24 show the mean waypoint position error and the mean integral parameter error converging to zero, in accordance to (i) and (ii) from Theorem 3.

![Figure 4-17: Initial paths](image)
Figure 4-18: Final paths

Figure 4-19: Environment

Figure 4-20: Consensus error

Figure 4-21: Lyapunov-like function in learning phase

Figure 4-22: Lyapunov-like function in path shaping phase

Figure 4-23: Mean waypoint position error

Figure 4-24: Mean integral parameter error
Chapter 5

Persistent Informative Controller for Multiple Robots

In this chapter we extend the persistent informative controller from Chapter 3 to multiple robots and prove that it enables the robots to learn the environment dynamics, in the form of growth rates of the field over the environment, and generates informative paths for multiple robots with finite sensor footprints to follow and sense the growing accumulation function to maintain the height of this function bounded.

5.1 Relation to Persistent Sensing Tasks

We assume each robot is equipped with a sensor with a finite footprint $F_r(p_r) = \{q \in Q : \|q - p_r\| \leq \rho\}$\(^1\). Similar to Section 3.2, the stability criterion for a persistent sensing task executed by multiple robots, when given the speed profile for each robot [35], is

$$
\phi(q) - \sum_{r'=1}^{N} \tau_{r'}(q,t) \frac{c_{r'}(q)}{T_{r'}(t)} = s(q,t) < 0, \quad \forall q \mid \phi(q) > 0,
$$

\(\text{(5.1)}\)

\(^1\)Any footprint shape can be used, and the footprint size does not have to be the same for all robots. For simplicity, we use a circular footprint with same size for all robots.
where the constant scalar $\phi(q)$ (the sensory function) is the rate at which the accumulation function grows at point $q$, the constant scalar $c_r(q)$ is the rate at which the accumulation function shrinks when robot $r$'s sensor is covering point $q$, $T_r(t)$ is the time it takes robot $r$ to complete the path at time $t$, and $\tau_r^*(q,t)$ is the time robot $r$'s sensor covers point $q$ along the path at time $t$. These two last quantities are calculated with the speed profiles. The stability margin of the system is $S(t) = -(\max_q s(q,t))$ and a stable persistent task is one in which $S > 0$. Since the robots do not know the environment, but have estimates of it, each robot $r$ uses the estimated version of (5.1), i.e.

$$\hat{\phi}_r(q,t) - \sum_{r'=1}^N \frac{\tau_r^*(q,t)}{T_{r'}(t)} c_{r'}(q) = \hat{s}_r(q,t) < 0, \quad \forall q \mid \hat{\phi}_r(q,t) > 0. \quad (5.2)$$

Robot $r$’s estimated stability margin at time $t$ is defined as $\hat{S}_r(t) = -(\max_q \hat{s}_r(q,t))$. We assume that each robot knows the speed profile that maximizes $\hat{S}_r(t)$ [35].

### 5.2 Persistent Informative Controller

Let the waypoints have new dynamics of the form

$$\dot{p}_i^r = I_i^r u_i^r, \quad (5.3)$$

where $u_i^r$ is defined in (4.10), and

$$I_i^r = \begin{cases} 0, & \text{if } \frac{\partial \hat{s}_r}{\partial p_i} u_i^r < 0 \text{ and } t - t_u^{r,i} > \tau_{\text{dwell}}, \\ 1, & \text{otherwise}, \end{cases} \quad (5.4)$$

where $t_u^{r,i}$ is the most recent time at which $I_i^r$, switched from zero to one (switched “up”).

**Theorem 4 (Convergence Theorem for Persistent Sensing-Multiple Robots)**

*Under Assumption 1, with waypoint dynamics specified by (5.3), control law specified...*
by (4.10), and adaptive law specified by (4.17), we have

1. \( \lim_{t \to \infty} I_r^T(t)\|\hat{M}_r^T(t)e_r^r(t)+\alpha_r^r(t)\| = 0, \quad \forall r \in \{1, \ldots, N\}, \forall i \in \{1, \ldots, n(r)\}, \)

2. \( \lim_{t \to \infty} \|\tilde{\phi}_{pr}(\tau)\| = 0, \quad \forall r \in \{1, \ldots, N\}, \forall \tau \mid w_r(\tau) > 0, \)

3. \( \lim_{t \to \infty} (\hat{a}_r - \hat{a}_{r'}) = 0, \quad \forall r, r' \in \{1, \ldots, N\}. \)

**Proof 4** We define a Lyapunov-like function based on the robots’ paths and environment estimates, and prove asymptotic stability of the system to a locally optimal equilibrium.

Let \( V_4 \) be the new Lyapunov-like function, and let \( V_4 = V_3 \) from (4.19). Then, following the procedure from Section 4.4, but with \( \tilde{p}_i \) defined by (5.3), we get

\[
V_4 = \sum_{r=1}^{N} \sum_{i=1}^{n(r)} -\frac{1}{\beta_r^T} (\hat{M}_r^T \hat{e}_i^r + \alpha_r^r)^T I_i^T K_i^r (\hat{M}_r^T \hat{e}_i^r + \alpha_r^r) - \gamma \sum_{r=1}^{N} \int_{0}^{t} w_r(\tau)(\tilde{\phi}_{pr}(\tau))^2 d\tau - \sum_{r=1}^{N} \hat{a}_r^T L_{proj} \hat{a}_{pre} - \zeta \sum_{j=1}^{m} \hat{\Omega}_j^T L\hat{\Omega}_j. \tag{5.5}
\]

Using a similar analysis as in Section 4.4, denote the four terms in (5.5) as \(-\xi_1(t), -\xi_2(t), -\xi_3(t)\) and \(-\xi_4(t)\), so that \( \dot{V}_4(t) = -\xi_1(t) - \xi_2(t) - \xi_3(t) - \xi_4(t) \). From previous sections we know that \( \xi_1(t) \to 0, \xi_2(t) \to 0 \) and \( \xi_4(t) \to 0 \), which implies (i), (ii) and (iii), respectively.

### 5.3 Simulation and Results

#### 5.3.1 System Architecture

The system architecture is the same as in Figure 4-1 from Section 4.5.1, but the algorithm for layer 2 changes slightly, and is seen in Algorithm 6.

#### 5.3.2 Numerical Simulation

The informative path controller for persistent sensing by multiple robots was simulated in a MATLAB environment for many test cases. Here we present a case for
Algorithm 6 Persistent Informative Controller for Multiple Robots:
Layer 2 for robot $r$’s $i^{th}$ waypoint

Require: Each waypoint knows $\hat{a}_r$ from Algorithm 4
Require: Each waypoint can calculate its Voronoi partition
Require: Each waypoint knows its neighboring waypoints
Require: Each waypoint has knowledge of $\hat{S}_r$

1: Initially compute the value of $\hat{S}_r$
2: loop
3: Compute the waypoint’s Voronoi partition
4: Compute $C_t^r$ according to (2.12), but integrating over $V_t^r$ from (4.1)
5: Obtain neighbor waypoint locations $p_{r-1}$ and $p_{r+1}$
6: Compute $u_t$ according to (4.10)
7: Compute $I_t^r$ according to (5.4)
8: Update $p_t^r$ according to (5.3)
9: end loop

$N = 10$ robots, $n(r) = 24$ waypoints, $\forall r$. The same fixed-time step numerical solver
is used with a time step of 0.01 seconds and $\tau_{dwell} = 0.009$. The environment parameters
are $\sigma = 0.4$ and $\rho_{trunc} = 0.2$, $a(j) = 60$, for $j \in \{7, 8, 9, 12, 13, 14, 17, 18, 19\}$,
and $a(j) = 0$ otherwise. The environment created with these parameters can be seen
in Figure 5-4c. The parameters $\hat{a}_r$, $\Lambda_r$ and $\lambda_r$, for all $r$ are initialized to zero. The
parameters for the controller are $K_r = 90$, $\forall i,r$, $\Gamma = identity$, $\gamma = 2000$, $W_n = 3$,
$W_s = 100$, $w_r = 10$, $\forall r$ and $\rho = 0.08$. In addition, $D_{max}$ is assumed to be very large,
so that $l_{r,r'}(t)\zeta = 10$, $\forall r, r'$, $\forall t$. The environment is discretized into a $8 \times 8$ grid and
only points in this grid that satisfy $\hat{\phi}_r(q) > 0$ are used as points of interest in (5.2).

The initial paths can be seen in Figure 5-4, where each robot has a “zig-zagging” path across a portion of the environment, and between all robots, most of the environment is initially traversed. Again, we present results in the two separate phases: 1) learning phase, and 2) path shaping phase, as described in Section 2.5.2. In the path shaping phase, (5.3) is used to reshape the path.

Learning Phase

In the learning phase, the adaptation laws cause $\tilde{\phi}_r(q) \to 0$, $\forall q \in Q$, $\forall r$. This
process can be seen in Figure 5-4, where the estimated environment for two robots
is presented on the right side. As can be seen, the estimated environments converge
to the real environment. This means that the union of all robots’ trajectories was rich enough to generate accurate estimates for all of the environment. Figure 5-1 shows that the mean over all robots of $\int_0^t w_r(\tau)(\hat{\theta}_{pr}(\tau))^2 d\tau$ converges to zero, in accordance with (ii) from Theorem 4. Figure 5-3 shows that the consensus error, referring to $\zeta \sum_{r=1}^N \hat{a}_r^T \sum_{r'=1}^N l_{r,r'}(\hat{a}_r - \hat{a}_{r'})$ converges to zero, in accordance with (iii) from Theorem 4, meaning that all robots end up with the same estimate of the environment (which converges to the real environment). Finally, for this learning phase, we see in Figure 5-2 that the Lyapunov-like function $\mathcal{V}_4$ is monotonically non-increasing, which supports our theory.
Figure 5-4: Simulated multi-robot system with persistent informative controller during the learning phase. Left: 1) the paths, shown as colored lines, connect all the corresponding waypoints, shown as black circles; 2) the points of interest are shown as green dots (concentrated in the center of the environment); and 3) the black arrowheads represent the simulated robots, and their sensor footprints are the colored circles around the robots' positions. Right: the environment description for two of the 10 robots, where the translucent environment represents the true environment (common to all robots) and the solid environment represents the individual estimated environments.
Path Shaping Phase

Once the robots travel through their paths once, the controller from Section 5.2 is activated. We can see how the path evolves under this controller in Figures 5-8a to 5-8f. Figure 5-5 shows the quantity $I_i'(t)\|\hat{M}_i'(t)e_i'(t)+\alpha_i'(t)\|$ converging to zero, $\forall i,r$ in accordance to (i) from Theorem 4.

Figure 5-6 shows the Lyapunov-like function $V_4$ monotonically non-increasing and reaching a limit during the path morphing phase. The initial value of this function in the path morphing phase is the final value of the function in the learning phase. Finally, Figure 5-7 shows the persistent sensing task’s stability margin increasing through time, as expected. The chattering in the stability margin is due to the discretization of the system. Reducing the length of time steps will reduce this chattering.
Figure 5-8: Simulated multi-robot system with persistent informative controller during the path shaping phase. The paths, shown as the colored lines, connect all the corresponding waypoints, shown as black circles. The points of interest are shown as green dots (concentrated in the center of the environment). The black arrowheads represent the robots, and their sensor footprints are represented by the colored circles around the robots' positions.
5.4 Discussion

Although the controller from this chapter and the one in Chapter 4 are very similar, the system’s behavior using Algorithm 6 can be very different from the behavior using Algorithm 5. The reason is that the switching variable in (5.4) can prevent the paths from shaping the same way that they would by using the informative path controller from Algorithm 5. Each robot will use the speed profile it computed to stabilize the system jointly with the other robots. If the robots’ estimates of the environment are different, then the speed profiles that the robots use might not be complementary towards stabilizing the persistent sensing task. This is why, in our simulations and experiments, we have a learning phase in which the robots use the consensus term in their adaptive laws to converge on their estimates. By doing so, when the path shaping phase starts, the robots work together with complementary speed profiles to stabilize the persistent sensing task in all of the environment.

In this chapter we have discussed how multiple robots shape their paths into informative paths that are useful for persistent sensing. In general these informative paths may intersect, or the paths may intersect as they shape into informative paths. In either case, robots following such paths could collide. Therefore, for a practical system, a collision avoidance procedure is needed. In the next chapter we discuss a collision avoidance algorithm that is designed to prevent robots from colliding and entering deadlock while attempting to maintain the stability of a persistent sensing task.
Chapter 6

Collision Avoidance for Persistent Sensing Tasks

Figure 6-1 shows four robots performing a persistent sensing task with a given set of paths. This set of paths generates multiple potential collision locations. Therefore, when implementing the system, if no collision avoidance procedure is used, the robots would collide frequently. However, collision avoidance must be done in a smart way in order to maintain some of the stability guarantees from previous work in persistent sensing. As a reminder, previous work developed a way to calculate the speed of robots traveling along known static paths in order to maintain the height of the accumulation function (or growing field) bounded. The accumulation function, referred to as $Z(q, t)$ at time $t$ for point $q$, evolves according to

$$
\dot{Z}(q, t) = \begin{cases} 
\phi(q) - \sum_{r \in N_q(t)} c_r(q), & \text{if } Z(q, t) > 0, \\
(\phi(q) - \sum_{r \in N_q(t)} c_r(q))^+, & \text{if } Z(q, t) = 0,
\end{cases}
$$

where $N_q(t)$ is the set of robots whose sensor footprints are over the point $q$ at time $t$, i.e. $N_q(t) = \{r \mid q \in F_r(p_r(t))\}$. Therefore, if collision is not avoided in a smart way,
Figure 6-1: Four robots (black arrowheads) performing a persistent task where their paths produce many possible collision zones. Each robot has a unique color to represent its path and its footprint. The paths are the four large circles, and the footprints are the small circles which are centered around each respective robot. The yellow circles represent the robot bodies which should not collide. In persistent sensing tasks, the robots follow speed profiles which seek to keep the accumulation function of each point of interest bounded. In this figure, the points of interest are colored in green and the value of the accumulation function at each point is proportional to the size of that point. If no collision avoidance procedure is implemented, these robots could collide frequently.

The accumulation function\(^1\) can grow unbounded. In this chapter\(^2\), we present and prove a practical collision avoidance procedure for a multi-robot persistent sensing task that seeks to minimize its effects on the stability of the system, i.e. maintain the height of the accumulation function bounded at the points of interest.

### 6.1 Problem Setup

Each robot \(r\) is constrained to move along its path, from now on referred to as \(\psi_r\), which is formed by the linear interpolation of the waypoints. For each robot \(r\), we introduce the parametrization variable \(\theta_r\), where \(0 \leq \theta_r \leq 2\pi\) and \(\theta_r\) is assumed

---

\(^1\)Note that from (6.1), the accumulation function is non-negative.

\(^2\)The majority of this chapter was published in [38].
to be the arc-length parametrization of the path. Therefore, we can formalize the
definition of the path as $\psi_r : [0, 2\pi] \to \mathbb{R}^2$, where $\psi_r(0) = \psi_r(2\pi)$ since it is a closed
path. Consequently, robot $r$’s position at time $t$ can be described by $\theta_r(t)$, i.e. the
position along the curve $\psi_r$. We can also describe the robots’ sensor footprints with
this parametrization as $F_r(\theta_r(t)) = \{q \in Q : \|q - \psi_r(\theta_r)\| \leq \rho\}$.

6.2 Collision Avoidance

For each robot $r$ we define a safety radius $\rho_{\text{safe}} > 0$, and the corresponding safety
disk

$$
B(\psi_r(\theta_r), \rho_{\text{safe}}) = \{q \in Q : \|q - \psi_r(\theta_r)\| \leq \rho_{\text{safe}}\}. \quad (6.2)
$$

We say that a collision occurs between robots $r$ and $r'$ at location $(\theta_r, \theta_{r'})$ if

$$
B(\psi_r(\theta_r), \rho_{\text{safe}}) \cap B(\psi_{r'}(\theta_{r'}), \rho_{\text{safe}}') \neq \emptyset.
$$

To avoid robot collisions, we must first know where collisions can occur. There are a
number of ways to do this. For example, we could search all $(\theta_r, \theta_{r'})$ pairs between
any two robots $r$ and $r'$ and obtain the set of collision configurations between them.
The set of collision configurations between robot $r$ and $r'$ is formally defined as

$$
P_{r, r'} = \{(\theta_r, \theta_{r'}) \in [0, 2\pi]^2 : B(\psi_r(\theta_r), \rho_{\text{safe}}) \cap B(\psi_{r'}(\theta_{r'}), \rho_{\text{safe}}') \neq \emptyset\}. \quad (6.3)
$$

An example of $P_{r, r'}$ can be seen in Figure 6-2. This figure shows the phase por-
trait of $\theta_r$ vs. $\theta_{r'}$ for a given $\psi_r$ and $\psi_{r'}$. The flow lines depict speed profiles that
stabilize the system for a given set of points of interest. The set $P_{r, r'}$ is given by the
black-colored regions. There are various flow lines that lead $(\theta_r, \theta_{r'})$ to enter $P_{r, r'}$,
resulting in collisions. One way to avoid collision is to not allow $(\theta_r, \theta_{r'})$ to enter $P_{r, r'}$
by forcing $(\theta_r, \theta_{r'})$ to move along the edge of $P_{r, r'}$. However, there are a few prob-
lems with this approach. First, since $P_{r, r'}$ can have an arbitrary geometric shape, a

---

3 For simplicity of presentation we use a safety disk, but our collision procedure works for any
safety set containing the robot’s current position.
Phase Portrait: robots 1 and 2. The axis are $\theta_1$ and $\theta_2$, which are the parametrized positions of robots 1 and 2. In this figure, the black-colored sets are the set of collision configurations $(\theta_1, \theta_2)$ for robots 1 and 2. The blue arrows in this plot correspond to the phase portrait, i.e. flow lines of the state $(\theta_1, \theta_2)$ through time according to the persistent task speed profiles. If no collision avoidance is used, the robots will eventually collide due to the flow lines leading $(\theta_1, \theta_2)$ into the black sets.

Figure 6-2: Phase portrait for robots 1 and 2. The axis are $\theta_1$ and $\theta_2$, which are the parametrized positions of robots 1 and 2. In this figure, the black-colored sets are the set of collision configurations $(\theta_1, \theta_2)$ for robots 1 and 2. The blue arrows in this plot correspond to the phase portrait, i.e. flow lines of the state $(\theta_1, \theta_2)$ through time according to the persistent task speed profiles. If no collision avoidance is used, the robots will eventually collide due to the flow lines leading $(\theta_1, \theta_2)$ into the black sets.

collision avoidance of this type will, in general, require robots to move backwards. Moving backwards affects the phase portrait by including flow lines at angles less than zero or greater than $\pi/2$. When considering multiple robots, such backward motion will require additional collision avoidance procedures. Second, we are interested in solutions that can be implemented in a distributed manner, and thus we would like the robots to utilize pair-wise decisions in order to avoid collisions. However, care must be taken when considering such pair-wise decision because separate decisions could contradict each other. This can result in a deadlock situation in which a group of robots are all blocking each other.

Based on the above discussion, we propose a collision avoidance method that relies on stopping the robots. When a robot stops in order to avoid collision, the collision avoidance procedure pauses the speed controller, and un-pauses it to resume robot motion. When the speed controller is un-paused, it is as if the system is re-started with a new set of initial conditions. This method for avoiding collision is tractable in a persistent sensing task because the speed profile prescribed by [35] is proven to stabilize the field for any set of initial conditions. Therefore, any increase in the accumulation functions of the field while a robot is stopped will eventually
be consumed, maintaining the system stable. However, depending on the robots’ trajectories, if collision avoidance is needed very frequently, then the stabilizing effect of the speed profile may not be “strong” enough to overcome the frequent stops.

We describe an algorithm for stopping robots that ensures no deadlock amongst stopped robots while enabling the persistent operation of the system without collision. The intuition is to compute, for each robot, the regions in space where the robot might collide with any other robot. Since the path for each robot is known, we can search for all possible collision locations along the paths. The set of all possible collision locations for robot $r$ is defined as

$$P_r = \{\theta_r \in [0, 2\pi] : B(\psi_r(\theta_r), \rho_{r\text{safe}}) \cap B(\psi_{r'}(\theta_{r'}), \rho_{r'\text{safe}}) \neq \emptyset, \forall r' \neq r\}. \quad (6.4)$$

For example, in Figure 6-2, $P_r$ would be set containing the projection of $P_{r,r'}$ onto the $\theta_r$ axis, and $P_{r'}$ would be the set containing the projection onto the $\theta_{r'}$ axis. We can decompose $P_r$ into a collection of $\nu_r$ connected sets $C^k_r$, which are pairwise disjoint. Thus, we have $P_r = \bigcup^\nu_{k=1} C^k_r$, and $C^k_r \cap C'^k_r = \emptyset$ for all $k, k' \in \{1, \ldots, \nu_r\}$. As an example, in Figure 6-2, both $P_r$ and $P_{r'}$ consist of two disjoint sets.

We would now like to determine the following: if robot $r$ enters the set $C^k_r$, which sets $C'^{k'}_{r'}$ must robot $r'$ avoid? We can relate individual collision zones by constructing an undirected graph where each $C^k_r$ is a node, $\forall r, k$. We define an edge between two nodes $C^k_r$ and $C'^{k'}_{r'}$ if $B(\psi_r(\theta_r), \rho_{r\text{safe}}) \cap B(\psi_{r'}(\theta_{r'}), \rho_{r'\text{safe}}) \neq \emptyset$, for some $\theta_r \in C^k_r$ and some $\theta_{r'} \in C'^{k'}_{r'}$. We will refer to this graph as the collision graph.

**Definition 3 (Collision Zone)** Given the collision graph, a collision zone CZ$^s$ is defined for each connected component in the graph, where $s$ will range from 1 to the total number of connected components in the graph. CZ$^s$ is defined as a tuple $\text{CZ}^s = (\text{CZ}^s_1, \text{CZ}^s_2, \ldots, \text{CZ}^s_N)$, where $\text{CZ}^s_r$ is the union of nodes $C^k_r$, $\forall k$, present in the connected component corresponding to $\text{CZ}^s$.

Note that $\text{CZ}^s_r$ is disjoint from $\text{CZ}^s_{r'}$ for all $r$, where $s \neq s'$, because if they were not, then $\exists \theta_r$ for some $r$ such that $\theta_r \in C^k_r$ and $\theta_{r'} \in C'^{k'}_{r'}$, where $C^k_r \in \text{CZ}^s_r$ and
However, by definition $C_r^k$ and $C_r'^k$ are disjoint. Therefore, $CZ_s^r$ and $CZ_s'^r$ are also disjoint. Example 1 illustrates these mathematical constructions.

Example 1 Figure 6-3 shows three paths that intersect each other at several places. After obtaining all the disjoint collision sets $C_r^k$ for all robots, we construct the collision graph in Figure 6-4, which shows five connected components, i.e. five collision zones $CZ_s^r$, $s \in \{1, 2, 3, 4, 5\}$, where:

1. $CZ_1^1 = C_1^1$, $CZ_2^1 = C_2^1$, $CZ_3^1 = (C_3^1 \cup C_3^2)$
2. $CZ_2^2 = C_1^2$, $CZ_2^2 = \emptyset$, $CZ_3^2 = C_3^3$
3. $CZ_3^3 = C_1^3$, $CZ_3^3 = \emptyset$, $CZ_3^3 = C_3^4$
4. $CZ_4^4 = \emptyset$, $CZ_2^4 = C_2^2$, $CZ_3^4 = C_3^5$
5. $CZ_5^5 = \emptyset$, $CZ_2^5 = C_2^3$, $CZ_3^5 = C_3^6$

The following shows the importance of collision zones.

Theorem 5 If any two robots $r$ and $r'$ collide, then $\theta_r \in CZ_s^r$ and $\theta_{r'} \in CZ_s'^r$, for some $s$.

Proof 5 Suppose there is a collision between robots $r$ and $r'$, when robot $r$ is at $\theta_r$ and robot $r'$ is at $\theta_{r'}$. This implies that $B(\psi_r(\theta_r), \rho_{r_{safe}}) \cap B(\psi_{r'}(\theta_{r'}), \rho_{r'_{safe}}) \neq \emptyset$. By

Figure 6-3: Paths for three robots used in Example 1. The final five collision zones $CZ_s^r$ (obtained from Figure 6-4) are plotted in segments of black, while the rest of the trajectories are plotted in different light colors.
Figure 6-4: Collision graph for Example 1, used to construct the collision zones CZ'. In this graph, there are 12 nodes corresponding to the disjoint connected sets in $P_1$, $P_2$, $P_3$, for robots 1, 2 and 3, respectively. The result from this graph is that there are five disjoint collision zones, i.e. $CZ^s$, $s \in \{1, 2, 3, 4, 5\}$. These five collision zones are mapped to their respective paths in Figure 6-3.

By Theorem 5, no collisions are possible in $CZ^s$ if there exists at most one robot $r$ such that $\theta_r \in CZ_r^s$. Therefore, the collision avoidance framework will allow at most one robot to travel through $CZ^s$ at any moment in time, for each $s$. Let $flag_s$ be a flag which is raised if any robot $r$ is currently inside $CZ_r^s$. Then the collision avoidance framework is given in Algorithm 7. A requirement for Algorithm 7 to work is that if $flag_s$ is raised by some robot $r$, then it is lowered only by robot $r$ once it exits $CZ_r^s$.

Note that the converse of Theorem 5 is not true. That is, if $\theta_r \in CZ_r^s$ and $\theta_{r'} \in CZ_{r'}^s$, for some $s$, this does not mean that the robots have collided. This is because, in general, a connected set of $(\theta_r, \theta_{r'})$ that results in collision is not equal to the cartesian product of its projections on the $\theta_r$ and $\theta_{r'}$ axis, although it is a subset. This means that this collision avoidance algorithm is conservative. However,

**Algorithm 7** Collision Avoidance Framework, for robot $r$

**Require:** $\theta_r$ is at the entering edge of $CZ_r^s$.

1: if $flag_s$ is raised then
2: Stop trajectory until $flag_s$ is lowered.
3: else
4: Robot $r$ can enter $CZ_r^s$ and raise $flag_s$.
5: When robot $r$ exits $CZ_r^s$, then $flag_s$ is lowered.
6: end if
the trade-off is a solution without backward motion and conflicting decisions where a robot stops while blocking the path of another, potentially causing a deadlock.

**Remark 5** The collision avoidance framework in Algorithm 7 requires that $\exists \theta_r \notin P_r$, $\forall r$. In other words, that for each robot, not all of its path is contained in some collision zone. Otherwise, the robot never exits a collision zone and the collision avoidance framework does not work.

### 6.3 Deadlock Avoidance

Much of the previous work in deadlock avoidance is based on re-planning of the robot trajectories, e.g. [2, 3, 15, 29], or schedule coordination for robots, e.g. [23]. In our problem formulation, however, the robots are constrained to their paths and speed profiles, so these approaches do not apply. Instead, our approach is similar to [31], where graphs are used to detect and avoid collisions in critical sections, and to [5], where permission is given to one robot to move along a zone that can cause deadlock. In order to avoid deadlock, we define the notion of a deadlock graph.

**Definition 4 (Deadlock Graph)** A deadlock graph is a directed graph, where an edge from node $r$ to node $r'$ encodes in robot $r$ is stopped waiting for robot $r'$ to exit a collision zone $CZ_r^s$, for some $s$.

It is assumed that all robots have knowledge of the deadlock graph. In our application, deadlocks can be avoided by noting that they can only occur when a cycle is created on a deadlock graph. If a cycle were to exist, then deadlock is avoided by erasing one of the edges in the cycle. This corresponds to one of the robots resuming motion and breaking the deadlock.

To help us decide which robots should stop when avoiding collision, we use what are stopping policies.

**Definition 5 (Stopping Policy)** A stopping policy, executed by robot $r$ when about to enter a collision zone is any algorithm that returns $N$ options ranked from best
Algorithm 8 Collision and Deadlock Avoidance Algorithm, for robot $r$

Require: $\theta_r$ is at the entering edge of $CZ^s_r$, for some $s$

1: if flag$_s$ is raised then
2: Stop trajectory.
3: else
4: Execute stopping policy.
5: if The decision is to continue trajectory then
6: flag$_s$ is raised
7: Any outgoing edge from robot $r$’s node in the deadlock graph is deleted.
8: else
9: The decision edge is drawn in deadlock graph.
10: end if
11: end if
12: When robot $r$ exits $CZ^s_r$, then flag$_s$ is lowered.

Theorem 6 Assuming all robots follow Algorithm 8, consider a robot $r$ about to enter
a collision zone $CZ_r^s$, for some $s$. Then, there exists a decision by robot $r$ which does not cause a deadlock.

**Proof 6** Suppose robot $r$ is about to enter $CZ_r^s$. If flag$_s$ is not raised, then it will test its $N$ options obtained from a stopping policy, in the deadlock graph. If the best ranked option does not cause a cycle in the deadlock graph, then it is allowed and the robot chooses that option. If the option causes a cycle, then it is not allowed, and the robot tests the next best-ranked option, and so on. By default, the robot will always have the option of continuing its trajectory, which does not create an edge in the deadlock graph, which in turn does not create a cycle.

If flag$_s$ is raised, then robot $r$ is forced to stop because some robot $r'$ is already in $CZ_{r'}^s$. Conceptually, robot $r$ would draw an edge to robot $r'$ in the deadlock graph. However, this does not create a cycle because robot $r'$ is moving inside the collision zone, and will never stop inside it because $CZ_{r'}^s$ is disjoint from $CZ_r^s$, $\forall s' \neq s$.

### 6.4 Stopping Policies

We are interested in stopping policies that minimize the effect of stopping to avoid collision on the stability of the persistent sensing task, i.e. we would like the stopping policy to result in maximizing the estimated stability margin of the system, which is given in (5.2). However, (5.2) assumes the system is periodic for a given path (a proof for this can be found in [35]). That is, it assumes that each robot takes a fixed amount of time to complete a cycle of its path. However, each time a robot stops, it breaks periodicity. In Sections 6.4.1, 6.4.2 and 6.4.3, we define three stopping policies, two of which use very similar versions of Equation (5.2) to generate their $N$ ranked options.

For the following policies, executed by robot $r$, option$_r$ corresponds to robot $r$ continuing its trajectory, and option$_{r'}$ corresponds to robot $r$ stopping for some other robot $r'$. These policies assume no other collision procedures are taking place while the current collision is being resolved. Although this is not always true, it provides a
quick and inexpensive approximation, compared to the price of obtaining the exact information.

**Remark 6 (Limitation of locally optimizing)** All of the following stopping policies look ahead in time, up to the point where the collision is avoided, and they optimize the system, according to their metric for that look-ahead time. Therefore, these policies are limited to optimizing locally in time, and there could be the case where a decision is optimal for the policy, but suboptimal over a longer time horizon.

### 6.4.1 Minimum Time Policy

This policy ranks the options based on how much time a robot spends stopped to avoid the collision. For robot \( r \), the policy takes into account all of the robots’ current positions and trajectories, and calculates the amount of time robot \( r \) would have to stop while the other robots enter and exit the collision zone, and the amount of time the other robots would have to stop while robot \( r \) exits the collision zone. The options are ranked in ascending order of stopping time, i.e. the best-ranked option is the one that results in the least amount of time a robot is stopped. Algorithm 9 presents this policy, which is called the *Min-Time* policy, where:

- \( T'_{\text{enter}} \) is the time robot \( r \) would take to get from where it is right now until it enters \( CZ^*_r \).

- \( T'_{\text{exit}} \) is the time robot \( r \) would take to get from where it is right now until it exits \( CZ^*_r \).

### 6.4.2 All Time Policy

This stopping policy approximates (5.2) with the *empirical estimated stability margin* of the system up until the current time \( t \). This is done by constructing the same expression in (5.2), where \( T_r \) becomes the current time \( t \), and \( \tau^r_c(q) \) becomes the total coverage time on point \( q \) by robot \( r \). The policy estimates the empirical stability margin at the time when the collision is avoided, and ranks the options in descending
Algorithm 9 Min-Time Policy, for robot $r$

1: Initialize $\text{option}_r = \infty$
2: for each robot $r' \neq r$ do
3:   if $T_{\text{exit}}^r > T_{\text{enter}}^{r'}$ then $\triangleright$ If stopping is necessary
4:      $\text{option}_r = \min(T_{\text{exit}}^r - T_{\text{enter}}^{r'}, \text{option}_r)$
5:      $\text{option}_{r'} = T_{\text{exit}}^{r'}$
6:   else
7:      $\text{option}_{r'} = \infty$
8: end if
9: end for
10: Rank the options in ascending order of their values.

order of estimated empirical stability margin, i.e. the best-ranked option is the one that results in the highest empirical estimated stability margin. Algorithm 10 presents this policy, which is called the All-Time policy, where:

- $TC_{\text{enter}}^r(q)$ is the time robot $r$ covers point $q$ while it moves from its current position until it enters $CZ_r^*$
- $TC_{\text{exit}}^r(q)$ is the time robot $r$ covers point $q$ while it moves from its current position until it exits $CZ_r^*$
- The average consumption on point $q$ from all robots, except robots $r$ and $r'$, up to time $t$ is obtained by

$$\sum_{r'' \neq r, r'} \frac{\tau_{r''}(q)}{t} c_{r''}(q) \quad (6.5)$$

- The average consumption on point $q$ from robots $r$ and $r'$ at the time the collision is avoided, assuming robot $r$ continues moving is obtained by

$$\frac{(\tau_c^r(q) + TC_{\text{exit}}^r)c_r(q) + (\tau_c^{r'}(q) + TC_{\text{enter}}^{r'} + I_{\text{cover}}^{r'}(q)(T_{\text{exit}} - T_{\text{enter}}^{r'}))c_{r'}(q)}{t + T_{\text{exit}}^r} \quad (6.6)$$

where $I_{\text{cover}}^{r'}(q) = 1$ if robot $r'$'s footprint covers $q$ while it is stopped, and zero otherwise.

- The average consumption on point $q$ from robots $r$ and $r'$ at the time the
Algorithm 10 All-Time Policy, for robot $r$

1: Initialize: $\text{option}_r = -\infty$
2: for each robot $r' \neq \text{robot } r$ do
3:  $V_r(q) = -\phi(q), \forall q$
4:  $V_{r'}(q) = -\phi(q), \forall q$
5:  if $T^r_{\text{exit}} > T^r_{\text{enter}}$ then $\triangleright$ If stopping is necessary
6:      for each $q$ do $\triangleright$ case: robot $r$ continues moving
7:      Add (6.5) to $V_r(q)$.
8:      Add (6.6) to $V_{r'}(q)$.
9:      end for
10:     for each $q$ do $\triangleright$ case: robot $r'$ continues moving
11:        Add (6.5) to $V_{r'}(q)$.
12:        Add (6.7) to $V_{r'}(q)$.
13:     end for
14:     $\text{option}_r = \max(\min_q(V_r(q)), \text{option}_r)$.
15:     $\text{option}_{r'} = \min_q(V_{r'}(q))$.
16: else
17:     $\text{option}_{r'} = -\infty$.
18: end if
19: end for
20: Rank the options in descending order of their values.

Collision is avoided, assuming robot $r'$ continues moving is obtained by

$$
\frac{(\tau^r_c(q) + I^r_{\text{cover}}(q)T^r_{\text{exit}})c_r(q) + (\tau^{r'}_c(q) + T^{r'}C^r_{\text{exit}})c_{r'}(q)}{t + T^r_{\text{exit}}}, \tag{6.7}
$$

where $I^r_{\text{cover}}(q) = 1$ if robot $r$'s footprint covers $q$ while it's stopped, and zero otherwise.

6.4.3 Time Window Policy

This policy is similar to the All-Time policy, but instead of considering all past information, it only considers information in a time window of constant length $T_w$. This is done by exchanging $t$ for $T_w$ in Algorithm 10. Also, $\tau^r_c(q)$ becomes the coverage time of point $q$ within the time window. We will refer to this stopping policy as the Time-Window policy.
6.5 Performance Bound

Equation (5.1) is not useful for persistent sensing while avoiding collisions since this problem is not periodic. However, it can be used to prove performance bounds on the system. From (5.1), we know that the stability margin for point of interest \( q \) with the paths at time \( t \) is

\[
s(q, t) = \phi(q) - \sum_{r=1}^{N} \frac{\tau^r_c(q, t)}{T_r(t)} c_r(q)
\]  

(6.8)

and the stability constraint is \( s(q, t) < 0, \forall q \). Remember that this equation does not consider robots stopping to avoid collision.

**Theorem 7** Consider a persistent task, and a set of speed profiles that result in a stability margin of \( s(q, t) \) for each point of interest for the paths at time \( t \). Then there exists a known \( \kappa(t) > 0 \) such that if \( s(q, t) < \kappa(t)\phi(q) \), \( \forall q \), then the persistent sensing task is stable at time \( t \) while avoiding collision using Algorithm 8. The value \( \kappa(t) \) is a function of the paths and safety disks of all robots.

**Proof 7** In the worst case, each robot will have to stop at every collision zone on each cycle of its path. In this worst case, each robot takes the same amount of time to complete each cycle, and thus we can use Equation (6.8). Then the stability constraint for all points \( q \) becomes

\[
\sum_{r=1}^{N} \frac{\tau^r_{\text{stop}}(q, t)}{h_r(t)T_r(t)} c_r(q) \geq \sum_{r=1}^{N} \frac{\tau^r_c(q, t)}{h_r(t)T_r(t)} c_r(q) > \phi(q),
\]

where \( \tau^r_{\text{stop}} \) is the total amount of time robot \( r \) covers point \( q \) in the worst case (including the time it is stopped), \( h_r(t) = (T_r(t) + T^s_r(t))/T_r(t) \geq 1 \), and \( T^s_r(t) \) is the total maximum amount of time robot \( r \) is stopped in one cycle along its path at time \( t \) in the worst case. Let \( h(t) = \max_r h_r(t) \). Then,

\[
\sum_{r=1}^{N} \frac{\tau^r_c(q, t)}{h_r(t)T_r(t)} c_r(q) \geq \frac{1}{h(t)} \sum_{r=1}^{N} \frac{\tau^r_c(q, t)}{T_r(t)} c_r(q).
\]
Therefore, the system is stable in the worst case at time $t$ if

$$\frac{1}{h(t)} \sum_{r=1}^{N} \frac{\tau^r_{c}(q,t)}{T_r(t)} c_r(q) > \phi(q).$$  \hspace{1cm} (6.9)$$

Substituting (6.8) into (6.9), we get

$$s(q,t) < (1 - h(t)) \phi(q).$$ \hspace{1cm} (6.10)

Letting $\kappa(t) = (1 - h(t))$ we get the desired result.

If a system satisfies (6.10) $\forall q$, then it is guaranteed to be stable for the path at time $t$, no matter how many collision avoidance steps are needed. Consequently, if (6.10) $\forall q$, $\forall t$, then the persistent sensing task will be stable $\forall t$ and will be free of collisions.

6.6 Simulation, Results and Implementation

6.6.1 Simulation Results

We extensively simulated the collision and deadlock avoidance strategies, as well as the three stopping policies in Section 6.4, for pre-defined static paths, using the stabilizing speed profiles from [35]. The Time-Window policy was implemented with two different time windows: $T_w = \max_r(T_r)$ and $T_w = 3 \max_r(T_r)$. We refer to the former as $Time-Window_1$ and the later as $Time-Window_3$. We also implemented a Greedy stopping policy, which allows the first robot to enter a collision zone, and queues subsequently arriving robots.

We generated six sets of test trajectories shown in Figure 6-5, for systems ranging from two robots to four robots. We simulated each trajectory set 100 times. Each one of these simulations is called a test case, and contained 10 randomly located points with random production rates, and a speed profiles obtained from [35] that stabilized the system. Each test case was simulated five times, one for each stopping policy.
Figure 6-5: Six different sets of trajectories used to obtain results on the performance of the tested stopping policies. These trajectories range from using only two robots to using four robots. Each robot has a different colored trajectory.

In each simulation instance, the robots had a safety disk of $p_{\text{safe}}$ equal to 1.25 times the radius of the robot, and they were initialized in collision-free starting locations. Each simulation instance ran for 10,000 iterations and, after finishing, the empirical stability margin was recorded. All test cases were collision-free and deadlock-free.

Besides testing the correctness of our collision and deadlock avoidance strategies, we were interested in knowing which stopping policy generated the best results, i.e. maintain the system "more stable". The aggregated results from all the simulations can be seen in Figure 6-6, which shows the ranking of the policies versus the number of instances that the policy achieved a ranking. A first place ranking corresponds to the policy producing the largest empirical stability margin at the end of the simulation instance. The simulation data shows that the most effective policy is the All-Time policy, followed by the Min-Time policy. Figure 6-7 shows the trajectory set number (using the same reference as in Figure 6-5) versus the number of instances that the policy outperformed its counterpart for the All-Time and Min-Time policies. The data shows that the overall best performance is achieved by the All-Time policy, but there is one trajectory where the Min-Time policy outperforms it. This suggests that
Stopping Policy Ranking

Figure 6-6: Results from stopping policy simulations. The horizontal axis corresponds to the ranking of the policies in the simulation instances, from first place (corresponding to the best policy) to fifth place (corresponding to the policy with worst results). The best policy refers to the policy that generated the largest empirical stability margin. The vertical axis corresponds to the number of instances that the policy achieved a particular ranking.

the geometry of the trajectories may affect the performance of the stopping policies.

In summary, six trajectory sets were simulated for five different stopping policies and for 100 different sets of points of interest. In total, 3,000 instances of the system were simulated, and 100% of the tested instances were free of collision and deadlock.

6.6.2 Distributed Implementation

We implemented a persistent sensing task with collision avoidance on a multi-robot system with pre-defined static paths, consisting of two iRobot Create robots. Algorithm 8 used safety disks with $\rho_{safe}$ equal to 1.1 times the radius of the robot, and it used the All-Time stopping policy. Figure 6-8 shows three snapshots of the evolution of the system in the implementation. This implementation was executed in a distributed way, in the sense that each robot only knew information about itself, and communicated with the other robot when entering a collision zone in order to decide whether to continue its trajectory or stop to avoid collision. The robots tracked their
paths with their speed profiles using a controller based on dynamic feedback linearization [24]. More than 20 experiments were executed, and all were collision-free and deadlock-free.

We ran the same experiment, i.e. same points of interest and same speed controller, seen in Figure 6-8 a total of 20 times, each with a different starting position for the robots. We ran each experiment for 10,000 iterations and recorded the ratio between number of stops and number cycles for each robot. The data revealed that for this experiment setup, using the All-Time algorithm, robot #1, following the ellipsoid path, stopped on average four times for every 10 cycles of its path, while the robot #2 stopped on average five times for every 1,000 cycles of its path. The results intuitively makes sense since most points of interest are closer to robot #2’s path.

When the same type of experiment was run, but with a new set of points and a new speed controller, these ratios changed drastically. Robot #1 stopped on average nine times for every 1,000 cycles, and robot #2 stopped on average two times for every 10 cycles. Therefore, the amount of stops made by a robot to avoid collision is dependent on the paths of the robots, where the points of interest are located relative to the paths, and the stopping policy used to avoid collision.

Figure 6-7: Head-to-head results from the All-Time and Min-Time policies. The horizontal axis corresponds to the trajectory set number, and the vertical axis corresponds to the number of instances that the policy generated the better performance between the two.
Figure 6-8: Snapshots at different times of a distributed implementation for the persistent monitoring task with collision avoidance for two ground robots. The points of interest are represented as green-filled circles, whose size is proportional to the value of the accumulation function for that point of interest. Each robot’s footprint is represented by a concentric circle around the robot’s location, and it is the same color as the path that robot is following.
6.7 Discussion

In this chapter we have presented a collision and deadlock avoidance algorithm that ensures that robots will not collide by not allowing more than one robot in any collision zone at any point in time. This statement was proven to be true in the case where the paths do not change. However, when applying it to our persistent informative controllers, there is a need for some additional decisions in order to ensure that no collisions are possible. For example, one scenario requiring additional action would be when two collision zones merge as a result of path reshaping. When this happens, if there is a robot within each collision zone, then when the collision zones merge there would be two robots within the resulting collision zone and intelligent action must be taken to avoid collisions. In such a case, another collision avoidance procedure, such as using velocity obstacles [37] or artificial potential fields [17] would be useful. Another possible collision avoidance procedure is such cases would be to quickly, but smartly, back up (or speed up) a robot until it is out of the collision zone. One possible way of avoiding such issues is to have the robots perform the learning phase where the paths do not change and collision is avoided properly by our algorithms, and when the path shaping phase starts, the robots can remain idle and let the paths shape themselves until they reach their final configurations. Once the final informative paths are obtained, then the robots can resume their motion avoiding collision with our algorithms.
Chapter 7

Hardware Implementation of Persistent Informative Controller for Multiple Robots with Collision Avoidance

7.1 System Architecture

We combined the persistent informative controller for multiple robots from Chapter 5 with the collision and deadlock avoidance algorithm from Chapter 6 and implemented it in a system with two quadrotors. The system architecture for the hardware implementation is shown in Figure 7-1, where the algorithms developed in this thesis are highlighted in orange boxes. This implementation is a centralized one, meaning that the adaptation, persistent informative controller, and robot movement and collision avoidance nodes perform the calculations for both robots. However, the system can be implemented in a distributed way if the robots can communicate their paths with each other. By distributed, we mean that each robot can perform its own adaptation law and robot movement calculations, and each waypoint can perform its own persistent informative controller calculations. Limited communication is needed be-
Figure 7-1: Architecture for the multi-robot hardware implementation. The orange-filled boxes represent the algorithms that were generated in this thesis, the blue-filled boxes represent the hardware used in the system, and the pink-colored box represents the motion capture infrastructure used in the implementation. Data colored in red represents the messages being sent between nodes. The direction of the arrow represents the direction of data transmission between nodes. The dashed lines represent the implicit communication between the quadrotors and the motion capture system.

tween these distributed processes. The system in Figure 7-1 was implemented using ROS (robot operating system) for message passing and MATLAB for execution of main algorithms. A Vicon motion capture system was used to obtain localization information from the quadrotors.

7.2 Results

We executed this final implemented system more than 10 times. Here we present a case for \( N = 2 \) robots, \( n(r) = 44 \) waypoints, \( \forall r \). A fixed-time step numerical solver is used with a time step of 0.01 seconds and \( t_{\text{dwell}} = 0.009 \). The environment parameters are \( \sigma = 0.4 \) and \( \rho_{\text{trunc}} = 0.2 \), \( a(j) = 60 \), for \( j \in \{3, 4, 5, 10, 15, 20, 23, 24, 25\} \), and \( a(j) = 0 \) otherwise. The environment created with these parameters can be seen in Figure 7-2. The parameters \( a_r, \Lambda_r \) and \( \lambda_r \), for all \( r \) are initialized to zero. The parameters for the controller are \( K_r^f = 90, \forall i, r, \Gamma = \text{identity}, \gamma = 3000, W_n = 6, W_s = 150, w_r = 3, \forall r \) and \( \rho = 0.12 \). In addition, \( D_{\text{max}} \) is assumed to be very large,
so that \( l_{r,r'}(t) \zeta = 20, \forall r, r', \forall t \). The environment is discretized into a 10 \( \times \) 10 grid and only points in this grid with \( \hat{\phi}_r(q) > 0 \) are used as points of interest in (5.2).

In this hardware implementation, we included the growing behavior of the accumulation function over the environment, following the description from (6.1). As an implementation detail, although \( \rho = 0.12 \) for purposes of the persistent informative controller, a value of \( \rho = 0.126 \) (5\% increase) was used on the physical robot to consume the accumulation function in the environment. This slight increase in the value of \( \rho \) allows to overcome the effects of small tracking errors from the quadrotors and the effects of the discretization of the path for the persistent sensing task. We used the collision avoidance procedures from Chapter 6, with \( \rho_{safe} \) approximately equal to 3 times the radius of the quadrotor, and used the Min-Time stopping policy since it is effective, yet inexpensive to calculate.

The environment and, therefore, sensor measurements are simulated. Again, we present results in the two separate phases: 1) learning phase, and 2) path shaping phase, as described in Section 2.5.2.

Figure 7-3: Initial paths for the multi-robot hardware implementation
Learning Phase

The initial paths, used in the learning phase can be seen in Figure 7-3, where each robot has a “zig-zagging” path across a portion of the environment, and between both robots, most of the environment is initially traversed. In this figure, the green dots represent the points of interest in the environment. In the learning phase, we can see from Figure 7-4 that $\tilde{\phi}_r(q) \to 0$, $\forall q \in Q$ for one of the robots. Since the consensus error converges to zero in accordance with (iii) from Theorem 4 and shown in Figure 7-5, then we can conclude that the adaptation laws cause $\tilde{\phi}_r(q) \to 0$, $\forall q \in Q$, $\forall r$. This means that the union of both robots’ trajectories was rich enough to generate accurate estimates for all of the environment. Figure 7-6 shows that the mean over both robots of $\int_0^t w_r(\tau)(\tilde{\phi}_p(\tau))^2 d\tau$ converges to zero, in accordance with (ii) from Theorem 4. Finally, for this learning phase, we see in Figure 7-7 that the Lyapunov-like function $\mathcal{V}_4$ is monotonically non-increasing.

![Figure 7-4: Basis coefficient error](image)

![Figure 7-5: Consensus error](image)

![Figure 7-6: Integral parameter error](image)

![Figure 7-7: Lyapunov-like function in learning phase](image)
Path Shaping Phase

Figure 7-12 shows snapshots of the multi-robot implementation at different iteration times. We can see how the path evolves under this controller in the path shaping phase in Figures 7-13a to 7-13f. Figure 7-8 shows quantity $I_i^r(t)\|\dot{M}_i^r(t)\dot{e}_i^r(t) + \alpha_i^r(t)\|$ converging to zero, $\forall i, r$ in accordance to (i) from Theorem 4. Figure 7-9 shows the Lyapunov-like function $V_4$ monotonically non-increasing and reaching a limit during this phase. Figure 7-10 shows the persistent sensing task's stability margin increasing through time, as expected. The chattering in the stability margin is due to the discretization of the system. Reducing the length of time steps will reduce this chattering. Finally, Figure 7-11 shows the mean over all points of interest of the value of the accumulation function over time. As shown, this value initially increases on average due to the initialization of the system. Later, it starts to decrease and reaches an approximate steady-state behavior that corresponds to the locally optimal final configuration of the system.
Figure 7-12: Three snapshots of the hardware implementation of the multi-robot system during the path shaping phase at different iteration values. The system is comprised of two quadrotor robots. The paths, shown as the blue and red lines, connects all the waypoints corresponding to each robot. The points of interest in the environment are shown as green dots, and the size of a green dot represents the value of the accumulation function at that point. The robots are the blue-lit and red-lit quadrotors, and their sensor footprints are represented by the colored circles under the robots' positions.
Figure 7-13: Path shaping phase of the multi-robot system hardware implementation. The paths, shown as the blue and red lines (for robot 1 and robot 2, respectively), connect all the waypoints, shown as black circles. The points of interest are shown as green dots.
The hardware implementation was run for more than 10 times without any collision or deadlock between the robots, and generating informative paths that are practically identical to the simulated cases. This implementation was a centralized one, meaning that a single computer calculated the movement for all waypoints in both robots’ paths. These calculations were done serially, which caused the paths to reshape slower than they would in the case where each waypoints could perform its own calculations. The implementation required approximately three hours to achieve a near final informative path. If the implementation was performed in a decentralized way (each waypoint performing its own calculations), then the running time would be greatly reduced. For this case, theoretically, it should take approximately 10-15 seconds for the paths to converge to the final informative paths.
Chapter 8

Conclusion and Lessons Learned

In this thesis, we introduced the informative path problem, and generated an informative path controller for both a single-robot and a multi-robot case. This informative path controller is an adaptive controller based on a Voronoi-based coverage approach, building upon previous work in [34], that allows robots to learn the distribution of sensory information in the environment (the dynamic/static structure of the environment) and reshape their paths into informative paths, i.e. locally optimal paths for sensing dynamic regions in the environment. A Lyapunov-like proof shows that the controller will reshape the paths to locally optimal configurations and drive the estimated parameter vector error to zero, assuming that the robots’ trajectories are rich enough.

We observed how the weights assigned to the sensing task ($W_s$) and the neighbor distance ($W_n$) affected the behavior of the paths. Thus, these parameters can be used to tune the system according to the desired behavior. For example, if shorter paths are desired, then setting $W_n$ higher could be a good option. If, on the contrary, sensing is very important and short paths are not necessary, then having a high $W_s$ could generate desired results. Additionally, these weights can have a big effect on whether the final paths will intersect or not. It was observed in our simulations and experiments that a high $W_n$ tends to generate non-intersecting paths, which could be very useful if multiple robots are being used and collision avoidance is an issue.

The informative path controllers provide a strong tool for robots to explore and
generate useful paths in unknown environments for any sensing task such as cleaning, surveilling or, to our particular interest, persistent sensing. Therefore, an extension to the adaptive controllers was designed and proven to drive the paths into locally optical configurations that are beneficial for persistent sensing tasks. With this extended controller, the robots drive their paths in a direction where the stability margin of the persistent sensing task does not decrease, hence improving the performance of the robots executing the persistent sensing task. Increasing the stability margin has an additional benefit; it allows to overcome more easily un-modeled errors in the persistent sensing task, for example, tracking errors by the robots and discretization of the path.

We simulated and performed hardware experiments on many cases, some of them starting of as unstable persistent sensing tasks. In all cases, the stability margin (and thus the persistent task) improved through time indicating that the designed controllers were beneficial to the persistent task. Chattering in the stability margin was due to the discretization of the system and can be avoided by decreasing the time step size. In general the final paths using the persistent informative controller were different than the paths using the informative path controller. This is due to the additional restriction of the non-decreasing stability margin. This restriction may cause the system to get stuck on a local minimum early in the path shaping phase. One possible way to not get stuck on a local minimum for persistent sensing is to initially have the robots use the informative path controller, and once the robots achieve, for example, collision free paths then the persistent informative controller could be used to improve the persistent sensing task.

We presented a collision avoidance procedure for persistent sensing tasks. This procedure was based on computing collision zones and ensuring that only one robot occupied a given collision zone at any moment in time. This was performed by stopping robots before they entered a collision zone if another robot was inside that collision zone, and resuming motion only once the collision zone was free of other robots. We empirically investigated the performance of several different stopping policies to observe which one generated best results, and we presented a distributed
implementation with iRobot Create platforms, which generated successful results.

Finally, we combined the persistent informative controller with the collision avoidance algorithm and implemented a system comprised of two quadrotor robots. In this combined system, we assume that if two collision zones merged while robots where in each one, then an external collision avoidance procedure would solve the problem. An example of such a procedure could be to back-up the robot closest to the entering edge of the collision zone, or to use repulsive forces. In our simulations and experiments, this was not an issue and, thus, was not treated. Our hardware implementation was tested more than 10 times, all of them generating successful results that were nearly identical to simulated results. This final implementation shows the practical importance of our algorithms.
Appendix A

Tables of Important Symbols

Table A.1: Common symbols to all controllers

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q$</td>
<td>Convex bounded environment where the robotic tasks are being held</td>
</tr>
<tr>
<td>$q$</td>
<td>An arbitrary point in $Q$</td>
</tr>
<tr>
<td>$m$</td>
<td>number of parameters</td>
</tr>
<tr>
<td>$\phi(q)$</td>
<td>Sensory function at point $q$</td>
</tr>
<tr>
<td>$K(q)$</td>
<td>Vector of basis functions for the sensory function at point $q$</td>
</tr>
<tr>
<td>$a$</td>
<td>True parameter vector for the sensory function, $\phi(q) = K(q)^T a$</td>
</tr>
<tr>
<td>$c(q)$</td>
<td>Consumption rate at point $q$ for persistent sensing</td>
</tr>
<tr>
<td>$\sigma_j$</td>
<td>Standard deviation of the $j^{th}$ Gaussian basis function</td>
</tr>
<tr>
<td>$\mu_j$</td>
<td>Mean of the $j^{th}$ Gaussian basis function</td>
</tr>
<tr>
<td>$\rho_{\text{trunk}}$</td>
<td>Truncation distance for Gaussian basis functions</td>
</tr>
<tr>
<td>$s(q)$</td>
<td>Stability margin of point $q$ for persistent sensing</td>
</tr>
<tr>
<td>$S$</td>
<td>Stability margin of the persistent sensing task</td>
</tr>
<tr>
<td>$G(j)$</td>
<td>Gaussian function used to calculate truncated Gaussian basis</td>
</tr>
<tr>
<td>$W_w$</td>
<td>Weight assigned to neighboring waypoint distance</td>
</tr>
<tr>
<td>$W_s$</td>
<td>Weight assigned to the Voronoi-based coverage task</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Radius for circular sensor footprint for persistent sensing</td>
</tr>
<tr>
<td>$\tau_{\text{dwell}}$</td>
<td>Dwelling time between switching from one to zero</td>
</tr>
<tr>
<td>$v$</td>
<td>Arbitrary real constant scalar</td>
</tr>
<tr>
<td>$\Gamma$</td>
<td>Diagonal, positive-definite adaptation gain matrix</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Gain for adaptation law</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Radius for circular sensor footprint for persistent sensing</td>
</tr>
</tbody>
</table>
Table A.2: Symbols for single-robot controllers

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_r(t)$</td>
<td>The robot’s position at time $t$</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of waypoints in the robot’s path</td>
</tr>
<tr>
<td>$p_i$</td>
<td>The robot’s $i^{th}$ waypoint position</td>
</tr>
<tr>
<td>$V_i$</td>
<td>Voronoi partition of the $i^{th}$ waypoint</td>
</tr>
<tr>
<td>$M_i$</td>
<td>Mass of $V_i$</td>
</tr>
<tr>
<td>$\hat{M}_i$</td>
<td>Approximation of $M_i$</td>
</tr>
<tr>
<td>$L_i$</td>
<td>First mass moment of $V_i$</td>
</tr>
<tr>
<td>$\hat{L}_i$</td>
<td>Approximation of $L_i$</td>
</tr>
<tr>
<td>$C_i$</td>
<td>Centroid of $V_i$</td>
</tr>
<tr>
<td>$\hat{C}_i$</td>
<td>Approximation of $C_i$</td>
</tr>
<tr>
<td>$e_i, \dot{e}_i$</td>
<td>$C_i - p_i, \dot{C}_i - p_i$</td>
</tr>
<tr>
<td>$\alpha_i$</td>
<td>$W_n(p_{i+1} + p_{i-1} - 2p_i)$</td>
</tr>
<tr>
<td>$\beta_i, \dot{\beta}_i$</td>
<td>$M_i + 2W_n, \hat{M}_i + 2W_n$</td>
</tr>
<tr>
<td>$\phi_{pr}(t)$</td>
<td>Sensory function at the robot’s position, $\phi(p_r(t))$</td>
</tr>
<tr>
<td>$\dot{\phi}(q,t)$</td>
<td>The robot’s approximation of $\phi(q)$</td>
</tr>
<tr>
<td>$K_{pr}(t)$</td>
<td>Vector of basis functions at robot’s position, $K(p_r(t))$</td>
</tr>
<tr>
<td>$\dot{a}$</td>
<td>The robot’s parameter estimate</td>
</tr>
<tr>
<td>$\ddot{a}$</td>
<td>The robot’s parameter error, $\dot{a} - a$</td>
</tr>
<tr>
<td>$u_i$</td>
<td>Control input for the $i^{th}$ waypoint</td>
</tr>
<tr>
<td>$H_1(p_1, \ldots, p_n)$</td>
<td>Locational cost function</td>
</tr>
<tr>
<td>$\Lambda$</td>
<td>The robot’s weighted integral of basis functions</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>The robot’s weighted integral of sensory measurements</td>
</tr>
<tr>
<td>$K_i$</td>
<td>Control gain matrix for the $i^{th}$ waypoint in the robot’s path</td>
</tr>
<tr>
<td>$w$</td>
<td>The robot’s data weighting function</td>
</tr>
<tr>
<td>$\nu_1$, $\nu_2$</td>
<td>Lyapunov-like functions for informative path without and with persistent sensing, respectively</td>
</tr>
<tr>
<td>$b$</td>
<td>Term in adaptation laws for purposes of the Lyapunov proof</td>
</tr>
<tr>
<td>$\dot{a}_{pre}$</td>
<td>Time derivative of the robot’s parameter before projection</td>
</tr>
<tr>
<td>$I_{proj}$</td>
<td>Projection matrix</td>
</tr>
<tr>
<td>$T(t)$</td>
<td>Time it takes the robot to complete its path at time $t$</td>
</tr>
<tr>
<td>$\tau_o(q,t)$</td>
<td>Time the robot covers $q$ along its path at time $t$</td>
</tr>
<tr>
<td>$F(p_r)$</td>
<td>The robot’s sensor footprint</td>
</tr>
<tr>
<td>$\dot{s}(q)$</td>
<td>The robot’s estimated stability margin of point $q$</td>
</tr>
<tr>
<td>$\dot{S}$</td>
<td>The robot’s estimated stability margin of the persistent task</td>
</tr>
<tr>
<td>$\tau_w$</td>
<td>Time at which the adaptation data weighting function is set to zero</td>
</tr>
<tr>
<td>$I_i$</td>
<td>Boolean control input for waypoint movement in persistent sensing</td>
</tr>
<tr>
<td>$t_u'$</td>
<td>Most recent time the boolean input $I_i$ switches to one for the $i^{th}$ waypoint</td>
</tr>
<tr>
<td>Symbol</td>
<td>Definition</td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
</tr>
<tr>
<td>( p_r(t) )</td>
<td>Robot ( r )'s position at time ( t )</td>
</tr>
<tr>
<td>( n(r) )</td>
<td>number of waypoints in robot ( r )'s path for multi-robot system</td>
</tr>
<tr>
<td>( N )</td>
<td>number of robots in the multi-robot system</td>
</tr>
<tr>
<td>( p^i_r )</td>
<td>Robot ( r )'s ( i )th waypoint position</td>
</tr>
<tr>
<td>( V^i_r )</td>
<td>Voronoi partition of ( i )th waypoint in robot ( r )'s path</td>
</tr>
<tr>
<td>( M^i_r )</td>
<td>Mass of ( V^i_r )</td>
</tr>
<tr>
<td>( \tilde{M}^i_r )</td>
<td>Approximation of ( M^i_r )</td>
</tr>
<tr>
<td>( L^i_r )</td>
<td>First mass moment of ( V^i_r )</td>
</tr>
<tr>
<td>( \hat{L}^i_r )</td>
<td>Approximation of ( L^i_r )</td>
</tr>
<tr>
<td>( C^i_r )</td>
<td>Centroid of ( V^i_r )</td>
</tr>
<tr>
<td>( \hat{C}^i_r )</td>
<td>Approximation of ( C^i_r )</td>
</tr>
<tr>
<td>( e^i_r )</td>
<td>( C^i_r - p^i_r )</td>
</tr>
<tr>
<td>( \epsilon^i_r )</td>
<td>( \hat{C}^i_r - p^i_r )</td>
</tr>
<tr>
<td>( \alpha^i_r )</td>
<td>( W_n(p^i_{r+1} + p^i_{r-1} - 2p^i_r) )</td>
</tr>
<tr>
<td>( \beta^i_r )</td>
<td>( M^i_r + 2W_n )</td>
</tr>
<tr>
<td>( \dot{\beta}^i_r )</td>
<td>( \tilde{M}^i_r + 2W_n )</td>
</tr>
<tr>
<td>( \phi_{p_r}(t) )</td>
<td>Sensory function at robot ( r )'s position, ( \phi(p_r(t)) ).</td>
</tr>
<tr>
<td>( \hat{\phi}_r(q, t) )</td>
<td>Robot ( r )'s approximation of ( \phi(q) )</td>
</tr>
<tr>
<td>( \mathcal{K}_{p_r}(t) )</td>
<td>Vector of basis functions at robot ( r )'s position, ( \mathcal{K}(p_r(t)) )</td>
</tr>
<tr>
<td>( \hat{a}_r )</td>
<td>Robot ( r )'s parameter estimate</td>
</tr>
<tr>
<td>( \tilde{a}_r )</td>
<td>Robot ( r )'s parameter error for the multi-robot system, ( \tilde{a}_r - a )</td>
</tr>
<tr>
<td>( u^i_r )</td>
<td>Control input for the ( i )th waypoint in robot ( r )'s path</td>
</tr>
<tr>
<td>( \mathcal{H}_2 )</td>
<td>Locational cost function</td>
</tr>
<tr>
<td>( \Lambda_r )</td>
<td>Robot ( r )'s weighted integral of basis functions</td>
</tr>
<tr>
<td>( \lambda_r )</td>
<td>Robot ( r )'s weighted integral of sensory measurements</td>
</tr>
<tr>
<td>( K^i_r )</td>
<td>Control gain matrix for the ( i )th waypoint in robot ( r )'s path</td>
</tr>
<tr>
<td>( w_r )</td>
<td>Robot ( r )'s data weighting function</td>
</tr>
<tr>
<td>( \mathcal{V}_3, \mathcal{V}_4 )</td>
<td>Lyapunov-like functions for multi-robot coverage and multi-robot persistent sensing, respectively</td>
</tr>
<tr>
<td>( l_{r,r'} )</td>
<td>Weighting between parameters for robots ( r ) and ( r' )</td>
</tr>
<tr>
<td>( L )</td>
<td>Graph Laplacian of the robot network</td>
</tr>
<tr>
<td>( b_r )</td>
<td>Terms in adaptation laws for purposes of the Lyapunov proof</td>
</tr>
<tr>
<td>( \dot{a}_{p,r} )</td>
<td>Time derivative of robot ( r )'s parameter before projection</td>
</tr>
<tr>
<td>( D_{\text{max}} )</td>
<td>Maximum distance the robot's can have and still communicate</td>
</tr>
<tr>
<td>( I_{\text{proj}} )</td>
<td>Projection matrix</td>
</tr>
<tr>
<td>( \Omega_j )</td>
<td>Vector containing the ( j )th parameter of each robot</td>
</tr>
<tr>
<td>( \zeta )</td>
<td>Consensus gain</td>
</tr>
<tr>
<td>( T_r(t) )</td>
<td>Time it takes robot ( r ) to complete its path at time ( t )</td>
</tr>
<tr>
<td>Symbol</td>
<td>Definition</td>
</tr>
<tr>
<td>--------</td>
<td>-----------</td>
</tr>
<tr>
<td>$\tau_r(q, t)$</td>
<td>Time robot $r$ covers $q$ along its path at time $t$</td>
</tr>
<tr>
<td>$F_r(p_r)$</td>
<td>Robot $r$'s sensor footprint</td>
</tr>
<tr>
<td>$\hat{s}_r(q)$</td>
<td>Robot $r$'s estimated stability margin of point $q$ for persistent sensing</td>
</tr>
<tr>
<td>$\hat{S}_r$</td>
<td>Robot $r$'s estimated stability margin of the persistent sensing task</td>
</tr>
<tr>
<td>$\tau_{uvr}$</td>
<td>Time at which the adaptation data weighting function is set to zero for robot $r$</td>
</tr>
<tr>
<td>$I_i^r$</td>
<td>Boolean control input for shutting down waypoint movement in persistent sensing</td>
</tr>
<tr>
<td>$t_{ui}^r$</td>
<td>Most recent time the boolean input $I_i^r$ switches to one for the $i^{th}$ waypoint in robot $r$'s path</td>
</tr>
</tbody>
</table>

Table A.4: Symbols for collision avoidance

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Z(q, t)$</td>
<td>Accumulation function value at time $t$ for point $q$</td>
</tr>
<tr>
<td>$N_q(t)$</td>
<td>Set of robots whose sensor footprints are over the point $q$ at time $t$</td>
</tr>
<tr>
<td>$\theta_r$</td>
<td>Path and position parametrization variable for robot $r$</td>
</tr>
<tr>
<td>$\psi_r$</td>
<td>Robot $r$'s path</td>
</tr>
<tr>
<td>$\rho_{safe}$</td>
<td>Safety radius for robot $r$ for collision avoidance</td>
</tr>
<tr>
<td>$B$</td>
<td>Safety disc for collision avoidance</td>
</tr>
<tr>
<td>$P_{r,r'}$</td>
<td>Set of collision configurations between robots $r$ and $r'$</td>
</tr>
<tr>
<td>$P_r$</td>
<td>Set of all possible collision locations for robot $r$</td>
</tr>
<tr>
<td>$C_r^k$</td>
<td>The $k^{th}$ pairwise disjoint connected set within $P_r$</td>
</tr>
<tr>
<td>$\nu_r$</td>
<td>Number of $C_r^k$ sets within $P_r$</td>
</tr>
<tr>
<td>$CZ_s^r$</td>
<td>The $s^{th}$ collision zone</td>
</tr>
<tr>
<td>$CZ_s^*$</td>
<td>The set of collision locations for robot $r$ within the $s^{th}$ collision zone</td>
</tr>
<tr>
<td>$\text{flag}_s$</td>
<td>Flag assigned to collision zone $s$</td>
</tr>
<tr>
<td>$T_{\text{enter}}^r$</td>
<td>Time robot $r$ would take to get from where it is right now until it enters $CZ_s^*$</td>
</tr>
<tr>
<td>$T_{\text{exit}}^r$</td>
<td>Time robot $r$ would take to get from where it is right now until it exits $CZ_s^*$</td>
</tr>
<tr>
<td>$TC_{\text{enter}}^r(q)$</td>
<td>Time robot $r$ covers point $q$ while it moves from its current position until it enters $CZ_s^*$</td>
</tr>
<tr>
<td>$TC_{\text{exit}}^r(q)$</td>
<td>Time robot $r$ covers point $q$ while it moves from its current position until it exits $CZ_s^*$</td>
</tr>
<tr>
<td>$I_{\text{cover}}^r(q)$</td>
<td>Boolean that is true if robot $r$'s footprint covers $q$ while it is stopped, and zero otherwise</td>
</tr>
<tr>
<td>$T_w$</td>
<td>Time window length for Time-Window stopping policy</td>
</tr>
<tr>
<td>$\kappa(t)$</td>
<td>Coefficient for worst case analysis</td>
</tr>
<tr>
<td>$\tau_{\text{total}}^r$</td>
<td>Total amount of time robot $r$ covers point $q$ in the worst case analysis</td>
</tr>
<tr>
<td>$T_r^c(t)$</td>
<td>Total maximum amount of time robot $r$ is stopped in one cycle along its path at time $t$ in the worst case analysis</td>
</tr>
<tr>
<td>$h_r(t)$</td>
<td>$(T_r(t) + T_r^c(t))/T_r(t)$</td>
</tr>
</tbody>
</table>
Bibliography


129


