Analysis of multi-agent systems under varying degrees of trust, cooperation, and competition

Pierson, Alyssa

http://hdl.handle.net/2144/20848

Boston University
BOSTON UNIVERSITY
COLLEGE OF ENGINEERING

Dissertation

ANALYSIS OF MULTI-AGENT SYSTEMS UNDER
VARYING DEGREES OF TRUST, COOPERATION, AND
COMPETITION

by

ALYSSA PIERSON
B.S., Harvey Mudd College, 2010
M.S., Boston University, 2016

Submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

2017
Approved by

First Reader

Mac Schwager, PhD
Assistant Professor of Aeronautics and Astronautics
Stanford University

Second Reader

Sean Andersson, PhD
Associate Professor of Mechanical Engineering
Associate Professor of Systems Engineering

Third Reader

Calin Belta, PhD
Professor of Mechanical Engineering
Professor of Systems Engineering
Professor of Bioinformatics

Fourth Reader

Roberto Tron, PhD
Assistant Professor of Mechanical Engineering
Assistant Professor of Systems Engineering
Know what? Bitches get stuff done.

- Tina Fey
Acknowledgments

This dissertation would not exist without the help and support of many people in my life. First and foremost, I thank my advisor, Mac Schwager, for his guidance and direction throughout my PhD. Over countless discussions, coffees, and meetings, he has taught me how to be a better thinker and more focused problem solver. Thank you for believing in me. I am also grateful to Professor Calin Belta, Professor Sean Andersson, and Professor Roberto Tron, whose discussions and advice in the BU robotics lab have shaped my research.

I am incredibly thankful to have worked alongside various co-authors and collaborators over the years. I thank Lucas Figueiredo, for his help wrangling the m3pis. I also thank Professor Luciano Pimenta for all the insightful Skype discussions and insight on our coverage control work. I am so happy to have worked with Dr. Armin Ataei and Professor Ioannis Paschalidis, and thank them for our collaborations on the quadrotor pursuit. A huge thanks my Zijian Wang, not only for his robot whisperer skills, but also for his infectious positive attitude and making the lab a happier place over the years.

For the past few years, the Multi-Robot Systems Lab has been my home. To all my friends and colleagues between Boston University and Stanford, thank you for making the lab an enjoyable place to be. A special thanks to Kevin, Eric, Cristi, Xi, Trevor, and Brian for many day-to-day discussions and distractions.

I want to thank the other mentors in my life, for their encouragement and advice. First, to Professor Lori Bassman, for not only guiding me through undergrad, but has always supported me throughout the years, ready to listen and give advice. Thank you for being a role model and a friend. Second, to Gary Thomasulo, for taking me under his wing at Cobham and helping me navigate life after college, thank you for being my advocate.
None of this would be possible without the support and love from my friends and family. I thank my sister Kelsey, for keeping me sane through the years and constantly challenging me to be a better person. I love you, you know. To my parents, Lee and Sheryl, who shaped me into the person I am today. They taught me to work hard and play harder, to always have a sense of humor, and that any bad day can be made better by getting outdoors. Thank you for the unconditional love and encouragement through the years. Finally, I thank my husband, Bryan Teague, for his constant support and love, superb quesadilla-making skills (the perfect writing fuel), and for being my cheerleader, my partner in all the adventures.

- Alyssa Pierson
Multi-agent systems rely heavily on coordination and cooperation to achieve a variety of tasks. It is often assumed that these agents will be fully cooperative, or have reliable and equal performance among group members. Instead, we consider cooperation as a spectrum of possible interactions, ranging from performance variations within the group to adversarial agents. This thesis examines several scenarios where cooperation and performance are not guaranteed. Potential applications include sensor coverage, emergency response, wildlife management, tracking, and surveillance. We use geometric methods, such as Voronoi tessellations, for design insight and Lyapunov-based stability theory to analyze our proposed controllers. Performance is verified through simulations and experiments on a variety of ground and aerial robotic platforms. First, we consider the problem of Voronoi-based coverage control, where a group of robots must spread out over an environment to provide coverage. Our approach adapts online to sensing and actuation performance variations with the group. The robots have no prior knowledge of their relative performance, and
in a distributed fashion, compensate by assigning weaker robots a smaller portion of the environment. Next, we consider the problem of multi-agent herding, akin to shepherding. Here, a group of dog-like robots must drive a herd of non-cooperative sheep-like agents around the environment. Our key insight in designing the control laws for the herders is to enforce geometrical relationships that allow for the combined system dynamics to reduce to a single nonholonomic vehicle. We also investigate the cooperative pursuit of an evader by a group of quadrotors in an environment with no-fly zones. While the pursuers cannot enter the no-fly zones, the evader moves freely through the zones to avoid capture. Using tools for Voronoi-based coverage control, we provide an algorithm to distribute the pursuers around the zone’s boundary and minimize capture time once the evader emerges. Finally, we present an algorithm for the guaranteed capture of multiple evaders by one or more pursuers in a bounded, convex environment. The pursuers utilize properties of the evader’s Voronoi cell to choose a control strategy that minimizes the safe-reachable area of the evader, which in turn leads to the evader’s capture.
4.5.2 Herding with \( n > 1 \) Sheep

4.6 3D Simulations
   4.6.1 Relocation to Goal
   4.6.2 Target Tracking
   4.6.3 Robustness to Noise

4.7 Experiments

4.8 Summary

5 Cooperative Multi-Quadrotor Pursuit of an Evader in an Environment with No-Fly Zones
   5.1 Related Work
   5.2 Zone-Aware Pursuit in Free Space
   5.3 Pursuing an Evader in a No-Fly Zone
   5.4 Quadrotor Modeling and Control
      5.4.1 Robust Model Predictive Control
   5.5 Experiments
      5.5.1 Simulated Evader
      5.5.2 Human-Controlled Evader
   5.6 Summary

6 Guaranteed Capture of Multiple Evaders with Multiple Pursuers
6.4.2 Global Policy with Guaranteed Capture ................. 155
6.4.3 Decentralized Local Policy ................................. 156
6.5 Simulations .......................................................... 157
  6.5.1 Multi-Evader Pursuit in 2D ................................. 158
  6.5.2 Multi-Evader Pursuit in 3D ................................. 160
  6.5.3 Comparison to Other Methods ............................. 162
6.6 Experiments .......................................................... 165
  6.6.1 4 Autonomous Evaders ...................................... 166
  6.6.2 1 Human Evader + 3 Autonomous Evaders ............... 168
6.7 Summary .............................................................. 170

7 Conclusions and Future Work ................................. 171

A Index of Notation ................................................. 175
   A.1 Voronoi-Based Coverage Control ............................. 175
   A.2 Herding ............................................................. 176
   A.3 Cooperative Pursuit .......................................... 176
   A.4 Guaranteed Capture of Multiple Evaders ................. 177

References ............................................................. 178

Curriculum Vitae ...................................................... 187
List of Tables

4.1 Success rate for varying $\frac{K_n}{v_{\text{max}}}$ ................................. 110
4.2 Success rate for varying $\frac{K_n}{r}$ ............................................. 111
6.1 Summary of pursuer policies ...................................................... 162
## List of Figures

1-1 Spectrum of Cooperation in Multi-Agent Systems  

3-1 Weighted vs Regular Voronoi Partition  

3-2 Illustration of $K_i u_i$  

3-3 Shared Voronoi Boundary Between Two Agents  

3-4 Shared Point Along Boundary  

3-5 Initial and Final Configurations of Sensing-Based Task Simulation  

3-6 Cost Over Time for Sensing-Based Task Simulation  

3-7 Performance Weights Over Time for Sensing-Based Task Simulation  

3-8 Initial and Final Configurations with Randomized Health  

3-9 Cost and Performance Weights for Randomized Health  

3-10 Initial and Final Configurations for Actuation-Based Task  

3-11 Cost and Performance Weights for Actuation-Based Task  

3-12 Convergence of $\|\tilde{K}_i\|$  

3-13 Experiment Setup with m3pi Robots  

3-14 Initial and Final Configurations for Sensing-Based Experiment  

3-15 Cost and Performance Weights Over Time  

3-16 Original and Simulated Noise on Image  

3-17 Initial and Final Configurations with Noise  

3-18 Performance Weights Over Time  

3-19 Sensing Cost over Multiple Trials  

3-20 Initial and Final Configurations for Actuation-Based Experiment
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{V_i}$</td>
<td>Centroid of $V_i$</td>
</tr>
<tr>
<td>HJI</td>
<td>Hamilton-Jacobi-Isaacs</td>
</tr>
<tr>
<td>LMI</td>
<td>Linear Matrix Inequality</td>
</tr>
<tr>
<td>$M_{V_i}$</td>
<td>Mass of $V_i$</td>
</tr>
<tr>
<td>MPC</td>
<td>Model Predictive Control</td>
</tr>
<tr>
<td>RMPC</td>
<td>Robust Model Predictive Control</td>
</tr>
<tr>
<td>$V_i$</td>
<td>Voronoi Cell for agent $i$</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

For multi-agent systems to be practical in real-world settings, it is important to understand how variations in cooperation and competition affect the group. As robotic systems become cheaper and more prevalent, we can integrate these systems into more aspects of everyday life. Potential applications of multi-agent systems include sensor coverage over environments, emergency response, wildlife management, tracking, and surveillance. Each of these problems require different levels of cooperation, coordination, and trust. To effectively address these tasks, there is a need to quantify the relationships among the agents. It is easy to assume that all agents will be fully cooperative, with equal and reliable performance across the group. Accounting for performance variations in the controller design makes the system more robust to faults and failures within the group. Other situations may include agents that are neither inclined nor opposed to cooperation. These agents aren’t malicious, but merely non-cooperative to the rest of the group. In this situation, incorporating this non-cooperative behavior into the control design allows the group to achieve tasks that are impossible under a fully cooperative model. Finally, in the case of tracking, treating the agents as active evaders provides a “worst-case” bound on the performance of the pursuer agents.

The aim of this dissertation is to analyze scenarios where cooperation and performance are not guaranteed. We consider cooperation as a spectrum of possible interactions, ranging from performance variations within a cooperative group to ad-
versarial agents, illustrated in Figure 1.1. Using geometric methods, such as Voronoi tessellations, we design controllers based on the level of cooperation in the system. We use Lyapunov-based stability theory to analyze our proposed controllers, and verify performance through simulations and experiments on a variety of ground and aerial robotic platforms.

**Figure 1.1:** For multi-agent systems, problems can be defined along a spectrum of cooperation and competition. Above are a few examples of various problem types, ranging from an all-equal, all-cooperative group to a group with malicious agents.

First, we explore a scenario wherein all robots cooperate, but have varying levels of performance abilities. We illustrate this example through a Voronoi-based coverage control algorithm in Chapter 3. Next, we examine non-cooperative multi-agent herding in Chapter 4, which illustrates an example of a heterogeneous system that is not hostile, but not cooperative. To examine cases with potentially adversarial agents, we first consider the problem of tracking an evader in Chapter 5. In this example, a group of quadrotors work together to track an evader as it moves through the environment while avoiding no-fly zones. Finally, Chapter 6 presents our algorithm for the guaranteed capture of multiple evaders by one or more pursuers in a bounded, convex environment.

Our work draws inspiration from several key concepts within multi-agent systems research. Much of our Voronoi-based coverage control work is built upon a decen-
tralized, multi-robot algorithm first proposed by Cortés et al., wherein the robots continuously move towards the centroid of their Voronoi cell to provide sensor coverage over some environment [Cortes et al., 2004]. This is commonly referred to as the “move-to-centroid” algorithm. We present an overview of this algorithm in Chapter 2. Taking cues from this work, we first explore how performance variations impact the move-to-centroid algorithm in Chapter 3. In Chapter 5, the we use a variation on the centroidal policy to assign pursuers to possible evader locations. Finally, in Chapter 6, we use properties of the Voronoi tessellation to design the control laws for the pursuers. Another source of inspiration is within consensus and formation control. Distributed flocking systems, such as those described by Olfati-Saber [Olfati-Saber, 2006], utilize nearest-neighbor rules to control a group of agents. Flocking is pertinent to multi-agent systems, as it provides a framework for a large group of cooperative agents working towards a common objective, with applications in self-organization, transportation, and relocation. An extension to this is coordinated tracking, which is illustrated in Egerstedt et. al’s formation control research [Egerstedt and Hu, 2001]. While both of these problems assume a cooperative group, our herding work in Chapter 4 extends some of these behaviors to a system with non-cooperative agents.

1.1 Synopsis

This dissertation encompasses several problems along the spectrum of cooperation, from performance variations to adversarial agents. Each problem addresses a different level of cooperation between agents in the system, and formulates a control strategy based on these interactions. In summary, the problems addressed within this dissertation are:

1. **Performance Variations in Coverage Control**

   We introduce online, adaptive performance weights for robots conducting a
Voronoi-based coverage control algorithm. The robots adjust to relative performance differences, giving lower-performing robots smaller areas of responsibility.

2. Non-Cooperative Multi-Agent Herding

In this problem, a group of herders must relocate a non-cooperative herd, akin to shepherding. The non-cooperative herd agents are not directly controllable, and respond to the herders with a repulsive potential field. By exploiting geometries of the system, we design controllers for the herders that allow them to relocate the herd to a goal.

3. Cooperative Pursuit with No-Fly Zones

A group of quadrotors track an evader through the environment while avoiding no-fly zones. The evader may freely enter these zones, and when it does, the quadrotors must arrange themselves about the obstacle to minimize the distance to the evader once it emerges.

4. Guaranteed Capture of Multiple Evaders

We present a decentralized control strategy for one or more pursuers to guarantee capture of multiple evaders in a bounded, convex environment where all agents have equal speeds. The strategy utilizes properties of the Voronoi tessellation to reduce the safe area of the evader, resulting in capture in finite time.

Each of these problems represent varying levels of trust and cooperation in the system, and manifests with different applications in multi-agent systems. The remainder of this chapter provides an expanded introduction to each of these topics, and the full formulation of these problems are found in subsequent chapters.
1.2 Voronoi-Based Coverage Control

In Chapter 3, we present our decentralized control strategies for groups of robots that can adapt to individual deficiencies and performance variations within the group. Our work considers a team of robots carrying out a coverage control task, wherein the robots must spread out across the environment while covering areas of high importance. Once deployed, the robots may be given a variety of tasks, such as sensing, surveillance, or servicing events within the environment. The group is heterogeneous in that some robots may perform better than others. Differences in performance are assessed by various parameters of the robot and will depend on the given task. We assume the robots are unaware of their relative performance with respect to the group, as in the case of a real-world implementation. We propose an algorithm that incorporates online learning and adaptive control into Voronoi-based coverage control. The robots will learn “performance weightings” which indicate their performance metric relative to the other robots in the group. This is calculated using the robot’s sensing and actuation errors and local communication with nearby neighbors. If a robot is determined to have a low performance weight, this will shrink the size of the robot’s dedicated coverage area, while a high performance weight corresponds to a robot taking charge of a larger portion of the environment.

We divide coverage tasks into two categories: sensing-based and actuation-based. For sensing-based tasks, the robots move to the centroid of their weighed Voronoi cell, and use sensors to monitor the environment. Here, performance variations can occur due to different sensors on each robot, differences in quality or degradation of the sensors, and sensor creep in long-term deployments. External factors, such as dust or fog, can also affect the quality of each robot’s camera and our algorithm adjusts for these variations online. For actuation-based tasks, the Voronoi cells provide regions over which a robot responds to events, requiring the robot to move around its cell as
events occur. Both speed and accuracy of the robot are important to the successful completion of their tasks. We combine speed and accuracy into a single metric for the performance weighting. Variations in actuation can occur for several reasons. The robots may have different hardware components, or the terrain over which the robots are moving can be varied. For example, if the environment contains a mix of paved roads, forested areas, or dirt paths, the terrain will affect the robots’ actuation performance. Overall, internal and external variations lower the efficiency of the entire group. Our algorithm accounts for these variations and adapts the performance weights accordingly.

Applications of sensing-based or actuation-based tasks can be illustrated with several examples. For a sensing-based task, consider a group of robots that have been deployed to take pictures of a region following some disaster, such as an earthquake or building collapse. Here, the robots’ Voronoi cells dictate the part of the region for each robot to photograph. Due to varying levels of dust and debris, each robot’s picture quality can vary. Using our adaptive weighting algorithm, these lower-performing robots can be compensated for and given a smaller region in the environment. One example of an actuation-based task is a group of agricultural robots that are deployed to water crops within a field. Once deployed, the robots must move around their Voronoi cells to water crops as needed. Errors in their speed and precision affect how successfully they deliver water, which can affect the overall health of the crops. Another application to consider is the problem of illegal fishing. Some experts estimate this accounts for approximately 1 in 5 fish caught in the wild, which may account for up to $23.5 billion worth of fish in the world market [Pew Charitable Trusts, 2015]. Here, Voronoi-based coverage control could be used to distribute a robotic network across an area of interest. As boats enter a robots’ region, the robots may need to move closer or track patterns of the boats to identify illegal activity.
Errors in actuation can compromise how quickly or effectively a robot could track a suspicious boat.

1.3 Non-Cooperative Multi-Agent Herding

Chapter 4 examines the problem of non-cooperative herding, analogous to shepherding wherein dogs drive a herd of sheep to a goal location. In this system, the “sheep” agents naturally run away from the “dog” robots, and by designing controllers for the dogs, we can relocate the sheep to some desired region in the environment. We propose a feedback control strategy for the dogs to coordinate their positions with one another to partially encircle the herd. This partial encirclement applies pressure to the herd and moves it in a desired direction, which allows the dogs to steer the herd towards the goal. In Chapter 4, we examine the case of 2D herding, akin to shepherding in a field, and its extension to 3D, applicable to aerial and underwater robots. We begin by introducing control strategies for two dogs controlling a single sheep in 2D, and show that under certain geometrical constraints, the dynamics of this system reduce to the well-known unicycle kinematic robot. Using this insight, we map a simple linear control strategy for the unicycle robot back into a nonlinear feedback control law for the two dogs. We generalize this approach to the case of an arbitrary number of dogs driving a single sheep, as well as to the general case of multiple dogs driving multiple sheep in 2D and 3D. For the 3D case, we present extensions to target tracking, a decentralized approach, and a study on its robustness to noise. Performance of the control strategies is demonstrated in Matlab simulations for both 2D and 3D and hardware experiments in 2D with Pololu m3pi robots in a motion capture environment.

Although we use the dog-sheep analogy to describe this system, this is only for illustrative purposes. In general the “dogs” are robotic agents under our control. The
“sheep” agents are not under our control, but assumed to behave with herd dynamics. The herd members respond to the dogs with a repulsive potential field commonly used to model the response of herding animals to perceived threats. Our control strategy could be useful in wildlife management, as well as other applications. For example, in Australia, helicopters are used to muster cattle for large-scale relocation. This dangerous profession requires pilots to fly at low altitudes and perform quick maneuvers, which results in as many as 10 deaths per year [Lane, 2011]. Implementing our control strategy on teams of UAVs to autonomously muster cattle can reduce risk to humans and limit fatalities. Another application is managing wildlife populations in national parks, where it is necessary to monitor animals and steer them away from potential environmental dangers. In 3D, this may include aquatic life or aerial animals. Our controllers may also apply to the problem of micro-manipulation of bacteria with magnetic fields [Becker et al., 2014]. In the case of an emergency evacuation, human crowds could be directed by robots using our control strategy.

We consider this a non-cooperative multi-robot problem, since the objective of the dogs is to steer the sheep, but the sheep are not actively inclined nor opposed to being steered. This scenario lies somewhere between a fully a cooperative setting, in which all robots work towards the goal, and a fully adversarial setting, in which two teams of robots work against each other toward opposite goals.

1.4 Cooperative Multi-Quadrotor Pursuit of an Evader

In Chapter 5, we study the problem of coordinating a group of cooperative pursuers to track an evader as it moves through the environment. The pursuers are quadrotors that must maneuver an environment with “no-fly zones,” which are regions that the quadrotors cannot enter. Some examples of no-fly zones are buildings, restricted airspace, or forests. The evader can freely move through these zones, while the
pursuing quadrotors must position themselves around the boundary of the no-fly zones, ready to pursue the evader when it eventually emerges. Furthermore, the quadrotors can sense the position of the evader when it is in free space, but may not have information about the evader when it is inside a no-fly zone. Our algorithm uses Robust Model Predictive Control (RMPC) tools to always stay a prescribed safe distance away from the no-fly zones while pursuing the evader, and it adapts methods from Voronoi-based coverage control to position the quadrotors when the evader is inside a no-fly zone. We demonstrate our algorithm in hardware experiments with three quadrotors pursuing a manually-controlled ground robot.

Our algorithm is useful in a number of applications of emerging importance, such as search and rescue, robotic aerial videography, and security and surveillance. One example is where the evader is a suspected criminal fleeing the scene of a crime, and the pursuers are police surveillance drones. The pursuers track the suspect as they flee, but also must avoid buildings, bridges, trees and other environmental obstacles. If the suspect enters a building where the drones cannot follow, our algorithm will position the drones around the building so that they can continue tracking the suspect once it re-emerges, illustrated by Figure 5.2. Other settings where the algorithm may be useful include tracking a lost person or endangered animal in a park, or the subject of a sports film such as a snowboarder or mountain biker. In this setting, treating the person as an evader provides a “worst-case” bound on the tracking performance. Additionally, the pursuers will not “capture” the target, but rather rendezvous and maintain a close distance with the target.

When the evader is in free space, the quadrotors pursue it directly using an RMPC controller to guarantee that they do not enter the no-fly zones. When an evader enters a no-fly zone, we adopt a three-part strategy for the quadrotors: (i) compute the reachable set of the evader, (ii) tessellate the reachable set with a centroidal Voronoi
tessellation, then (iii) drive the quadrotors to points on the perimeter of the no-fly zone that are as close as possible to the centroids of the tessellation. The quadrotors may not have access to the evader’s position inside the no-fly zone, so the evader’s reachable set is a conservative estimate that grows in time, eventually filling up the no-fly zone. The RMPC control approach is again used to keep the quadrotors safely outside of the no-fly zones at all times as they navigate to their positions on the perimeter of the no-fly zone.

1.5 Multi-Agent Pursuit-Evasion

In Chapter 6, we consider the problem of coordinating a group of pursuer robots to capture a group of evader robots within a convex, bounded environment. The pursuers do not know the evaders’ policy, but instead move to minimize the safe-reachable area of an evader to guarantee capture. Our pursuit strategy is inspired by the area-minimization policy in [Huang et al., 2011, Zhou et al., 2016] for multiple pursuers chasing a single evader in a 2D environment.

We present three main results in this chapter. First, we extend the results of [Huang et al., 2011, Zhou et al., 2016] to environments of arbitrary dimension, making it practical for aerial robots in 3D environments. Second, we propose a pursuer algorithm for the case of multiple evaders, and prove that it guarantees the capture of all evaders in finite time, however, this algorithm requires global information. Finally, we present a local, decentralized version of the multi-evader pursuit algorithm that performs as well as the global policy in simulation, and can be implemented on-board robots with local sensing and communication. Our approach is decentralized among the pursuers, wherein each pursuer only needs information about itself and its Voronoi neighbors to compute its control algorithm. Simulation results demonstrate the performance of our algorithm in 2D and 3D environments, as well as a comparison
with other control laws. Hardware experiments were conducted in a motion capture environment using Ouijabots and GoPiGo robots. For the experiments, each robot runs its control strategy on board its Raspberry Pi 2. We also conducted experiments with a human-controlled evader that could not avoid capture.

Our algorithm is useful in a number of applications of emerging importance, such as security and surveillance, search and rescue, and wildlife monitoring. The problem is inspired by the classic game of “cops and robbers,” [Nowakowski and Winkler, 1983], [Aigner and Fromme, 1984] where the “cop” attempts to capture the “robber” while the robber simultaneously attempts to avoid capture. With the rise of recreational and industrial use of drones comes a significant threat of drones wandering into restricted airspace over airports, public buildings, protected parklands, or other sensitive areas. Our algorithm provides a practical method by which a fleet of autonomous pursuit drones can neutralize such threats. It is equally applicable to intercepting rogue watercraft in seaports, as well as vehicles or suspicious people on land. The algorithm is also useful for search and rescue applications, where survivors may not know they are being sought, therefore a search strategy must assume no knowledge of the survivors’ policy. The algorithm may be useful for wildlife monitoring, where the pursuers are autonomous vehicles tasked with tracking or tagging wildlife, and the wildlife may react to the pursuers as threats.

1.6 Organization

The remained of this dissertation is organized as follows: we first introduce background concepts pertinent to multi-robot research in Chapter 2. Chapter 3 presents our work on incorporating performance variations within a Voronoi-based coverage control algorithm. Chapter 4 examines the problem of non-cooperative herding, wherein a group of herder agents must relocate a group of sheep agents to a goal
region in the environment. In Chapter 5, we extend tools from Voronoi-based coverage control as a method to assign pursuers to possible locations of an evader. Chapter 6 presents our algorithm to capture multiple evaders in a convex, bounded environment. Finally, Chapter 7 presents our conclusions and directions of future work.
Chapter 2

Background

This chapter presents several mathematical preliminaries used throughout this dissertation. We first detail the style of notation and mathematical definitions used in subsequent chapters. Next, we present several results from graph theory, which will be used in multiple proofs. We also present several theorems from Lyapunov-based stability theory that will be used to analyze our nonlinear systems. Finally, we include a background on locational optimization and Voronoi-based coverage control.

2.1 Notation

Here, we summarize the mathematical notation and definitions to be used for the remainder of this dissertation. We use $\mathbb{R}^N$ to denote the $N$-dimensional Euclidean space, and use 2D to refer to $\mathbb{R}^2$ and 3D to refer to $\mathbb{R}^3$. For the purposes of this dissertation, the symbols $>$, $<$, $\geq$, and $\leq$ apply element-wise to vectors. For a matrix $A$, we use $A > 0$ to denote a positive-definite matrix, and $A \geq 0$ to denote a positive semi-definite matrix. The identity matrix is always denoted $I$. Rotation matrices from reference frame $B$ to a new frame $A$ are denoted $^A R^B$, and $(^A R^B)^T =^B R^A$. For a set $\Omega$, the boundary of the set is denoted $\partial \Omega$, while partial derivatives are denoted with $\frac{\partial}{\partial r}$. The $\ell^2$ norm is denoted $\| \cdot \|$. Vectors will not be differentiated from scalars with a bold font, but instead we state the dimension of a variable when it is declared. The derivative of $x$ with respect to time is either denoted $\frac{dx}{dt}$, or the shorter form $\dot{x}$ when convenient.
2.2 Graph Theory

For multi-agent systems, graphs can model how information propagates through the network. In this section, we summarize a few key concepts from graph theory relevant to this dissertation. Let $G = (\mathcal{V}, \mathcal{E})$ be an undirected graph defined by set of vertices $\mathcal{V} = \{v_1, ..., v_n\}$ and edges $\mathcal{E} = \{e_1, ..., e_m\}$, where $e_k = \{v_i, v_j\}$ and $v_i, v_j \in \mathcal{V}$. Here, we consider each agent as a vertex in the graph. The agents share an edge if they are in communication with one another. We define “neighbors” of an agent as the set of all agents that share an edge, and denote this set as $\mathcal{N}_i$ for agent $i$. For an undirected graph, if $v_j \in \mathcal{N}_i$, then it follows that $v_i \in \mathcal{N}_j$.

In Chapter 3, we also use the weighted graph Laplacian matrix, $L$. For neighboring agents $i$ and $j$, let $d_{ij}$ be the weight of an edge connecting vertices $v_i$ and $v_j$. We write $L$ as

$$L = \begin{bmatrix}
  \cdots & \sum_{j \in \mathcal{N}_i} d_{ij} & L_{ij} \\
  L_{ij} & \cdots
\end{bmatrix},$$

where

$$L_{ij} = \begin{cases}
  -d_{ij} & \text{for } j \in \mathcal{N}_i \\
  0 & \text{otherwise}
\end{cases}.$$ 

The following is a well-known result from graph theory and will be useful for the proofs of our weightings adaptation laws in Chapter 3.

**Theorem 1 (Graph Laplacians).** For a connected, undirected graph, the weighted Laplacian is symmetric and positive semi-definite. There exists exactly one zero eigenvalue, with the associated eigenvector $\mathbf{1} = [1, ..., 1]^T$. Furthermore, $L\mathbf{1} = \mathbf{1}^T L = 0$, and $\mathbf{x}^T L \mathbf{x} > 0, \forall \mathbf{x} \neq c\mathbf{1}, c \in \mathbb{R}$.

**Proof.** See [Godsil and Royle, 2001].
2.3 Stability Theory

In this dissertation, we propose a variety of controllers for multi-agent systems. To analyze the behavior of these proposed controllers, we use Lyapunov-based stability theory, which provides sufficient conditions for stability in nonlinear systems. Here, we present several theorems and basic results from Lyapunov’s methods as well as LaSalle’s Invariance Principle. Consider a system with dynamics

\[ \dot{x} = f(x), \]  

(2.1)

where \( f : D \to \mathbb{R}^n \) is a locally Lipschitz map from some domain \( D \) onto \( \mathbb{R}^n \). Without loss of generality, we can define our system such that \( x = 0 \) is an equilibrium point of the system. The following defines asymptotic stability of the system.

**Theorem 2** (Asymptotic Stability [Khalil, 2002]). Let \( x = 0 \) be an equilibrium point of (2.1) and \( D \subset \mathbb{R}^n \) be a domain containing \( x = 0 \). Let \( V : D \to \mathbb{R} \) be a continuously differentiable function such that

\[ V(0) = 0 \quad \text{and} \quad V(x) > 0 \quad \text{in} \quad D - \{0\}, \]

\[ \dot{V}(x) \leq 0 \quad \text{in} \quad D. \]

Then \( x = 0 \) is stable. Moreover, if

\[ \dot{V}(x) < 0 \quad \text{in} \quad D - \{0\}, \]

then \( x = 0 \) is asymptotically stable.

**Proof.** See [Khalil, 2002].

In some cases, the Lyapunov function fails to prove stability, particularly in cases where \( \dot{V} \) is negative semi-definite. We can use LaSalle’s Invariance Principle to demonstrate stability under these conditions.

**Theorem 3** (LaSalle’s Invariance Principle [Khalil, 2002]). Let \( \Omega \subset D \) be a compact set that is positively invariant with respect to (2.1). Let \( V : D \to \mathbb{R} \) be a continuously
differentiable function such that \( \dot{V}(x) \leq 0 \) in \( \Omega \). Let \( E \) be the set of all points in \( \Omega \) where \( \dot{V}(x) = 0 \). Let \( M \) be the largest invariant set in \( E \). Then every solution starting in \( \Omega \) approaches \( M \) as \( t \to \infty \).

Proof. See [Khalil, 2002].

To show that \( x(t) \to 0 \), we need to establish that the largest invariant set is the origin. When \( V(x) \) is positive definite, the following corollary can be used to extend LaSalle’s theorem.

**Corollary 1** (Corollary 4.1, [Khalil, 2002]). Let \( x = 0 \) be an equilibrium point for (2.1). Let \( V : D \to \mathbb{R} \) be a continuously differentiable positive definite function on a domain \( D \) containing the origin \( x = 0 \), such that \( \dot{V}(x) \leq 0 \) in \( D \). Let \( S = \{ x \in D \mid \dot{V}(x) = 0 \} \) and suppose that no solution can stay identically in \( S \), other than the trivial solution \( x(t) \equiv 0 \). Then, the origin is asymptotically stable.

For nonautonomous systems of the form

\[
\dot{x} = f(t, x),
\]

we can define exponential stability as

**Definition 1** (Exponential Stability [Khalil, 2002]). The equilibrium point \( x = 0 \) is exponentially stable if there exist positive constants \( c, k, \) and \( \lambda \) such that

\[
\|x(t)\| \leq k\|x(t_0)\|e^{-\lambda(t-t_0)}, \quad \forall \|x(t_0)\| < c
\]

and globally exponentially stable if this is satisfied for any initial state \( x(t_0) \).

### 2.4 Locational Optimization

Here, we present a summary of Voronoi-based coverage control, a foundation for several chapters in this dissertation. Cortés et al. first proposed a controller that deployed a group of robots to provide sensor coverage of the environment [Cortes
et al., 2004]. The agents’ controller utilized a Voronoi tessellation to divide the environment among the robots, then each robot moved to the centroid of its Voronoi cell. We refer to this strategy as the “move-to-centroid” controller. Under this decentralized strategy, the robots converge to a locally optimal coverage configuration for the environment.

Consider a group of $n$ robots in a bounded, convex environment $Q \subset \mathbb{R}^2$. Points in $Q$ are denoted $q$, and the positions of individual agents are denoted $p_i \in Q$. Let $\{V_1, ..., V_n\}$ be the Voronoi cells of $Q$, where each cell is defined as

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

For our region $Q$, we also define an integrable function $\phi : Q \to \mathbb{R}_{>0}$ to represent the areas of importance in the environment. Large values of $\phi(q)$ correspond to areas of more importance than small values of $\phi(q)$. We assume all agents have knowledge of this function.

To perform the sensing coverage task, each agent is equipped with a sensor, and the quality of sensing is assumed to decrease according to a differentiable, strictly increasing function $f : \mathbb{R}_{\geq 0} \to \mathbb{R}$. Specifically, $f(\|q - p_i\|)$ describes the cost of the measurement of information at $q$ by a sensor at $p_i$. Intuitively, the cost of sensing a point increases as the agent moves farther away from that point. A cost function that describes the total sensing cost of the system can be written [Cortes et al., 2004]

$$\mathcal{H}_V(p_1, ..., p_n) = \int_Q \min_{i \in \{1, ..., n\}} f(\|q - p_i\|) \phi(q) dq.$$ 

The minimum inside the integral refers to the fact that a point $q$ should be covered by the sensor with the best sensing performance at $q$. Given the properties of $f(\|q - p_i\|)$,
the integral becomes the Voronoi partition of $Q$, as

$$H_V(p_1, ..., p_n) = \sum_{i=1}^{n} \int_{V_i} f(\|q - p_i\|) \phi(q) dq,$$

where $V_i$ is the Voronoi cell of agent at $p_i$. While this cost function takes many forms, most coverage control applications assume $f(\|q - p_i\|) = \frac{1}{2} \|q - p_i\|^2$. Thus, the cost function becomes

$$H_v(p_1, ..., p_n) = \sum_{i=1}^{n} \int_{V_i} \frac{1}{2} \|q - p_i\|^2 \phi(q) dq.$$

Intuitively, a low value of $H_V$ indicates a good coverage configuration of the robots across the environment. We can also define a “mass” and “centroid” of the Voronoi cell, as

$$M_{V_i} = \int_{V_i} \phi(q) dq, \quad \text{and} \quad C_{V_i} = \frac{1}{M_{V_i}} \int_{V_i} q \phi(q) dq.$$

By definition, $\phi(q)$ is strictly positive, thus $M_{V_i}$ and $C_{V_i}$ are analogous to the physical mass and centroid of the cell. Although there exists a complex dependency between the position of the robots and the geometry of their Voronoi cells, a surprising result from locational optimization [Drezner, 1995] is that

$$\frac{\partial H_V}{\partial p_i} = - \int_{V_i} (q - p_i) \phi(q) dq = -M_{V_i} (C_{V_i} - p_i),$$

which implies that the critical points of $H_V$ will correspond to the robots positioned at the centroids of their Voronoi cells, or $p_i = C_{V_i}$ for all $i$. Critical points can correspond to either local minima, local maxima, or saddle points. Cortés introduced a controller that drives robots only to the critical points corresponding to local minima of the cost function [Cortes et al., 2004]. For $\dot{p}_i = u_i$, where $u_i$ is the control input to an agent, the move-to-centroid controller is

$$u_i = k_p (C_{V_i} - p_i).$$
In Chapter 3, we present a modification to this algorithm using weighted Voronoi cells with adaptive performance weights, and derive a similar controller. Using weighted Voronoi cells, we implement the same move-to-centroid positional control law, but add an additional adaptation law to the cell weights. This allows us to adjust for performance variations within the group while maintaining the Voronoi-based coverage.
Chapter 3

Adapting to Performance and Sensing Variations in Multi-Robot Coverage Control

In this chapter, we present our work integrating an adaptive controller that accounts for sensing and performance variations within a move-to-centroid coverage control algorithm. The work in this chapter combines three of our publications on the topic: two conference papers presented at the International Symposium on Robotics Research (ISRR) and the International Conference on Robotics and Automation (ICRA), and a journal paper recently accepted for publication in the International Journal of Robotics Research (IJRR) [Pierson and Schwager, 2013, Pierson et al., 2015, Pierson et al., 2016b]. This problem illustrates an example of a multi-agent system whose agents cooperate, but have performance variations within the group.

In our work, we present weighting adaptation laws that run in parallel to the main move-to-centroid positional controller. We first present an adaptation law in the context of sensing-based tasks, where the robot knows its performance parameters and then adapts the performance weight relative to the performance of the robot’s neighbors. Next, we use an actuation-based task to illustrate the case where a robot does not know its performance parameters, and must estimate them online. Using a Lyapunov-style analysis, we show the robots converge to a locally optimal coverage configuration, while the weightings converge to a set of values defined by the performance error. Our algorithm is demonstrated through Matlab simulations and
hardware experiments using Pololu m3pi robots.

The rest of the chapter is organized as follows: Section 3.1 summarizes related work pertaining to our coverage control problem. Section 3.2 outlines the main assumptions, definitions, and problem formulation. Section 3.3 presents the proof of convergence to a locally optimal configuration when no parameter estimation is needed. Section 3.4 incorporates a parameter estimator for an actuation-based task and illustrates how the proof of convergence can be written for this added uncertainty. Results of Matlab simulations are presented in Section 3.5, experimental results are given in Section 3.6, and summary in Section 3.7.

3.1 Related Work

Voronoi-based coverage control is a common problem in multi-robot systems. A decentralized, multi-robot algorithm was first proposed by Cortés et. al, commonly called the move-to-centroid algorithm, and discussed in Chapter 2 [Cortes et al., 2004, Cortés, 2010]. Through this online algorithm, the robots can move through the environment to some locally optimal configuration. The algorithm is decentralized, meaning it scales with the size of the group, and does not require communication with some central agent. Voronoi-based coverage control builds upon previous work in locational optimization, such as the optimal placement of retail facilities, as well as algorithms for data compression (i.e. “vector quantization”) [Drezner, 1995]. However, a limitation of this original algorithm is that all agents are assumed to have equal characteristics, while in practice a group of agents may have a wide range of performance and sensing capabilities.

Other extensions of the Voronoi-based coverage control algorithm use a weighted Voronoi diagram, also called a power diagram. These weightings account for heterogeneity among the robots. Pavone et. al showed that different cell weights allow the
agents to take on varying sensing responsibility by modifying the accountable area for each robot [Pavone et al., 2009]. Here, a lower relative weight resulted in a smaller cell area for each robot. When the performance or health of the robots is known globally, the weightings can be used to adjust accountability within the group. To link the cell weightings to performance, Pimenta et. al used the sensing radius of the robot as its weighting [Pimenta et al., 2008]. The sensing radius allows the robot to take on a relatively larger cell if it can sense points in its region, and conversely shrink its cell if it has a smaller sensing radius. Another approach used the weightings as an energy-efficiency metric [Kwok and Martinez, 2007]. Here, a lower efficiency corresponds to a lower weighting, allowing the high power robots to compensate for the low power robots in the Voronoi tessellation. Marier et. al quantified sensor health with the Voronoi weights, assigning low-performing robots smaller areas of coverage [Marier et al., 2011, Marier et al., 2013]. In their implementation, the health was incorporated as a multiplicative factor in a relevant cost function. In Mahboubi et. al’s work, the coverage radius of each agent is utilized to minimize the holes within their Voronoi polygons [Mahboubi et al., 2014b]. By minimizing the coverage holes individually, the group is able to achieve an improved global coverage. An extension to this work employs multiplicatively-weighted Voronoi cells, which have curved boundaries similar to a circular sensor footprint [Mahboubi et al., 2014a]. When the features of the environment are changing, Lee et. al present a strategy that tracks the time-varying information density function within the Voronoi-based coverage configuration [Lee et al., 2015].

Within the existing research utilizing weighted Voronoi cells, all to our knowledge assume the correct weightings are known beforehand. In contrast, our approach learns the performance weightings online using only information about the robot’s performance and the data from its neighbors. Preliminary versions of these results
appeared in conference versions, which incorporated sensing performance [Pierson and Schwager, 2013] and actuation variation [Pierson et al., 2015] into an adaptive trust weighting. The journal version of this work streamlined the theoretical approach to both cases, as well as provided new simulation results and hardware experiments [Pierson et al., 2016b]. The relative performance-based adaptation occurs in parallel to the Voronoi-based coverage control algorithm.

3.2 Problem Set-Up

Consider a group of \( n \) robots in a bounded, convex environment\(^1 \) \( Q \subset \mathbb{R}^2 \). Points in \( Q \) are denoted \( q \), and positions of individual agents are denoted \( p_i \in Q \). Prior coverage control algorithms use the standard Voronoi partition, and for our work, we will use the weighted Voronoi partition. Let \( \{V_1, \ldots, V_n\} \) be the Voronoi cells of \( Q \), and recall from Chapter 2 that

\[
V_i = \{q \in Q \mid \|q - p_i\| \leq \|q - p_j\|, \quad \forall j \neq i\}.
\]

The weighted Voronoi partition, also known as the Power Diagram [Aurenhammer, 1987], introduces a weight for each of the cells. Consider \( w_i \) as the cell weight for each robot \( i \), and let \( \{W_1, \ldots, W_n\} \) be the weighted Voronoi cells in \( Q \), with each cell defined as

\[
W_i = \{q \in Q \mid \|q - p_i\|^2 - w_i \leq \|q - p_j\|^2 - w_j, \quad \forall j \neq i\}. \quad (3.1)
\]

Figure 3.1 illustrates the differences between a regular and weighted Voronoi diagram.

\(^1\)It may be desirable to provide Voronoi coverage in a non-convex environment. Strategies to do so are common in literature, and it is straightforward to extend our approach to a non-convex environment following the techniques of [Pimenta et al., 2008], [Lekien and Leonard, 2010], or [Breitenmoser et al., 2010].
are the weighted Voronoi diagram. Here, Robot 2 has been assigned a lower weight relative to its neighbors, and we see that its weighted Voronoi cell is smaller. Similarly, Robot 6 has been assigned a higher relative weight, and its weighted cell is larger. Each robot uses its performance weightings as the Voronoi cell weighting, calculated using information about its individual performance health and the performance of its neighbors. We map these performance parameters to a scalar $h_i$, which indicates the relative “health” of the robot. In the event that the robot cannot directly measure its performance parameters, we also introduce an online estimator.

For our region $Q$, we also define an integrable function $\phi: Q \rightarrow \mathbb{R}_{>0}$ to represent the areas of importance in the environment. Large values of $\phi(q)$ correspond to areas of more importance than small values of $\phi(q)$. All robots are assumed to have
knowledge of this function. When the robots do not know this function, techniques have been developed to learn the function online [Schwager et al., 2009, Martínez, 2010].

3.2.1 Locational Optimization

In Chapter 2, we stated basic nomenclature and results from Voronoi-based coverage control. Here, we extend these notions to weighted Voronoi partitions, which will be used to formulate our coverage problem. Recall that a coverage cost function [Cortes et al., 2004, Pavone et al., 2009] for the robot network over the region $Q$ is formulated as

$$
\mathcal{H}_V(p_1, ..., p_n) = \sum_{i=1}^{n} \int_{V_i} \frac{1}{2} \| q - p_i \|^2 \phi(q) dq.
$$

(3.2)

Intuitively, a low value of $\mathcal{H}_V$ indicates a good coverage configuration of the robots across the environment. For our work, we use the weighted Voronoi cell, also known as the Power Diagram, given in (3.1). We formulate a similar cost function from the weighted cells that incorporates the robots’ performance in the cost. Consider a scalar-valued “health” $h_i$ indicative of the robot’s individual performance. We incorporate this into the cost function as

$$
\mathcal{H}_W(p_1, ..., p_n) = \sum_{i=1}^{n} \int_{W_i} \frac{1}{2} \left( \| q - p_i \|^2 - h_i \right) \phi(q) dq.
$$

(3.3)

Note this new cost function is calculated over the weighted Voronoi cell, $W_i$. We also define $M_{W_i}$ and $C_{W_i}$ of the weighted Voronoi cell as

$$
M_{W_i} = \int_{W_i} \phi(q) dq \quad \text{and} \quad C_{W_i} = \frac{1}{M_{W_i}} \int_{W_i} q\phi(q) dq.
$$

By definition, $\phi(q)$ is strictly positive, thus $M_{W_i}$ is analogous to the physical mass of the weighted Voronoi cell, and $C_{W_i}$ is analogous to the centroid. While there exists
a complex dependency between the position of the robots and the geometry of the Voronoi cells, similar to the derivation shown in Chapter 2, we find

$$\frac{\partial H_W}{\partial p_i} = - \int_{W_i} (q - p_i) \phi(q) dq = -M_{W_i}(C_{W_i} - p_i),$$

which implies the critical points of $H_W$ will correspond to the robots positioned at the centroids of their weighted Voronoi cells [Pavone et al., 2009, Marier et al., 2011], or $p_i = C_{W_i}$ for all $i$. Critical points can correspond to either local minima, local maxima, or saddle points. Cortés introduced a gradient-based controller that drives the robots to critical points corresponding to local minima of (3.2) [Cortes et al., 2004]. Using (3.4) we will introduce a similar controller that only drives the robots towards the local minima of (3.3). The global optimization of (3.3) is a variant of the p-center problem, and known to be NP-hard [Drezner, 1995], thus we only consider local minima of $H_W$. When referring to optimal coverage configurations, we mean locally optimal configurations. Variations on the control law that attempt to find the global minima of (3.2) via exploration are discussed in [Salapaka et al., 2003, Schwager et al., 2008].

### 3.2.2 Sensor Quality Model

We assume that the robots are equipped with passive sensors, such as cameras, microphones, IR cameras, or RF listening devices. Each of these sensors have performance variations that affect its overall sensor health. We introduce the function $\gamma_i(\cdot)$ that relates the sensor health to the quality of data sensed by the robot. Here, $\gamma_i(\cdot)$ represents a value measured by the robot’s sensor, such as pixel brightness for a camera, which can be used to compare relative quality between agents. The quality of $\gamma_i(\cdot)$ is influenced by the position of the robot, the point the robot is sensing, and the health of the sensor. For passive sensors with an unobstructed line-of-sight signal path, the
received signal quality for a point distance $d$ away is proportional to $d^2$ [Goldsmith, 2005]. To account for individual sensor variations, we consider a constant offset $h_i$ as a “sensor health” for robot $i$. In digital cameras, this offset $h_i$ is akin to a noise footprint, which varies by camera model as well as the sensor size [Lukas et al., 2006].

While in practice it is not necessary to know the exact form, for our convergence proofs we assume that $\gamma_i(\cdot)$ is approximated by [Pierson and Schwager, 2013]

$$\gamma_i(p_i, q, h_i) = -\alpha (\|q - p_i\|^2 - h_i), \quad (3.5)$$

where $h_i$ is the sensor performance health for robot $i$ and $\alpha$ is a scaling factor. Note this equation for $\gamma_i(p_i, q, h_i)$ shares a similar structure with the weighted Voronoi cell definition (3.1). We also see that $\gamma_i(p_i, q, h_i)$ can take on different values by different robots looking at the same point $q$. For example, if $\gamma_i$ represents the color at point $q$, variations in camera sensors may produce a different value for a robot $i$ located at $p_i$ versus a robot $j$ located at $p_j$.

It is not necessary for the robots to know $h_i$ or $\alpha$ directly, so long as $\gamma_i(\cdot)$ can be calculated from properties of the measurements. The variables $\alpha$ and $h_i$ shape the approximation of how the sensing quality of some point $q$ decreases as the sensor $p_i$ moves further away from $q$. Although we use the example of cameras, $\gamma_i(\cdot)$ can also model other passive sensors whose performance of sensing a point a distance $r$ away decreases quadratically (in contrast, active sensors, such as lidar, decrease by $\sim r^4$).

In Section 3.6, we present an experiment wherein the robots compare the variance of their noisy images with their neighbors to determine relative performance.

### 3.2.3 Robot Model

This section describes our models for the dynamics of the robots. Let the robots have integrator dynamics. We can equivalently assume that low-level controllers are in
place to cancel existing dynamics and enforce the desired control input. The robots are also given some additive actuation error, denoted by $\Delta_i$ and calculated from the robot’s performance parameters. The dynamics of the robot can then be written

$$\dot{p}_i = u_i + \Delta_i.$$ 

Here, $u_i$ is the control input and $\Delta_i$ is the actuation error. For ground robots in $\mathbb{R}^2$, we assume that $\Delta_i$ has the form

$$\Delta_i = \begin{bmatrix} \Delta_{i,1} & \Delta_{i,2} \\ \Delta_{i,3} & \Delta_{i,4} \end{bmatrix} u_i,$$ \hspace{1cm} (3.6)

which leads to

$$\dot{p}_i = K_i u_i,$$

$$K_i = \begin{bmatrix} 1 + \Delta_{i,1} & \Delta_{i,2} \\ \Delta_{i,3} & 1 + \Delta_{i,4} \end{bmatrix}.$$ \hspace{1cm} (3.7)

We assume $K_i$ remains a positive definite matrix. Practically speaking, this implies that the robot drives within $90^\circ$ of its intended direction, as shown in Figure 3·2. If there is no actuation error, then $K_i$ reduces to the identity matrix. In practice, we may not know the value of these actuation parameters, in which case we propose an estimator to find $\hat{K}_i$, derived from known quantities and discussed in Section 3.4.

We define the communication network as an undirected graph in which two robots share an edge if they share Voronoi cell boundaries. This is also known as a Delaunay graph. The set of neighbors for any robot $i$ can then be written as

$$\mathcal{N}_i := \{j | V_i \cap V_j \neq \emptyset\}.$$ 

We assume that the robots are able to communicate with their neighbors and share information, such as their position, weight, and sensing data. Additionally, we assume robots are able to compute their weighted Voronoi cell, as defined by (3.1), which can
be computed with well-known algorithms [Cortes et al., 2004, Marier et al., 2011, Salapaka et al., 2003].

Using these models for the sensor and actuation errors, we can now present the problem to be addressed in this chapter:

**Problem 1. (Adapting to Sensing and Actuation Variations in Coverage Control)** Given a group of robots with positions $p_i$ and performance weights $w_i$, find the control laws for $u_i$ that locally minimize the coverage cost function given in (3.3) and adaptation laws for $\dot{w}_i$ that compensate for variations in either sensor health modeled in (3.5) or variations in actuation errors modeled in (3.6).

### 3.3 Adapting to Sensing Variations

In this section we propose a controller and adaptation law that drives the robots to an optimal configuration while adjusting weightings to account for variations in sensing performance. To account for these performance variations, we will define a move-to-centroid control law, and an adaptation law to change the weightings of the robots based on sensor data. We will also assume that the sensor values can be measured directly. We will then prove that the control law drives the robots
Figure 3-3: For neighbors $i$ and $j$ the green line highlights their shared Voronoi cell boundary. The weightings adaptation law compares sensing data along points in this boundary.

to converge asymptotically to a stable equilibrium configuration corresponding to a local minimum of the sensing cost function. For this section, we assume there is no actuation error in the robots.

We propose to use the control law

$$\dot{p}_i = u_i = k_p (C_{W_i} - p_i), \quad (3.8)$$

where $C_{W_i}$ is the centroid of robot’s Voronoi cell and $k_p$ is a proportional gain. For the weightings adaptation law, using the sensing function described in (3.5), we propose

$$\dot{w}_i = \frac{k_w}{2M_{W_i}} \sum_{j \in N_i} \left( \int_{b_{ij}} \left[ \gamma_i(p_i, q, h_i) - \gamma_j(p_j, q, h_j) \right] dq \right), \quad (3.9)$$

where $k_w$ is a positive proportional gain constant, and $b_{ij}$ is the shared cell boundary line between neighboring agents $i$ and $j$. Essentially, this compares the values of the sensing data between two neighbors over shared points along their boundaries, which is illustrated in Figure 3-3.
To simplify this expression, we notice from (3.5) the integrand becomes

$$\gamma_i(p_i, q, h_i) - \gamma_j(p_j, q, h_j) = -\alpha \left( \|q - p_i\|^2 - h_i - \|q - p_j\|^2 + h_j \right).$$

However, we are evaluating the point $q$ along the cell boundary, so we know it satisfies (3.1)

$$\|q - p_i\|^2 - w_i = \|q - p_j\|^2 - w_j.$$ 

Combining these expressions, we find that for points $q$ along the cell boundary,

$$\gamma_i(p_i, q, h_i) - \gamma_j(p_j, q, h_j) = -\alpha (w_i - w_j - h_i + h_j).$$

This yields another form of the weightings adaptation law,

$$\dot{w}_i = -\frac{\alpha k_{w}}{2M_{W_i}} \sum_{j \in N_i} [(w_i - h_i) - (w_j - h_j)] d_{ij}, \quad (3.10)$$

where $d_{ij}$ is the length of the shared boundary $b_{ij}$. Note that (3.9) and (3.10) are mathematically equivalent, however, the robots can only calculate (3.9) from sensing data. We will use (3.10) in the proof of our following theorem.

**Theorem 4.** For a group of $n$ robots in a bounded, convex environment $Q$, using the control law (3.8) and the weightings adaptation law (3.9), the robots converge to the centroids of their weighted Voronoi cells,

$$\|p_i - C_{W_i}\| \to 0 \quad \forall \ i \in n. \quad (3.11)$$

Furthermore, the weightings satisfy

$$(w_i - w_j) \to (h_i - h_j) \quad \forall \ i, j. \quad (3.12)$$

**Proof.** To prove (3.11) and (3.12), we will invoke a global version of LaSalle’s Invariance Principle ([Bullo et al., 2009], Theorem 1.20). First, we will introduce a continuously differentiable Lyapunov-like function $V$ that is similar in form to our coverage cost function (3.3). We use this to show that all trajectories of the system
are bounded, and the function is non-increasing, thus \( \dot{V} \leq 0 \). Then we use LaSalle’s Principle to prove the claims of the theorem. Consider the function

\[
V = \sum_{i=1}^{n} \int_{W_i} \frac{1}{2} (\|q - p_i\|^2 - w_i) \phi(q) dq.
\]  

(3.13)

Its time derivative is

\[
\dot{V} = -\sum_{i=1}^{n} \dot{p}_i^T \int_{W_i} (q - p_i) \phi(q) dq - \sum_{i=1}^{n} \dot{w}_i \int_{W_i} \phi(q) dq.
\]

The derivative can then be written in two parts as

\[
\dot{V}_1 = -\sum_{i=1}^{n} \dot{p}_i^T \int_{W_i} (q - p_i) \phi(q) dq \dot{p}_i,
\]

\[
\dot{V}_2 = \sum_{i=1}^{n} -\frac{M_{W_i}}{2} \dot{w}_i,
\]

where \( \dot{V} = \dot{V}_1 + \dot{V}_2 \). Utilizing (3.4) and substituting our controller (3.8) into \( \dot{V}_1 \) yields

\[
\dot{V}_1 = -\sum_{i=1}^{n} [k_p(C_{W_i} - p_i)]^T (q - p_i) \phi(q) dq
\]

\[=
\sum_{i=1}^{n} -k_p M_{W_i} \|C_{W_i} - p_i\|^2 \leq 0.\]

We can also see that plugging in our adaptation law (3.9) into \( \dot{V}_2 \), this simplifies as

\[
\dot{V}_2 = \sum_{i=1}^{n} -\frac{k_w}{4} \sum_{j \in N_i} \left( \int_{b_{ij}} \gamma_i(p_i, q) - \gamma_j(p_j, q) \right) dq \]

\[= 0.\]

Thus, we have \( \dot{V} \leq 0 \).

Given that the derivative \( \dot{V} \leq 0 \), we can infer that the trajectories of the robots \( p_i(t) \) are bounded. To determine whether the weightings are bounded, consider the
vector form of $\dot{w}$, formulated from (3.10). First, we define

$$w(t) = \begin{bmatrix} w_1(t) \\ \vdots \\ w_n(t) \end{bmatrix}, \quad M^{-1} = \begin{bmatrix} \frac{1}{Mw_1} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \frac{1}{Mw_n} \end{bmatrix}, \quad h = \begin{bmatrix} h_1 \\ \vdots \\ h_n \end{bmatrix},$$

and

$L = \begin{bmatrix} \sum_{j \in \mathcal{N}_i} d_{ij} L_{ij} \\ \vdots \\ L_{ij} \end{bmatrix},$ where $L_{ij} = \begin{cases} -d_{ij} & \text{for } j \in \mathcal{N}_i \\ 0 & \text{otherwise} \end{cases}.$

Here, $M^{-1}$ is a diagonal matrix of positive entries and $L$ is the weighted Laplacian of the neighbor graph, which is known to be positive semi-definite [Godsil and Royle, 2001, Horn and Johnson, 1990], and it can be shown that the product $M^{-1}L$ is positive semi-definite. Hence we can write the derivative in vector form as

$$\dot{w}(t) = -\frac{\alpha k_w}{2} M^{-1} L w(t) + \frac{\alpha k_w}{2} M^{-1} L h.$$ (3.14)

We see that $w(t)$ is the state of a marginally stable filter defined by (3.14). Note that since the health is static, $h$ is bounded and we have the input to the filter defining $w(t)$ is bounded. From input-to-state properties of marginally stable filters we know that for the weights $w(t)$ to go unbounded the driving signal $\frac{\alpha k_w}{2} M^{-1} L h$ must lie in the null space of the dynamics matrix $-\frac{\alpha k_w}{2} M^{-1} L$ of the filter. However, since the input $h$ is itself multiplied by $\frac{\alpha k_w}{2} M^{-1} L$, the driving signal $\frac{\alpha k_w}{2} M^{-1} L h$ can have no component in the null space of $-\frac{\alpha k_w}{2} M^{-1} L$. Therefore, $w(t)$ remains bounded, and this shows that all trajectories of the system $(p_i(t)$ and $w_i(t)$ for all $i$) remain bounded.

By the properties of stable linear filters, we know that as the input approaches a limit, the state will also approach a limit [Khalil, 2002], which satisfies the steady-state equation

$$w \rightarrow \{w_\infty | 0 = -\frac{\alpha k_w}{2} M^{-1} L (h - w_\infty) \}.$$ 

Solving for $w_\infty$, we find

$$\frac{\alpha k_w}{2} M^{-1} L w_\infty = \frac{\alpha k_w}{2} M^{-1} L h$$

$$Lw_\infty = Lh.$$
Since L is the Graph Laplacian, this is equivalent to

\[ w_{i,\infty} - w_{j,\infty} = h_i - h_j, \]

proving (3.12) from Theorem 4.

Given that we have shown \( \dot{V} \leq 0 \), to complete the proof we must find the largest invariant set within the set defined by \( \dot{V} = 0 \). We can see that \( \dot{V} = 0 \) occurs when \( p_i = C_{W_i} \). From our control law (3.8), this itself is an invariant set. Therefore, by LaSalle’s Invariance Principle, we have that

\[ p_i(t) \to C_{W_i}(t) \quad \text{as} \quad t \to \infty, \]

proving (3.11) from Theorem 4.

**Remark 1.** This proof shows that using the weightings adaptation law (3.9), our weightings converge to a set of values relating the sensing performance among agents. Overall, the convergence of the weightings implies they will reach static values, which in conjunction with the move-to-centroid controller (3.8) means the robots will find final locations in the environment. While changing the weightings causes a change in the cell boundaries, thus a change in the centroids, the weightings eventually converge to an invariant set, which means the positions of the robots will eventually reach their centroids.

**Remark 2.** Theorem 4 guarantees convergence of the difference between the weightings to the difference between the corresponding health factors, but it does not guarantee the convergence of each weighting to its corresponding health factor. This is expected, as the robots only have relative sensor measurements to compare. However, weighted Voronoi cell boundaries are calculated by a relative difference (3.1), so any constant offset is canceled from both sides.

**Remark 3.** The proof structure was chosen to illustrate the parallels with the proof in Section 3.4 for variations in actuation. The proof can also be written with a simpler structure that only uses a Lyapunov-like function and LaSalle’s Invariance Principle, and is presented in the preliminary conference version [Pierson and Schwager, 2013].

If the weightings are initially assigned the correct values, it implies all robots will agree in the compared sensing data values. In the final result of Theorem 4,
the final positions of the robots in the environment are as good as if the correct performance weightings were known beforehand. Using this result, we can show the positional control law (3.11) and weightings adaptation law (3.12) provide a solution to Problem 1. Corollary 2 formalizes this by demonstrating that when the weightings converge to a static set of values, our Lyapunov-like function shares the same minima as our coverage cost function.

**Corollary 2.** Given the convergence of the performance weightings to the set described by (3.12), the local minima of our Lyapunov-like function (3.13) are equal to the local minima of our coverage cost function (3.3).

*Proof.* For some constant \( c \), (3.12) implies \( w_{i,\infty} = h_i - c \) for all \( i \). Substituting this into (3.13), we write

\[
V = \sum_{i=1}^{n} \int_{W_i} \frac{1}{2} \left( \|q - p_i\|^2 - h_i - c \right) \phi(q) dq
\]

\[
= H_W + c \int_{Q} \phi(q) dq.
\]

Since \( c \int_{Q} \phi(q) dq \) is constant, the set of positions \((p_1, ..., p_n)\) that comprise a local minima of (3.13) will also comprise a local minima of (3.3). \qed

We can simplify the computational complexity of the weightings adaptation law by comparing sensing values only at a single point, instead of across the entire boundary \( b_{ij} \). The motivation to compare sensing functions at fewer points, as illustrated in Figure 3-4, is that it may be faster and computationally easier than the boundary calculation, albeit less robust. Corollary 3 shows that a simplification to a single point still maintains the convergence of the weightings to an invariant set, as well as convergence of the location of the robots to their centroids.

**Corollary 3.** The claims of Theorem 4 also hold true for the adaptation law

\[
\dot{w}_i = \frac{k_w}{2M_{W_i}} \sum_{j \in N_i} \left( \gamma_i(p_i, q_c, h_i) - \gamma_j(p_j, q_c, h_j) \right),
\]

(3.16)
where $q_c$ is any point in $b_{ij}$.

Proof. Using (3.16) in place of the previous weightings adaptation law (3.9), noting the weighted graph Laplacian becomes the standard graph Laplacian [Godsil and Royle, 2001], the same proof and arguments hold from Theorem 4.

3.4 Adapting to Actuation Variations

In our sensing task example, we proposed a control law to drive robots to the centroids of their Voronoi cells, as well as a weightings adaptation law to compare sensor data to compensate for low-performing agents. This section will examine an actuation-based task. In contrast to the previous case, it is necessary to use a parameter estimator in determining the performance health of a robot. We use our estimated parameters in a weightings adaptation law similar to the previous case to adjust the robots’ weightings. We then prove that the robots still converge asymptotically to a stable equilibrium configuration corresponding to the local minimum of our cost function (3.3). We also prove our parameter estimator converges to the true parameter value.

The robots’ dynamics are composed of two parts: the desired input $u_i$ and some actuation error $\Delta_i$ defined in (3.6). Our desired input will be the move-to-centroid
controller introduced in the last section, written

\[ u_i = k_p(C_{W_i} - p_i). \]

Substituting this controller into the dynamics defined in (3.7), we write

\[ \dot{\hat{p}}_i = u_i + \Delta_i = K_i(C_{W_i} - p_i), \tag{3.17} \]

where \( C_{W_i} \) is the centroid of the robot’s weighted Voronoi cell and \( K_i > 0 \) is the matrix representing control gain and actuation error from (3.7). In the unlikely case that a Voronoi cell is empty, we evaluate (3.17) using the integral form of the gradient-descent based controller and let \( u_i = \Delta_i = 0 \).

We define a function mapping the matrix \( K_i \) to a scalar-valued health,

\[ h_i = g(K_i), \]

where \( h_i \) is the actuation performance “health,” and \( g(K_i) \) is a function of the properties of the matrix \( K_i \). We require that \( g(K_i) \) is bounded when \( K_i \) is bounded and continuous, however, the choice of \( g(K_i) \) is subjective to the desired performance metrics. Some common choices for \( g(K_i) \) include the matrix norm, determinant, trace, or eigenvalues. Here, the robots do not know \( K_i \), so we find an estimate of the matrix, denoted \( \hat{K}_i \). This mapping of the actuation parameters to a health can then be used in our weightings adaptation law. By changing the robot’s cell weights, we can adjust the size of their Voronoi cell corresponding to their relative performance. Similar to (3.9), we write the weightings adaptation law as

\[ \dot{w}_i = -\frac{k_w}{M_{W_i}} \sum_{j \in N_i} \left( (w_i - g(\hat{K}_i)) - (w_j - g(\hat{K}_j)) \right) \tag{3.18} \]

where \( k_w \) is a positive proportional gain constant. In the unlikely case of an empty
cell, where $M_{W_i}$ goes to zero, we let $\dot{w}_i = 0$.

### 3.4.1 Estimating $\hat{K}_i$

To compute the estimated matrix $\hat{K}_i$, we propose the following online estimator:

\begin{align}
\dot{\hat{K}}_i &= \lambda_i - \hat{K}_i \Lambda_i \\
\dot{\lambda}_i &= p_i (C_{W_i} - p_i)^T \\
\dot{\Lambda}_i &= (C_{W_i} - p_i)(C_{W_i} - p_i)^T.
\end{align}

(3.19)

We further simplify this expression as

\begin{align}
\hat{K}_i &= \int_0^t \dot{\lambda}_i(\tau) d\tau - \hat{K}_i \int_0^t \dot{\Lambda}_i(\tau) d\tau \\
&= \int_0^t K_i (C_{W_i}(\tau) - p_i(\tau)) (C_{W_i}(\tau) - p_i(\tau))^T d\tau \\
&\quad - \hat{K}_i \int_0^t (C_{W_i}(\tau) - p_i(\tau)) (C_{W_i}(\tau) - p_i(\tau))^T d\tau \\
&= - \hat{K}_i \Lambda_i(t)
\end{align}

(3.20)

where $\tilde{K}_i = (\hat{K}_i - K_i)$.

Note that although (3.19) and (3.20) are mathematically equivalent, the robots can only directly compute (3.19) because they do not have knowledge of the true error $K_i$, thus cannot calculate $\tilde{K}_i$. However, the form in (3.20) is useful for analysis. The behavior of our system compensating for actuation-based variations can be formalized in the following theorem.

**Theorem 5.** For a group of $n$ robots in a bounded, convex environment $Q$, using the control law (3.17), weightings adaptation law (3.18), and estimator for $\hat{K}_i$ (3.19), the robots converge to the centroids of their weighted Voronoi cells,

\begin{equation}
\lim_{t \to \infty} \|p_i(t) - C_{W_i}(t)\| = 0 \quad \forall \ i \in \{1, \ldots, n\}.
\end{equation}

(3.21)
Furthermore, the control gain matrix estimation error converges to the null space of $\Lambda_i(t)$,

$$\lim_{t \to \infty} \tilde{K}_i(t)\Lambda_i(t) = 0 \quad \forall \quad i \in \{1, \ldots, n\}. \tag{3.22}$$

Proof. The proof of (3.21) and (3.22) will invoke a global version of LaSalle’s Invariance Principle ([Bullo et al., 2009], Theorem 1.20), similar to the proof of Theorem 4. We will introduce a continuously differentiable Lyapunov-like function $\mathcal{V}$ which will include an additional term to account for the parameter estimation. This will be used to show that all trajectories of the system are bounded, and the function is non-increasing, $\dot{\mathcal{V}} \leq 0$. We then use LaSalle’s Principle to prove the claim of the theorem. Consider the function

$$\mathcal{V} = \sum_{i=1}^{n} \int_{\tilde{W}_i} \frac{1}{2} \left( \|q - p_i\|^2 - w_i \right) \phi(q) dq + \sum_{i=1}^{n} \frac{1}{2} \text{Tr}[\dot{\tilde{K}}_i\tilde{K}^T_i], \tag{3.23}$$

where $\text{Tr}[]$ is the trace. The time derivative of this function is

$$\dot{\mathcal{V}} = -\sum_{i=1}^{n} \dot{p}_i^T \int_{\tilde{W}_i} (q - p_i) \phi(q) dq - \sum_{i=1}^{n} \dot{w}_i \int_{\tilde{W}_i} \frac{1}{2} \phi(q) dq + \sum_{i=1}^{n} \text{Tr}[\dot{\tilde{K}}_i\tilde{K}^T_i].$$

We can break this into three parts

$$\dot{\mathcal{V}}_1 = -\sum_{i=1}^{n} \dot{p}_i^T \int_{\tilde{W}_i} (q - p_i) \phi(q) dq,$$

$$\dot{\mathcal{V}}_2 = -\sum_{i=1}^{n} \frac{1}{2} M_{\tilde{W}_i} \dot{w}_i,$$

$$\dot{\mathcal{V}}_3 = \sum_{i=1}^{n} \text{Tr}[\dot{\tilde{K}}_i\tilde{K}^T_i].$$

Substituting our controller (3.17) for $\dot{p}_i$, the time derivative $\dot{\mathcal{V}}_1$ becomes

$$\dot{\mathcal{V}}_1 = -\sum_{i=1}^{n} \int_{\tilde{W}_i} (q - p_i)^T \phi(q) dq \left[ K_i(C_{\tilde{W}_i} - p_i) \right]$$

$$= \sum_{i=1}^{n} -M_{\tilde{W}_i} (C_{\tilde{W}_i} - p_i)^T K_i(C_{\tilde{W}_i} - p_i).$$
Since $M_{W_i}$ is positive and $K_i$ is positive definite, we know $\dot{V}_1 \leq 0$. For $\dot{V}_2$, plugging in our adaptation law (3.18) for $\dot{w}_i$ yields

$$\dot{V}_2 = \sum_{i=1}^{n} \frac{k_w}{2} \sum_{j \in N_i} \left( (w_i - g(\hat{K}_i)) - (w_j - g(\hat{K}_j)) \right)$$

$$= 0.$$ 

For $\dot{V}_3$, we plug in our estimator (3.20) for $\dot{\hat{K}}_i$, thus

$$\dot{V}_3 = \sum_{i=1}^{n} \text{Tr} \left[ -\tilde{K}_i \Lambda_i(t) \tilde{K}_i^T \right]$$

$$\leq 0.$$ 

From $\dot{V}_1$, $\dot{V}_2$, and $\dot{V}_3$, we see $\dot{V} \leq 0$. Since the trajectories of both $p_i(t)$ and $\tilde{K}_i(t)$ are bounded, $\hat{K}_i$ is also bounded. To determine whether the weightings are bounded, consider the vector $\dot{w}$. Let

$$g_K(t) = \begin{bmatrix} g(\hat{K}_1) \\ \vdots \\ g(\hat{K}_n) \end{bmatrix}.$$ 

Hence we can write the adaptation law in vector form as

$$\dot{w} = -k_w M^{-1} Lw + k_w M^{-1} L g_K(t),$$

(3.24)

where $M^{-1}$ is a diagonal matrix of positive entries and $L$ is the Laplacian of Delaunay graph as defined in Theorem 4. By using the same arguments given in the proof of Theorem 4 based on the input-to-state properties of marginally stable filters and the fact that $g_K(t)$ is bounded and continuous, we can say that $w(t)$ remains bounded. Thus, all trajectories of the system $(p_i(t), \tilde{K}_i(t), \text{and } w_i(t))$ for all $i$ remain bounded.

Since we have already shown that $\dot{V} \leq 0$, to complete the proof we must find the largest invariant set within the set defined by $\dot{V} = 0$. We can see that $\dot{V} = 0$ occurs when $p_i = C_{W_i}$ and $\tilde{K}_i \Lambda_i = 0$. From our control law (3.17) and estimator (3.20), this itself is an invariant set. Therefore, by LaSalle’s Invariance Principle, we can say that the positions of the robots obey

$$p_i(t) \rightarrow C_{W_i}(t) \text{ as } t \rightarrow \infty.$$
and
\[ \tilde{K}_i(t)\Lambda_i(t) \to 0 \text{ as } t \to \infty, \]
proving (3.21) and (3.22) from Theorem 5. \(\square\)

**Corollary 4.** If \(\Lambda_i(t)\) achieves full rank for all \(i \in \{1, \ldots, n\}\) and any \(t > 0\), then
\[ \lim_{t \to \infty} \hat{K}_i(t) = K_i \ \forall \ i \in \{1, \ldots, n\}. \tag{3.25} \]
Furthermore,
\[ \lim_{t \to \infty} (w_i(t) - w_j(t)) = g(K_i) - g(K_j) \tag{3.26} \]
for all \(i, j \in \{1, \ldots, n\}\).

**Proof.** From Theorem 5, we see that \(\tilde{K}_i(t)\) converges to the null space of \(\Lambda_i(t)\). It can be shown that the rank of \(\Lambda_i(t)\) is nondecreasing in time. Thus, if at some \(\tau > 0\) we find \(\Lambda_i(\tau)\) has full rank, then \(\Lambda_i(t)\) has full rank for all \(t > \tau\), and the null space of \(\Lambda_i(t)\) is the set only containing the zero vector. Therefore for \(\tilde{K}_i = (\hat{K}_i - K_i)\), we can write
\[ \lim_{t \to \infty} \tilde{K}_i \Lambda_i = 0 \]
\[ \Rightarrow \lim_{t \to \infty} \hat{K}_i = K_i, \]
proving (3.25). Furthermore, our weightings adaptation law (3.18) can be written in vector form, as shown in (3.24). By the properties of stable linear filters, we know that as the input approaches a limit, the state will also approach a limit [Khalil, 2002], which satisfies the steady-state equation
\[ w \to \{w_\infty|0 = -k_wM^{-1}L(g_{K\infty} - w_\infty)\}, \]
where \(g_{K\infty}\) is the limit of \(g_K(t)\), written as
\[ g_{K\infty} = \begin{bmatrix} g(K_i) \\ \vdots \\ g(K_n) \end{bmatrix}. \]
Solving for \(w_\infty\) we find
\[ k_wM^{-1}Lw_\infty = k_wM^{-1}Lg_{K\infty} \]
\[ Lw_\infty = Lg_{K\infty}. \]
Given that $L$ is the Graph Laplacian, this is equivalent to

$$w_{i,\infty} - w_{j,\infty} = g(K_i) - g(K_j),$$

proving (3.26) from Corollary 4.

\[\square\]

**Remark 4.** In all of our simulations and experiments, we see that $\Lambda_i(t)$ quickly achieves full rank for all $i$. To see why, notice that for $\Lambda_i(t)$ not to achieve full rank, the robot $i$ must move in a precisely straight line throughout its entire trajectory, which is unlikely given the nonlinear nature of the system. However, this fact is difficult to prove rigorously.

**Remark 5.** Using arguments similar to Corollary 2, we can show that the local minima of our Lyapunov-like function (3.23) are also local minima of our coverage cost function (3.3). Using Theorem 5 and Corollary 4, our controller (3.17), weightings adaptation law (3.18), and $\hat{K}_i$ estimator (3.19) provide a solution to Problem 1.

### 3.4.2 Combining Sensing and Actuation Variations

It may be desirable to implement both sensing and actuation adaptation laws during a coverage control deployment. Here, we will outline the setup for simultaneous adaptation to both sensing and actuation variations. For our cost function in (3.3) consider,

$$h_i = h_{i,s} + h_{i,a},$$

where $h_{i,s}$ is the sensor health and $h_{i,a}$ is the actuation performance health. Let $w_{i,s}$ and $w_{i,a}$ to be performance weightings corresponding to the sensing and actuation performance, respectively, such that

$$w_i = w_{i,s} + w_{i,a}.$$

Let $\dot{w}_{i,s}$ to be the adaptation law presented in (3.9) and $\dot{w}_{i,a}$ is the adaptation law presented in (3.18). Following the proof of Theorem 5 and Corollary 4, it can be shown that the robots will converge to their centroids, the sensing performance weightings
satisfy

\[(w_{i,s} - w_{j,s}) \rightarrow (h_{i,s} - h_{i,j}) \forall \ i, j,\]

and the actuation performance weightings satisfy

\[(w_{i,a}(t) - w_{j,a}(t)) \rightarrow (g(K_i) - g(K_j)) \forall \ i, j.\]

Hence, we can combine sensing and actuation variations into a single weighting for the agents to use in implementation.

3.5 Simulations

To demonstrate our move-to-centroid controller (3.8), weightings adaptation laws (3.9) and (3.18), and actuation performance estimator (3.19), we conducted a series of simulations in Matlab. We present three different simulations that illustrate the performance of our algorithm: the first simulation is a sensing-based task, where all robots are initialized with equal weights, but one robot has a lower relative health. The second simulation is also for a sensing-based task, but the sensor healths have been randomized. The third simulation shows an actuation-based task, where the initial weights have been randomized and there is one higher-performing robot and one lower-performing robot. For the actuation-based scenarios, we chose

\[g(\hat{K}_i) = \|\hat{K}_i\|\]

to measure our actuation performance.

3.5.1 Sensing Example Simulation

We use Matlab to simulate the sensing-based weightings adaptation law within our coverage control algorithm for \(n = 8\) robots. A uniform information density function \(\phi(q)\) is used. Weightings are initialized to \(w_i = 1\) for all \(i\), and the sensing health
is $h_i = 1$ for all $i$, except for robot 2, which has a health of $h_2 = 0.1$. Figure 3.5 compares the initial and final configurations of the robots.

Figure 3.6 shows the cost function (3.3) over time. We see that the cost decreases until it reaches a minimum value, which corresponds to the group reaching the centroidal Voronoi configuration. Figure 3.7 shows the true value of the performance weights over time, and the convergence of the relative difference $(w_i - h_i)$ to a common value, with robot 2 shown in red. We see that robot 2’s weight decreases over time, which is expected given its lower sensor health. However, the difference $(w_i - h_i)$ reaches a common value across all agents, as predicted by Theorem 4.
Figure 3.5: The (a) initial and (b) final configurations of the robots during a sensing-based task. Here, the cell shading corresponds to the relative performance weight. In the initial configuration, all robots have equal weights, but by the final configuration, robot 2 has the lowest relative weight, as expected.
Figure 3.6: Sensing cost (3.3) over time. The cost decreases to a minimum value, indicating the group reached a locally optimal configuration.
Figure 3-7: (a) Values of the performance weights over time, with robot 2 in red. As predicted, robot 2 attains a lower relative weight than the rest of the group. (b) The difference \((w_i - h_i)\) over time, with robot 2 in red. For the group, this difference converges to a common value as predicted by Theorem 4.
3.5.2 Sensing Example with Randomized Health

This sensing-based task simulation was also performed in Matlab to demonstrate our controller performance in a randomized configuration. We use a uniform information density function $\phi(q)$ and $n = 30$ agents. In this simulation, the weightings were initialized to be $w_i = 1$ for all $i$, and sensing health factors were initialized as random numbers drawn from the uniform distribution over $[0, 1]$.

Figure 3-8 compares the initial and final configurations of the robots. Figure 3-9 shows the cost function (3.3) as well as the difference $(w_i - h_i)$ over time. From Figure 3-8, we see the algorithm reaches a centroidal Voronoi configuration from randomized initial positions. Note that in the initial configuration, all weights are equal, so the shading is the same (white) for all cells. However, by the end of the simulation, the shading reflects the various differences in performance weights to match the variations in health. Robots with the highest relative health have the strongest shade of green, while the lowest relative health is the strongest red. We also observe in Figure 3-9 that despite the weights diverging to unique values, the weightings converge to the invariant set of $(w_i - h_i) = (w_j - h_j)$. 
Figure 3-8: The (a) initial and (b) final configurations. The initial configuration begins with all robots at randomized locations, with equal initial weights, shown by the lack of cell shading. By the final configuration, the robots have reached a more balanced configuration, and the weights have adapted to the relative health differences. Lower weights are shown in red, and higher weights are shown in green.
Figure 3.9: (a) Cost (3.3) over time. As the robots spread out over the environment, they minimize the sensing cost before settling into a final, locally optimal configuration. (b) The difference ($w_i - h_i$) for each robot. As expected, this relative difference converges to a common value across the group.
3.5.3 Actuation Example Simulation

To demonstrate our adaptation law and parameter estimator, we simulated an actuation example in Matlab. We use a uniform information density function and $n = 10$ agents with randomized initial weighting values drawn from the uniform distribution over $[0, 1]$, as noted below. Robots were assigned $K_i = I$, except robots 1 and 6, whose values are given below.

$$w(0) = [0.2, 0.7, 0.4, 0.5, 0.1, 0.5, 1.0, 0.2, 0.4, 0.6],$$

$$K_1 = \begin{bmatrix} 0.05 & 0 \\ 0 & 0.05 \end{bmatrix}, \quad K_6 = \begin{bmatrix} 1.65 & 0 \\ 0 & 1.65 \end{bmatrix}.$$

Figure 3.10 shows the initial and final configurations of the agents, with their relative performance weights given by the cell shading. Initially, the robots are assigned random weights, but by the end of the simulation, robot 1 has the lowest weight, and robot 6 has the highest weight, as expected. Figure 3.11 shows the global cost and convergence of $w_i - \|\hat{K}_i\|$. We see that the cost over time decreases to a minimum, corresponding to the the group finding a locally optimal centroidal configuration. As expected, we see that the values of $w_i - \|\hat{K}_i\|$ are equal across all agents. Figure 3.12 shows the values of our $\hat{K}_i$ estimator over time.

Although robots 1 and 6 do not initially know their $K_i$ matrix, using our online estimator they are able to successfully determine the correct values. Since their estimate of performance converges, the conditions of Corollary 4 are satisfied, and we know that the performance weightings will converge to $(w_i - \|\hat{K}_i\|)$. 
Figure 3.10: The initial (a) and final (b) configurations, with the cell shading corresponding to the relative performance weight. The initial configuration has the robots starting at randomized locations with varying initial performance weights. By the final configuration, the highest performing robot 6 has the largest cell, and the lowest performing robot 1 has the smallest cell, with its relative weight indicated by the shading.
Figure 3.11: (a) Cost (3.3) over time. We see the cost decreases to a minimum, demonstrating the group reaches a locally optimal configuration. (b) The relative difference ($w_i - \|\hat{K}_i\|$), with Robot 1 shown in red, and Robot 6 in green. As predicted, the group reaches a common value.
3.6 Experiments

In order to verify the behavior of our controller, we implemented our algorithm using m3pi\(^2\) robots equipped with XBee\(^3\) radios. Pose data was calculated using an OptiTrack\(^4\) system. The experiments were run to parallel the scenarios presented in our Simulation section. Videos of the experiments can be found on the Multi-robot Systems Lab website\(^5\). For our weightings adaptation law, all experiments use the performance function \(g(\hat{K}_i) = \|\hat{K}_i\|\).

The m3pi robot is a small differential drive robot from Pololu Robotics, shown in Figure 3-13. It utilizes an onboard mbed microcontroller to handle actuation and communication. The mbed controls the motors on the robot, and we send velocity

---
\(^2\)Pololu’s m3pi: www.pololu.com/product/2151  
\(^3\)Digi’s XBee: www.digi.com/xbee/  
\(^4\)Natural Point OptiTrack: www.naturalpoint.com/optitrack/  
\(^5\)Experimental videos: http://msl.stanford.edu
data via XBee radios to the robots based on the Voronoi calculations in Matlab. To localize our robots, we used NaturalPoint’s OptiTrack system with sixteen IR cameras. Short-throw projectors were used to display the centroid and Voronoi boundaries on the floor mats during the experiments, also shown in Figure 3-13.

![Figure 3-13](image)

**Figure 3-13:** (a) The m3pi robots used in the experiments. Each robot is equipped with an mbed processor to handle the low-level control. Communication with Matlab is done via an XBee radio, and the silver reflective markers are used for identification in our OptiTrack system. (b) The Voronoi cells are projected onto the floor during each experiment.

Given the m3pi robots are nonholonomic vehicles, we also incorporated a low-level point-offset controller [Michael and Kumar, 2009] to account for the dynamics. Here, instead of driving the robot to the centroid, we will drive the point-offset of the robot to the centroid. In the experiment video stills, the point-offset is plotted with a circle, and the centroid of the Voronoi cells are plotted with a ‘+’ symbol.
3.6.1 Sensing Example

This experiment demonstrates the performance of the controllers presented in Section 3.3. Here, our environment has a uniform information density function $\phi(q)$. Seven robots were initialized with different weight and health values, assigned as

$$h = [1.0 \ 0.3 \ 1.0 \ 1.0 \ 1.0 \ 1.0 \ 1.0],$$

$$w(0) = [0.6 \ 0.8 \ 0.4 \ 0.1 \ 0.7 \ 0.3 \ 0.5].$$

For this example, the health value $h$ simulates the sensor quality, given in (3.5).

Figure 3.14 shows the initial and final configurations of the agents over the course of the experiment. Each cell is shaded to indicate its relative weight, with green indicating a higher relative weight, and red indicating a lower relative weight. To assess the performance, we can also examine a plot of the cost and $(w_i - h_i)$ given in Figure 3.15. Similar to the simulations, we see the cost decreasing to a minimum value, indicating the robots reach a locally optimal configuration. The jumps present in the cost plot are due to noise and other typical sources of error in experimental hardware. We also see that over time, the difference $(w_i - h_i)$ converges to the same value across all robots, as predicted by Theorem 4.
Figure 3.14: In the initial configuration (a), the robots have been assigned random initial weights. We see Robot 2 has the largest performance weight, despite it actually having the lowest health. In the final configuration (b), we see that the group has compensated, and now Robot 2 has the lowest relative weight, indicated by the red shading.
Figure 3.15: (a) Cost (3.3) over time. We see the cost decreases to a minimum value, indicating the group achieves a locally optimal final configuration. Note that minor jumps are due to noise present in experimental hardware. (b) The value of \((w_i - h_i)\), with Robot 2 indicated in red. Over time, the group reaches a consensus on the difference between the weights and the health.
3.6.2 Sensing Example with Noisy Images

This experiment demonstrates the performance of our sensing adaptation law by introducing simulated noisy cameras for each robot. The robots must then compare properties of their image with their neighbors to determine relative performance. Here, each robot is given a noisy image of a forest environment, with the noise generated based on their location and sensor health. By comparing sections of their image along the Voronoi boundaries with their neighbors, they are able to adapt to the performance variations within the group. Consider $H_{i,q}$ to be the random amount of noise seen by sensor $p_i$ looking at point $q$, generated as

$$H_{i,q} = \alpha \left( ||p_i - q||^2 - h_i \right) X,$$

where $h_i$ is the sensor health, $\alpha$ is a scaling factor and $X$ is a random number drawn from a uniform distribution. Note that $H_{i,q}$ is similar to the sensor quality function $\gamma_i(p_i, q, h_i)$ in (3.5). We choose this model for $H_{i,q}$ to be random noise consistent with passive sensor quality loss [Goldsmith, 2005], discussed in Section 3.2.2. Here, we simulate a noisy camera for visual feedback, but this model applies to other passive sensors, such as microphones, IR cameras, or RF listening devices. Figure 3-16 illustrates an example of the noise seen by a robot, shown in red for clarity.
Although the robots do not know how noisy their sensor is, the variance provides a relative level of noise between neighbors looking at the same region. Using local statistics, such as variance, is common in noise-modeling for image processing [Lee and Hoppel, 1989, Liu et al., 2006]. A higher variance indicates a higher presence of noise, whereas a lower variance indicates lower levels of noise. By computing the pixel variance over a small patch centered on their boundary and comparing it with their neighbors, the robots can determine a relative noise level in their group. Let \( Q_{ij} \) be a small, shared patch of the image around the shared boundary \( b_{ij} \) of neighbors \( i \) and \( j \). We calculate the weightings adaptation law as

\[
\dot{w}_i = \sum_{j \in \mathcal{N}_i} \frac{k_w}{M_{W_i}} \left( -\text{Var}_i(Q_{ij}) + \text{Var}_j(Q_{ij}) \right),
\]

where \( \text{Var}_i(Q_{ij}) \) is the empirical variance computed by robot \( i \) over the shared region \( Q_{ij} \) and \( M_{W_i} \) is the mass of the weighted Voronoi cell. For this experiment, the environment has a uniform information density function \( \phi(q) \). Multiple trials were conducted with \( n = 3 \) and \( n = 4 \) robots. In the first trial, three robots were initialized...
with equal weights but different health values, assigned as

\[ h = [-0.1, -0.7, -0.1] \times 10^5 \]
\[ w = [1.0, 1.0, 1.0] \times 10^5. \]

In the second trial, four robots were initialized with equal initial weights but different health values, assigned

\[ h = [-0.1, -0.8, -0.1, -0.1] \times 10^5 \]
\[ w = [1.0, 1.0, 1.0, 1.0] \times 10^5. \]

Figure 3·17 shows the initial and final configurations of the agents over the two trials using the forest image. In the initial configuration, the large amount of red shading indicates a large amount of noise. By the final configuration, the overall image is less noisy, and the agent with the lowest health has the smallest cell.

Figure 3·18 shows the difference \((w_i - h_i)\) converge to the same value over time, as predicted by Theorem 4. The minor fluctuations in value are due to the random noise added in the image comparison, but despite this noise, the robots still find a common value. To further assess the performance, we can examine a plot of the cost over time in Figure 3·19. Here, we have compiled the cost over multiple trials. As in the simulations, the cost decreases to a minimum value, indicating the robots reach a locally optimal configuration. In our experimental videos found on the MSL website, one can see the noise decrease dynamically as the agents move towards the centroids of their Voronoi cells and improve their configuration.

\footnote{Here, the health and weight are multiplied by \(10^5\) due to the scale of the environment}
Figure 3.17: Initial and final configurations for two trials. Initially, there is lots of noise present, shown in red on (a) and (c). By the final configurations, the group has compensated for noisy robots, as seen by the decrease in red in (b) and (d).
Figure 3.18: The value of \((w_i - h_i)\) over time shown for (a) \(n = 3\) and (b) \(n = 4\) robots. As expected, the robots converge to a common value.
Figure 3.19: Cost over multiple trials with $n = 3$ and $n = 4$ robots with random initial configurations. Over time, the cost decreases to a minimum, indicating convergence to a locally optimal configuration. Note that each initial configuration will yield a different locally optimal solution, and a different minimum cost value.

3.6.3 Actuation Example

In this experiment, we used six robots and set $\phi(q)$ to be a constant information density function over the environment. All robots were initialized with weights of $w_i = 1$. Robots were given actuator performance matrices $K_i = I$, except robot 1, which was assigned $K_1 = [0.6, 0; 0, 0.6]$, and robot 6, which was assigned $K_6 = [1.2, 0; 0, 1.2]$. Figure 3.20 shows the initial and final configurations of the agents. In Figure 3.21a we can see that the cost function decreases over time, which means that the group converged to a locally optimal configuration. We can also see the successful convergence of our estimator $\hat{K}_i$ to $K_i$ in Figure 3.21b. A plot of the weightings over time in Figure 3.22a shows that the lowest performing agent 1, shown in red, has the lowest weighting over time. Figure 3.22b shows that the difference $(w_i - \|\hat{K}_i\|)$ converges to a common value.
Figure 3.20: In the initial configuration (a), note that all robots start off with equal weights. Robot 1 has the lowest health, but a relatively large cell, while Robot 6 has a high health, but small cell. In the final configuration (b), the shading indicates the relative weights. Note that Robot 1’s cell is now much smaller, and the red indicates its lower relative weight. Similarly, Robot 6’s cell has increased in size, and the green indicates a relatively higher performance weight.
Figure 3.21: (a) Cost (3.3) decreases to a minimum value, indicating the group reaches a static, locally optimal configuration. Minor jumps are due to noise and error present in the experimental hardware. (b) The $\hat{K}_i$ estimator versus the true value $K_i$. For agents 6 (green) and 1 (red), we see the estimator successfully converges to the true value over the experiment.
Figure 3.22: (a) True value of the weights over time. Note agent 6 (green) has a higher weight, corresponding to its better performance, and agent 1 (red) has a lower health, corresponding to its weaker performance. (b) The difference $(w_i - \|\hat{K}_i\|)$ over time, which converges to a common value across all agents.
3.7 Summary

This chapter presents a method of using adaptive weightings to adjust for individual variations in performance within multi-robot coverage control. We consider both errors due to sensing variations, and errors due to variation in actuation abilities. To account for these errors, the robots compare values of an error estimate with their neighbors, and using an adaptive weighting law, adjust the value of their weightings online. By controlling these weights, we are able to modify the Voronoi boundaries between neighboring robots, which adjusts a robot’s cell size relative to its neighbors. The weightings adaptation law and error estimation occur online within the coverage control algorithm. We demonstrate the algorithm in both simulation and experiments using m3pi robots.

Our method incorporates performance error into the decentralized algorithm while maintaining stability and performance. This can provide an additional level of robustness in real-world applications when the robots are in an unknown environment and may have varying capabilities across the team. It can also provide insight into identifying failures of a team member. In this chapter, we only consider robots with variations in performance, but they are not malicious or manipulative.
Chapter 4

Controlling Non-Cooperative Herds with Robotic Herders

In this chapter, we present our problem formulation of non-cooperative bio-inspired multi-agent herding, previously published in [Pierson and Schwager, 2015], with a journal version under review for Transactions on Robotics (T-RO) [Pierson and Schwager, 2016]. This represents a system that is non-cooperative, but there are no malicious agents. Here, our goal is to have multiple “dog” herders manipulate a herd of one or more “sheep” agents in an environment, either relocating them to a goal region or tracking some trajectory.

We model the sheep agents as being repulsed from the herders with a potential field, common in biological models. While the system is highly nonlinear, we design controllers that place all the herders on a radius around the herd. By enforcing this constraint, the dynamics of the system reduce to a single nonholonomic vehicle. Once this kinematic reduction occurs, we can design controllers for the nonholonomic vehicle and map those controls back to the individual herder controls. Here, we choose to employ a point-offset control strategy, which allows us to drive the herd to a goal region in the environment.

The remainder of the chapter is organized as follows. Section 4.1 presents related work to this herding problem. In Section 4.2 we present our mathematical formulation of the problem. Section 4.2.3 builds the 2D kinematic models for the various numbers of dogs and sheep and describes the reduction to a unicycle robot. We propose a two...
part control strategy in Section 4.2.4. Section 4.3 presents the extensions of the model in 3D. Simulation results are presented in Sections 4.5 and 4.6 for 2D and 3D respectively, with 2D experimental results in Section 4.7. We give our summary in Section 4.8.

4.1 Related Work

There has been surprisingly limited prior work on non-cooperative robotic herding. One exception is Vaughan’s pioneering work [Vaughan, 1999, Vaughan et al., 2000], in which a single robot is used to herd ducks in a specially designed experimental arena. In Vaughan’s work, the robot communicates with a centralized computer vision system to determine the mean of the duck herd. From there, the robot drives towards the mean as to move the herd to a goal. More recently, Lien et al. developed a set of behavior primitives for controlling a flock with multiple shepherds [Lien et al., 2005]. The herders are placed at a set of “steering points” around the flock, and choose their behavior primitive based on the herd and environmental properties. In contrast to both of these, our work takes a control theoretic approach to design feedback laws for an arbitrary number of dogs to drive an arbitrary number of sheep. Other authors have formulated the problem as a dynamic pursuit-evasion game to find optimal trajectories that allow the herder to drive the sheep to some goal position [Shedied, 2002, Lu, 2006]. In this work, the herder “catches” the sheep at the goal location, whereas in our setting the herders relocate the herd without the intent to “catch” it. Furthermore, in the area of multi-agent formation control, researchers have considered driving robots into a desired formation [Egerstedt and Hu, 2001, Ferrari-Trecate et al., 2006]. In this setting, the robots typically have linear dynamics, and have cooperative control laws that are intended to move them into a formation. In contrast, our herd of sheep are non-cooperative, and have a nonlinear response to the dogs.
To model the herd dynamics, we use potential fields, which is common in animal aggregation modeling for schools of fish [Breder, 1954], birds, slime molds, mammal herds, and other swarms [Gazi and Passino, 2004, Toner and Tu, 1998]. These models have been applied to multi-agent systems to simulate flocking [Reynolds, 1987], cooperative group control [Howard et al., 2002, Tanner et al., 2003, Tanner et al., 2007], and interaction with collision avoidance [Olfati-Saber, 2006]. We use a potential field model for our theoretical analysis, and provide simulation results with noise and flocking dynamics.

Our work proposes a reduction from the nonlinear dog-sheep system to well-known nonholonomic vehicle models. In 2D, our system reduces to a unicycle model for a differential drive robot [DeVon and Bretl, 2007, Murray et al., 1994]. This introduces a nonholonomic constraint, which limits the robot to only translate in the direction of its heading. Several techniques to drive the unicycle robot to the origin without violating Brockett’s Theorem [Brockett, 1983] include optimal control [Cameron and Book, 1994, Laumond et al., 1998], sliding mode control [Chwa, 2004], or Lyapunov-like functions [Aicardi et al., 1995]. Our chosen strategy is to control a point that is offset from the center of mass of the robot, whose dynamics then become holonomic [Michael and Kumar, 2009]. We call this control strategy a point-offset controller. Similarly, in 3D we show that our system reduces to a common nonholonomic vehicle model used in underwater autonomous vehicle modeling [Nakamura and Savant, 1991, Aicardi et al., 2000, Aicardi et al., 2001, Egeland et al., 1994, Canudas de Wit and Sordalen, 1991] and aerial vehicles [Roussos et al., 2008, Roussos et al., 2010, Ambrosino et al., 2006]. We derive a 3D extension of the point-offset controller, which to our knowledge is not present in prior literature. By designing feedback controllers for the point offset, we obtain nonlinear feedback controllers for the herders, which in turn drive the herd to a goal region in the environment. A preliminary version of the 2D problem appeared in
our conference publication [Pierson and Schwager, 2015]. Here we include additional experimental results for the 2D problem, the extension to 3D, and a study on its robustness to noise. A journal version of this work is also under preparation [Pierson and Schwager, 2016].

4.2 2D Problem Formulation and Controller Design

Consider $m$ herders (or “dogs”) with positions $d_j \in \mathbb{R}^2$, where $j \in \{1, \ldots, m\}$, and $n$ herd members (or “sheep”) with positions $s_i \in \mathbb{R}^2$, where $i \in \{1, \ldots, n\}$. The “dogs” in this model are presumed to be robots since they are under our control, while the herd members can be robots, sheep, cattle, other herding animals, or even humans. However, for the purposes of this chapter we will use the shepherding analogy throughout. We assume the dogs have integrator dynamics,

$$\dot{d}_j = u_j. \quad (4.1)$$

Here, $u_j$ is the control input moving $d_j$ through the environment. Our main goal is to design $u_j$ such that the dogs drive the sheep to some goal region. We model the sheep’s repulsion from the dogs using an artificial potential field [Howard et al., 2002], which is common in robotics and in models of biological herding animals. Let $d = [d_1 \ldots d_m]$ denote the group of herders and $s = [s_1 \ldots s_n]$ denote the group of sheep. Using the potential field $W(d, s) = \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{1}{\|d_j - s_i\|}$, we obtain the sheep’s dynamics as

$$\dot{s}_i = \frac{\partial W}{\partial s_i} = \sum_{j=1}^{m} \frac{-(d_j - s_i)}{\|d_j - s_i\|^3}. \quad (4.2)$$

For now we do not consider the additional forces from flocking dynamics between members of the herd, although this will be introduced later in Section 4.2.3.
Also consider a user-defined goal region

\[ B_\ell(g) = \{ q \in \mathbb{R}^2 \mid \|q - g\| \leq \ell \} \]

centered at a goal point \( g \in \mathbb{R}^2 \) with a desired radius \( \ell > 0 \). This goal region represents the set of allowable final configurations for the sheep to occupy. Without loss of generality, we can define our coordinate frame to be centered at the goal point, so that \( g = 0 \). We take the goal point to be the origin through the rest of the chapter.

**Problem 2. (Multi-Agent Herding)** Given the dynamics of the herd (4.2), find control laws \( u_j = f(d, s) \) for \( d_j \) herdres with dynamics (4.1) to relocate the herd from arbitrary initial conditions to the desired region in the environment \( B_\ell(g) \).

We propose a solution to Problem 2 that is both simple and scalable to \( m \) herdres. The key insight of our approach lies in enforcing geometrical relationships that map the complex, nonlinear dog and sheep dynamics to a simple unicycle model. This creates an ideal unicycle-like system which we utilize in our controller design. We first introduce terminology and basic nomenclature to describe the unicycle model, then present our herding models that reduce to the unicycle-like system.

### 4.2.1 Modeling of a 2D Unicycle Vehicle

Consider the unicycle-like vehicle shown in Figure 4.1. For a unicycle-like vehicle with position \( s \), we define a local reference frame \( B \) relative to the global frame \( A \),
with the rotation matrix from the $B$ frame to the $A$ frame written

$$A^B R = \begin{bmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{bmatrix}. $$

We express the $B$ frame in global coordinates as

$$b_x = A^B R \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad b_y = A^B R \begin{bmatrix} 0 \\ 1 \end{bmatrix}. $$

The vehicle moves with forward velocity $v$ in the local $b_x$ direction, and angular velocity $\omega = \dot{\psi}$, as shown on the right in Figure 4·1. Overall, the kinematic constraints are written

$$\dot{s} = A^B R \begin{bmatrix} v \\ 0 \end{bmatrix}, $$
$$\dot{\psi} = \omega. \quad (4.3) $$

### 4.2.2 Point-Offset Control

Consider a point offset $p$ a distance $\ell$ from $s$, written

$$p = s + A^B R \begin{bmatrix} \ell \\ 0 \end{bmatrix}. \quad (4.4) $$

While the unicycle-like vehicle has nonholonomic dynamics, it turns out that $p$ is holonomic [Michael and Kumar, 2009], which allows us to design controllers for $p$ then transform them back to the vehicle dynamics $v$ and $\omega$. The derivative of $p$ is

$$\dot{p} = \dot{s} + \frac{d}{dt} A^B R \begin{bmatrix} \ell \\ 0 \end{bmatrix}. $$

It can be shown that the derivative of a rotation matrix reduces to [Murray et al., 1994]

$$\frac{d}{dt} [A^B R] = A^B R \Omega, \quad \Omega = \begin{bmatrix} 0 & -\omega \\ \omega & 0 \end{bmatrix}, $$
where $\Omega$ is the skew-symmetric matrix of the local angular velocities. Thus, $\dot{p}$ becomes

$$\dot{p} = ^A R^B \begin{bmatrix} v \\ \ell \omega \end{bmatrix}.$$  (4.5)

To then solve for $v$ and $\omega$, we rearrange as

$$\begin{bmatrix} v \\ \ell \omega \end{bmatrix} = ^B R^A \dot{p},$$  (4.6)

where $^B R^A$ is the rotation matrix from the local frame $B$ to the global frame $A$, calculated $^B R^A = (^A R^B)^T$. Written in terms of the local basis vectors $b_x$ and $b_y$, (4.6) becomes

$$v = b_x^T \dot{p},$$

$$w = \dot{\psi} = \frac{1}{\ell} b_y^T \dot{p}.$$  (4.7)

This relationship allows us to find some desired control $\dot{p} = u$ and map it back to the vehicle controls $v$ and $\omega$.

### 4.2.3 Kinematic Reduction

Instead of allowing the herders to occupy any point in the environment, consider the case where all herders are a fixed distance $r$ from the sheep. In this section, we show that under this constraint, the system dynamics reduce to a unicycle-like vehicle. First, we introduce basic concepts in our kinematic model with a single sheep and two dogs, then generalize to any $m$ dogs. We also present extensions to $n$ sheep and $m$ dogs.

**Single-Sheep Model with Two Dogs**

We begin with the case of $n = 1$ sheep and $m = 2$ dogs, shown in Figure 4-2.
Figure 4.2: Configuration of two dogs and a single sheep

Figure 4.2 illustrates the configuration where both dogs are located some distance $r$ from the sheep. The position of the dogs $d_j$ can be written in terms of their angular orientation $\alpha_j$ relative to the sheep as

$$d_j = s + r \begin{bmatrix} \cos(\alpha_j) \\ \sin(\alpha_j) \end{bmatrix}. \quad (4.8)$$

Furthermore, the dynamics of the herd introduced in (4.2) simplify as

$$\dot{s} = -\frac{1}{r^2} \left[ \sum \cos(\alpha_j) \right] \left[ \sum \sin(\alpha_j) \right]. \quad (4.9)$$

To maintain this kinematic relationship, the dynamics of the dogs in (4.1) must take the form

$$\dot{d}_j = \dot{s} + r \dot{\alpha}_j \begin{bmatrix} -\sin(\alpha_j) \\ \cos(\alpha_j) \end{bmatrix}. \quad (4.10)$$

Similar to our unicycle model, we define the heading $\psi$ as the direction of $\dot{s}$ relative to the base frame, where

$$\psi = \frac{1}{2} (\alpha_1 + \alpha_2) + \pi.$$  

We also see that $d_1$ and $d_2$ are always symmetric around the line formed by $\dot{s}$. Consider the angular separation between the two herders as $\Delta = \alpha_2 - \alpha_1$. Thus, we re-write
the angles in terms of $\psi$ and $\Delta$ as

$$
\alpha_1 = \psi + \pi - \frac{\Delta}{2}, \quad \alpha_2 = \psi + \pi + \frac{\Delta}{2}.
$$ (4.11)

These simplifications of the angle in (4.11) allow us to distill the complex dynamics of the herders into two main state variables, $\psi$ and $\Delta$, which makes it much simpler to describe the dynamics when considering $m$ dogs.

**Single Sheep with $m$ Dogs**

To generalize to $m$ dogs, we assume equal spacing of the dogs between $d_1$ and $d_m$ along the desired radius, as shown in Figure 4-3. Thus, $\Delta$ becomes the total separation between the first dog $d_1$ and last dog $d_m$.

![Figure 4-3: Configuration of $m$ dogs and one sheep.](image-url)

The angular orientation of each dog with respect to $\psi$ is written

$$
\alpha_j = \psi + \pi + \Delta_j,
$$ (4.12)

where

$$
\Delta_j = \Delta \frac{(2j - m - 1)}{(2m - 2)}.
$$
Substituting (4.12) into (4.9), the sheep dynamics become

\[ \dot{s} = \frac{-1}{r^2} \left[ \sum \cos(\alpha_j) \right] = \frac{-\sin \left( \frac{2m\Delta}{2-2m} \right)}{r^2 \sin \left( \frac{\Delta}{2-2m} \right)} \left[ \begin{array}{c} \cos(\psi) \\ \sin(\psi) \end{array} \right], \quad (4.13) \]

which allows us to describe the dynamics of the sheep using only the two state variables, \( \psi \) and \( \Delta \), despite having \( m \) dogs. Similarly, by substituting (4.12) in (4.10), the dynamics for the dogs become

\[ \dot{d}_j = \dot{s} + r \left( \psi + \Delta_j \right) \left[ \begin{array}{c} \sin(\psi + \Delta_j) \\ -\cos(\psi + \Delta_j) \end{array} \right]. \quad (4.14) \]

By defining the orientation of the dogs in terms of \( \psi \) and \( \Delta \) along some radius in (4.12) and restricting the dogs’ kinematics to obey (4.14), we can map these quantities to the angular and linear velocity of a unicycle-like vehicle.

**Remark 6.** Note that this model assumes the dogs are fixed on some circle of radius \( r \) relative to the herd, which limits the initial configurations of the dogs relative to the sheep. Later, we introduce a tracking controller for the dogs that allows them to start anywhere in the environment and converge upon this configuration. We also present a radial controller in (4.20) to adjust the radius used by the dogs online when controlling multiple sheep.

**Proposition 1.** The herding dynamics in (4.9) and (4.14) can be reduced to an equivalent unicycle model with forward velocity \( v \) and orientation \( \psi \), described by (4.3).

**Proof.** We can solve this mapping by equating (4.13) to (4.3) and solving for \( v \). To see this mapping, note that (4.13) written in terms of basis vectors becomes

\[ \dot{s} = \left( \frac{-\sin \left( \frac{m\Delta}{2-2m} \right)}{r^2 \sin \left( \frac{\Delta}{2-2m} \right)} \right) b_x. \]

We see that the direction of the herd’s velocity is only in the local-x direction, similar to the unicycle model in (4.3). The direction in global coordinates is defined by \( \psi \).
The magnitude of the velocity is

\[ v = \| \dot{s} \| = \frac{\sin \left( \frac{\Delta}{r} \right)}{r^2 \sin \left( \frac{\Delta}{2} \right)}. \]  

(4.15)

Note that for (4.15), there exist an infinite number of possible values of \( \Delta \) for a given value of \( v \). However, over the range of \( \Delta = (0, 2\pi) \), this mapping is one-to-one. Thus, for a given velocity, we can find the corresponding \( \Delta \).

It remains to map \( \dot{\psi} \) in the herder’s dynamics (4.14). We directly map \( \dot{\psi} = \omega \) from the unicycle dynamics.

Ultimately, we use Proposition 1 in our controller design of the system. Instead of trying to determine individual controllers for all of the dogs, we instead design controllers for the ideal unicycle-like system. Based on the idealized system, we find controllers for the dogs that enforce this behavior.

4.2.4 Controller Design

Section 4.2.3 introduced geometric constraints on the system, which allow us to map the kinematics of the herding system to a unicycle-like vehicle. Our goal, as stated in Problem 2, is to drive the herd to some ball around the origin. To control this system, we propose a controller that drives a point-offset of the ideal unicycle-like system to the origin. Given the velocity and angular velocity controls of the ideal system, we then calculate the positions for the dogs along the circumference of the circle. Although we choose a point-offset controller, other techniques can be applied to the same kinematic mapping.

Consider the desired position for each dog, which lies on a circle of radius \( r \) around the sheep with spacing \( \Delta_j \) between dogs. We assume the dogs are able to perfectly track their desired positions, a common assumption in multi-robot literature [Egerstedt and Hu, 2001, Mesbahi and Egerstedt, 2010]. Let the ideal orientation, \( \psi^* \), be the angle that points the herd’s velocity towards the origin. To find the ideal
velocity for the unicycle-like system, we find a controller for a point-offset \( p \) from the sheep that drives the point offset to the origin. While there exist many possible choices for controlling the point-offset \( p \), we opt for a simple proportional feedback controller,

\[
\dot{p} = -kp. \tag{4.16}
\]

Plugging this into our mapping in (4.7), we find the ideal velocity becomes

\[
v^* = [\cos(\psi^*) \sin(\psi^*)](-kp).
\]

Using the mapping in (4.15), we then determine the desired separation \( \Delta_j^* \) for the dogs. Overall, this yields the desired position of the dogs, written as

\[
d_j = s + r \left[ \begin{array}{c} \cos(\psi^* + \Delta_j^*) \\ -\sin(\psi^* + \Delta_j^*) \end{array} \right]. \tag{4.17}
\]

We can now analyze the system as if it were the simple unicycle-like system, and are ready to state our main proposition on the behavior of a single sheep and \( m \) dogs.

**Proposition 2.** For the single sheep, \( m \) dog system described in (4.13) and (4.14), using the controllers

\[
v = -kb_x^T (s + \ell b_x) \\
w = \dot{\psi} = -\frac{k}{\ell} b_y^T (s + \ell b_x) \tag{4.18}
\]

the herd converges to the ball of radius \( l \) about the origin, \( B_l \).

**Proof.** It is equivalent to say that if the point offset converges to the origin, the herd converges to the ball \( B_l \) about the origin. Consider the Lyapunov candidate function

\[
V = \frac{1}{2} \left( s + A_R^B \begin{bmatrix} \ell \\ 0 \end{bmatrix} \right)^T \left( s + A_R^B \begin{bmatrix} \ell \\ 0 \end{bmatrix} \right).
\]

From (4.4), this is equivalent to

\[
V = \frac{1}{2} p^T p,
\]
with derivative
\[ \dot{V} = p^T \dot{p} \]

Substituting our expressions for \( \dot{p} \) (4.5), this becomes
\[ \dot{V} = p^T (v b_x + \ell \dot{\psi} b_y). \]

Plugging in the expressions for \( v \) and \( \dot{\psi} \) chosen in (4.18), this becomes
\[ \dot{V} = p^T (-k (b_x^T p) b_x + -k (b_y^T p) b_y)
= -k \|p\|^2 < 0. \]

By Lyapunov’s direct method [Khalil, 2002], the equilibrium point \( p^* = 0 \) is asymptotically stable. Furthermore, by the form of \( \dot{V} \), we see it is exponentially stable. When \( p = 0 \), the sheep are at most a distance \( \ell \) away from the origin, thus proving Proposition 2.

Proposition 2 proved that \( m \) dogs can relocate a single sheep to a goal region when the dogs are spaced on a circle around the herd. We present two modifications to our control strategy. First, we extend our controllers to a multi-sheep model and introduce a radial controller to compensate for changes in the herd. Next, we introduce a tracking controller for the dogs, which allows them to start from any point in the environment. These additional modifications are summarized in our overall control algorithm, presented in Algorithm 1.

**Multi-Sheep Model**

Here, we extend our control design to the case of \( m \) dogs and \( n \) sheep. We use the herd’s mean, \( \bar{s} \), in the dogs’ controller and introduce a radial controller to account for changes in the herd’s footprint. The radius \( r \) is defined from the mean of the herd, \( \bar{s} \), as shown in Figure 4.4.
From (4.2), the dynamics of the herd mean are

$$\dot{s} = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{-(d_j - s_i)}{\|d_j - s_i\|^3}.$$  \hspace{1cm} (4.19)

Due to the varying nature of the extent of the herd, we also introduce an additional term $\dot{r}$ in the controller to regulate the radius $r$. The dynamics are written

$$\dot{d}_j = \dot{s} + r \dot{\alpha}_j \begin{bmatrix} -\sin(\alpha_j) \\ \cos(\alpha_j) \end{bmatrix} + \dot{r} \begin{bmatrix} \cos(\alpha_j) \\ \sin(\alpha_j) \end{bmatrix}.$$  \hspace{1cm} (4.20)

We design the radius controller $\dot{r}$ to maintain some desired radius $r_0$, as well as adjust for the standard deviation of the herd. Our proposed controller is

$$\dot{r} = (r_0 - r) + \frac{1}{n} \sum_{i=1}^{n} 2(s_i - \bar{s})^T (\dot{s}_i - \dot{\bar{s}}).$$  \hspace{1cm} (4.20)

where $r_0$ is the desired radius if the herd were a single sheep. This radial controller assumes that the dogs initially encircle all sheep and can always remain outside the convex hull of the sheep over time. By incorporating this radius controller, our control algorithm allows for fluctuations in the extent of the herd.
Tracking Control

For Proposition 2, we enforce that the dogs remain on a circle of radius $r$ around the herd. In this section, we discuss the addition of a tracking controller that drives the dogs to their ideal positions. This allows the dogs to start from any location in the environment, and is written

$$
\dot{d}_j = -K_d (d^*_j - d_j),
$$

where $d^*_j$ is the desired location expressed in (4.17). Under the following mild assumption, we can analyze the performance of this control strategy.

**Assumption 1.** The desired dog positions $d^*_j$ (4.17) evolve slowly enough compared to the speed of our dogs $\dot{d}_j$ (4.21) that we can assume perfect tracking, $d^*_j = d_j$.

In practice, this assumption means the dynamics of the ideal unicycle-like system are significantly slower than the dynamics to drive the dogs to their ideal positions. Under Assumption 1, the desired positions $d^*_j$ are constant and the tracking controller (4.21), $d_j$ converges exponentially to $d^*_j$. This allows us to start the dogs from any point within the environment, and they will converge upon the ideal unicycle-like system. We validate this assumption in Matlab simulations and hardware experiments.

The following algorithm summarizes the steps in the controller incorporating the radial controller (4.20) and tracking controller (4.21). In the case of a single sheep, the radius is always constant, thus, $\dot{r} = 0$.

**Algorithm 1** Herding Control

1: Calculate the controller for $\dot{p}$ (4.16)
2: Find ideal heading $\psi^*$ and velocity $v^*$ (4.6)
3: Find $\Delta_j^*$ from $v^*$ using (4.15)
4: Calculate desired dog positions $d^*_j$ (4.17)
5: Calculate radial controller for $\dot{r}$ (4.20)
6: Calculate tracking controller for $\dot{d}_j$ (4.21)
4.3 3D Problem Formulation and Controller Design

In this section, we present the herding problem formulation for a 3D system. For consistency with the previous section, we denote the \( m \) herders as “dogs” with positions \( d_j \in \mathbb{R}^3 \), and \( n \) herd members as “sheep” with positions \( s_i \in \mathbb{R}^3 \). Of course real dogs and sheep do not move freely in three dimensions, but we keep the analogy for ease of the reader. In practice, 3D herding may be applied to situations with aquatic life, such as relocating or trapping fish, applied to spacecraft, or applied to aerial vehicles, such as quadrotors. Here, we drive the dogs to a configuration in which controlling the herd is like controlling a nonholonomic vehicle. First, we formulate a kinematic model for a nonholonomic vehicle, introduce geometric constraints that reduce the herd dynamics to a nonholonomic vehicle, and later present our controllers for the herd.

As before, we model the sheep’s repulsion from the dogs using an artificial potential field (4.2). Here, our goal region \( B_\ell(g) \) is a ball defined around a goal point \( g \in \mathbb{R}^3 \) with radius \( \ell > 0 \). Without loss of generality, we define our coordinate frame to be centered at the goal point, such that \( g = 0 \).

4.3.1 Modeling of a 3D Nonholonomic Vehicle

Consider a three-dimensional nonholonomic vehicle with position \( s = [x \ y \ z]^T \) relative to the global reference frame, shown in Figure 4-5a. For this vehicle, we define a local reference frame \( B \) relative to some global frame \( A \). Its forward velocity \( v \) defines the local \( b_x \) direction, as shown in the figure. The vehicle is also able to rotate around its \( x \)-, \( y \)-, and \( z \)-axis with angular velocities \( \omega_x, \omega_y, \) and \( \omega_z \), respectively. This is a common model in both underwater autonomous vehicle modeling [Nakamura and Savant, 1991, Aicardi et al., 2000, Aicardi et al., 2001, Egeland et al., 1994, Canudas de Wit and Sordalen, 1991] and aerial vehicles [Roussos et al., 2008, Roussos et al.,]
Using the North-East-Down notation, the rotation matrix between the local frame \( B \) to the global frame \( A \) is

\[
A^B R = \begin{bmatrix}
\cos \psi \cos \theta & \cos \psi \sin \theta \sin \phi - \sin \psi \cos \phi & \sin \psi \sin \phi + \cos \psi \cos \phi \sin \theta \\
\sin \psi \cos \theta & \cos \psi \cos \phi + \sin \psi \sin \theta \sin \phi & \sin \theta \sin \psi \cos \phi - \cos \psi \sin \phi \\
-\sin \theta & \cos \theta \sin \phi & \cos \theta \cos \phi 
\end{bmatrix},
\]

where \( \phi, \theta, \) and \( \psi \) are the ZYX Euler angles, as described in Figure 4·5b. We express the \( B \) frame in global coordinates as

\[
b_x = A^B R \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad b_y = A^B R \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad b_z = A^B R \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.
\]
Overall, the kinematic constraints of the vehicle dynamics are

\[
\dot{s} = ARB \begin{bmatrix} v \\ 0 \\ 0 \end{bmatrix},
\]

\[
\begin{bmatrix}
\dot{\phi} \\
\dot{\theta} \\
\dot{\psi}
\end{bmatrix} = \begin{bmatrix}
1 & \sin(\phi) \tan(\theta) & \cos(\phi) \tan(\theta) \\
0 & \cos(\phi) & -\sin(\phi) \\
0 & \sin(\phi) \sec(\theta) & \cos(\phi) \sec(\theta)
\end{bmatrix} \begin{bmatrix}
\omega_x \\
\omega_y \\
\omega_z
\end{bmatrix},
\]

where \( v \) is the velocity of the vehicle and \( \omega_x, \omega_y, \) and \( \omega_z \) are the angular velocities of the vehicle [Nakamura and Savant, 1991].

### 4.3.2 Point-Offset Control

Similar to the 2D case, we wish to control the nonholonomic vehicle by controlling a point offset along the \( b_x \) axis, and it turns out that the point \( p \) is holonomic. To our knowledge, the point-offset control for a 3D nonholonomic vehicle does not exist in literature, so we present the formulation in the following section. For a vehicle with position \( s \), the point \( p \) is written:

\[
p = s + ARB \begin{bmatrix} \ell \\ 0 \\ 0 \end{bmatrix}.
\]

Its derivative is

\[
\dot{p} = \dot{s} + \frac{d}{dt} [ARB] \begin{bmatrix} \ell \\ 0 \\ 0 \end{bmatrix}.
\]

By definition, \( \dot{s} = vb_x \), since our vehicle can only move forward or backwards in its local x-axis. It is well known from 3D kinematics that the derivative of a rotation matrix reduces to [Murray et al., 1994]

\[
\frac{d}{dt} [ARB] = ARB \Omega, \quad \Omega = \begin{bmatrix}
0 & -\omega_z & \omega_y \\
\omega_z & 0 & -\omega_x \\
-\omega_y & \omega_x & 0
\end{bmatrix},
\]
where $\Omega$ is a skew-symmetric matrix of the local angular velocities. Overall, (4.24) reduces to

$$
\dot{p} = A R^B \begin{bmatrix}
v & -\ell \omega_z & \ell \omega_y \\
\ell \omega_z & v & -\ell \omega_x \\
-\ell \omega_y & \ell \omega_x & v
\end{bmatrix} \begin{bmatrix}
1 \\
0 \\
0
\end{bmatrix}. 
$$ (4.25)

To solve for $[v, \omega_x, \omega_y, \omega_z]$, we rearrange (4.25) as

$$
B R A \dot{p} = \begin{bmatrix} v \\
\ell \omega_z \\
-\ell \omega_y
\end{bmatrix},
$$ (4.26)

and let $\omega_x = 0$. This relationship allows us to find some desired control, $\dot{p} = u$ and map it back to the vehicle controls, $v$, $\omega_z$, and $\omega_y$. We can now present our lemma defining the point-offset control of a 3D nonholonomic vehicle.

**Lemma 1** (Point-Offset Control in 3D). For a nonholonomic vehicle located at $s$ with forward velocity $v$ and angular velocities $(\omega_x, \omega_y, \omega_z)$, using the controller

$$
v = -k(b_x^T p),
$$

$$
\omega_x = 0,
$$

$$
\omega_y = \frac{k}{\ell}(b_y^T p),
$$

$$
\omega_z = -\frac{k}{\ell}(b_y^T p),
$$ (4.27)

the vehicle converges to a ball of radius $\ell$ around the origin.

**Proof.** It is equivalent to say that if the point offset converges to the origin, the herd converges to the ball $B_{\ell}$ about the origin. Consider the Lyapunov candidate function

$$
V = \frac{1}{2} p^T p = \frac{1}{2} \left( s + A R^B \begin{bmatrix} \ell \\
0 \\
0
\end{bmatrix} \right)^T \left( s + A R^B \begin{bmatrix} \ell \\
0 \\
0
\end{bmatrix} \right).
$$

Its derivative is

$$
\dot{V} = \left( s + A R^B \begin{bmatrix} \ell \\
0 \\
0
\end{bmatrix} \right)^T \left( A R^B \begin{bmatrix} v \\
\ell \omega_z \\
-\ell \omega_y
\end{bmatrix} \right).
$$
Substituting (4.27) yields
\[
\dot{V} = p^T \left( (-kb_x^T p) b_x + (-kb_y^T p) b_y + (-kb_z^T p) b_z \right)
= -k\|p\|^2 \leq 0.
\]

By Lyapunov’s direct method [Khalil, 2002], the equilibrium point \( p^* = 0 \) is asymptotically stable. Furthermore, by the form of \( \dot{V} \), we see that \( p^* = 0 \) is exponentially stable. When \( p = 0 \), the vehicle is a distance \( \ell \) away from the origin, thus completing the proof of Lemma 1.

\[\square\]

**Remark 7.** For the purposes of our controller, we allow the vehicle to move forwards and backwards in the \( b_x \) direction. However, if the vehicle can only move forward, i.e., \( v \geq 0 \), then there is a discontinuity in the control policy when the desired direction falls along the \( -b_x \) direction. This discontinuity can be resolved by rotating about the local \( z \)-axis until the \( b_x \) axis is aligned with the intended direction.

### 4.3.3 Kinematic Reduction

Instead of allowing the herders to occupy any point in the environment, consider the case where all herders are a fixed distance \( r \) from the sheep. We show that when this occurs, the system dynamics reduce to a 3D nonholonomic vehicle as described in Section 4.3.1. For the case of \( n = 1 \) sheep and \( m \) dogs, the relative location of a dog becomes
\[
d_j = s + A R^B \begin{bmatrix} r \cos(\alpha_j) \sin(\beta_j) \\ r \sin(\alpha_j) \sin(\beta_j) \\ r \cos(\beta_j) \end{bmatrix},
\]
where \( \alpha_j \) is the azimuthal angle and \( \beta_j \) is the polar angle relative to the herd.

Previously, we introduced \( \Delta \) that described the angular separation between \( d_1 \) and \( d_m \), and evenly distributed all other dogs within \( \Delta \). Extending these principles to this spherical case, we can similarly distribute the herders across their azimuthal and polar angles. Let \( \Delta_\alpha \) and \( \Delta_\beta \) describe the separation in the azimuthal and polar
angles. We then define the individual angles as

\[
\alpha_j = \Delta_{\alpha_j} - \pi, \quad \beta_j = \Delta_{\beta_j} + \frac{\pi}{2},
\]

where

\[
\Delta_{\alpha_j} = \Delta_{\alpha} \frac{(2j - m - 1)}{(2m - 2)} \quad \text{and} \quad \Delta_{\beta_j} = \Delta_{\beta} \frac{(2j - m - 1)}{(2m - 2)}.
\]

In our implementation, we assign \( \Delta_{\beta} = \pi \) and vary \( \Delta_{\alpha} \). Figure 4-6 illustrates how the herders distribute themselves along a spiral wrapping around the herd’s sphere.

The spiral wrapping is one method of distributing the herders around the sphere, and what we focus on in our subsequent controller design. Other methods may be to place the herders on fixed latitude and longitude “tracks”, or a mesh grid that scales with the desired velocity. Alternatively, the herders could be placed in a platonic solid formation [Wenninger, 1974] about the herd, creating a net-zero repulsion on the herd while surrounding the group in all directions. From there, the herd could be controlled by displacing the herder group as to direct the herd in a desired direction. A limitation of platonic solids is that they require particular group sizes, limiting the number of herders to \( m = \{4, 6, 8, 12, 20\} \). We choose the spiral design due to its
ability to scale to any number of herders while only requiring two parameters, $\Delta_\alpha$ and $\Delta_\beta$, to control the group.

By distributing the herders around the parameterized spiral, we find that the dynamics reduce to a 3D nonholonomic vehicle similar to a fixed-wing aircraft or an autonomous underwater vehicle (AUV). Given the desired angular separations $\Delta_\alpha$ and $\Delta_\beta$, the herd’s dynamics in (4.2) become

$$\dot{s} = -\frac{1}{r^2} A^B \left[ \sum \cos(\alpha_j) \sin(\beta_j) \right] \sum \sin(\alpha_j) \sin(\beta_j) \sum \cos(\beta_j).$$

To simplify this expression, we first note

$$\sum \cos(\beta_j) = \sum \cos \left( \frac{\pi}{2} + \frac{2j - m - 1}{2m - 2} \right) = 0.$$  

Next, we apply a product-to-sum rule to the remaining expressions,

$$\sum \cos(\alpha_j) \sin(\beta_j) = \frac{1}{2} \left( \sin(\alpha_j + \beta_j) - \sin(\alpha_j - \beta_j) \right) = \frac{1}{2} \left( \sin \left( \frac{m(\Delta_\alpha - \Delta_\beta)}{2 - 2m} \right) + \sin \left( \frac{m(\Delta_\alpha + \Delta_\beta)}{2 - 2m} \right) \right),$$

and

$$\sum \sin(\alpha_j) \sin(\beta_j) = \frac{1}{2} \left( \cos(\alpha_j - \beta_j) - \cos(\alpha_j + \beta_j) \right) = 0.$$  

Overall, the expression for the velocity becomes

$$\dot{s} = -\frac{1}{2r^2} \left( \sin \left( \frac{m(\Delta_\alpha - \Delta_\beta)}{2 - 2m} \right) + \sin \left( \frac{m(\Delta_\alpha + \Delta_\beta)}{2 - 2m} \right) \right) b_x,$$

which implies that the herd only moves in its local $b_x$ direction. Furthermore, to maintain the kinematic relationship, we see the dogs’ dynamics
become
\[ \dot{d}_j = \dot{s} + A^R B \Omega \begin{bmatrix} -r \sin(\Delta_{\alpha j}) \cos(\Delta_{\beta j}) \\ -r \cos(\Delta_{\alpha j}) \cos(\Delta_{\beta j}) \\ -r \sin(\Delta_{\beta j}) \end{bmatrix} + A^R B \dot{\Delta}_{\alpha j} \begin{bmatrix} -r \cos(\Delta_{\alpha j}) \cos(\Delta_{\beta j}) \\ r \sin(\Delta_{\alpha j}) \cos(\Delta_{\beta j}) \\ 0 \end{bmatrix} \],
\]

(4.29)

where \( \Omega \) is the skew-symmetric matrix of local angular velocities and \( \dot{\Delta}_{\alpha j} \) is the derivative of the local azimuthal angle. Note that since \( \Delta_{\beta} \) is constant, \( \dot{\Delta}_{\beta j} = 0 \). By defining the dogs in terms of the rotation matrix \( A^R B \) and angular separations \( \Delta_{\alpha} \) and \( \Delta_{\beta} \) along some radius and restricting the kinematics to obey (4.29), we can map these quantities to the linear and angular velocities of a 3D nonholonomic vehicle.

**Proposition 3.** The herding dynamics in (4.28) and (4.29) can be reduced to an equivalent three-dimensional nonholonomic vehicle with forward velocity \( v \) and angular velocities \( (\omega_x, \omega_y, \omega_z) \) described by (4.22).

**Proof.** To see this mapping, note that the direction of herd is determined by its velocity, which moves solely in the local \( x \)-direction. For the velocity, we find from (4.28)
\[ v = \| \dot{s} \| = \frac{1}{2r^2} \left( \frac{\sin \left( \frac{m(\Delta_{\alpha}-\Delta_{\beta})}{2-2m} \right)}{\sin \left( \frac{\Delta_{\alpha}-\Delta_{\beta}}{2-2m} \right)} + \frac{\sin \left( \frac{m(\Delta_{\alpha}+\Delta_{\beta})}{2-2m} \right)}{\sin \left( \frac{\Delta_{\alpha}+\Delta_{\beta}}{2-2m} \right)} \right). \]

(4.30)

Note that for (4.30), there are an infinite number of possible values of \( \Delta_{\alpha} \) and \( \Delta_{\beta} \) for a given value of \( v \). However, when \( \Delta_{\beta} = \pi \), over the range of \( \Delta_{\alpha} = (0, 3\pi) \), this mapping is one-to-one. Thus, for a given velocity, we can find the corresponding \( \Delta_{\alpha} \).

The dynamics for \( \dot{\Delta}_{\alpha} \) are also be found from the dynamics of \( \dot{v} \).

It remains to map the angular rotation rates to the dogs dynamics. Recall from (4.29) that the dogs’ kinematics directly incorporate \( \Omega \), the skew-symmetric matrix of angular velocities. Thus, given a linear velocity \( v \) and angular velocities \( (\omega_x, \omega_y, \omega_z) \), we can control the herd dynamics to that of a 3D nonholonomic vehicle.

Ultimately, we use Proposition 3 in our controller design of the system. Instead of determining individual controllers for each of the herding agents, we instead design controllers for the ideal nonholonomic-like system. Based on this idealized system, we then find the controllers for the herders to enforce this behavior.
4.3.4 Controller Design

Section 4.3.3 introduced the geometric constraints on the system, which allow us to map the kinematics of the herding system to a 3D nonholonomic vehicle. Our goal, as stated in Problem 2, is to relocate the herd to some ball around a point in the environment. To control this system, we use a point-offset controller as described in Section 4.3.2, although other nonholonomic vehicle controllers can be mapped using our kinematic reduction.

To find the desired dog positions, we first find a controller for a point offset $p$ from the herd, defined in (4.23). While there exist many possible choices for controlling $p$, we opt for a simple proportional feedback controller,

\[ \dot{p} = -kp. \] (4.31)

Substituting (4.31) into (4.26) yields the controllers for $(v, w_y, w_z)$ presented in (4.27). Using the mapping defined in (4.30), we calculate the desired separation $\Delta^*\alpha_j$ for the herders. Furthermore, the desired angular velocities yield $\Omega$, which we use to calculate a desired rotation matrix $A^RB^*$. We combine this to determine the locations of the herders, written

\[ d_j = s + A^RB^* \begin{bmatrix} -r \sin(\Delta^*_\alpha_j) \cos(\Delta^*\beta_j) \\ -r \cos(\Delta^*_\alpha_j) \cos(\Delta^*\beta_j) \\ -r \sin(\Delta^*\beta_j) \end{bmatrix}. \] (4.32)

Proposition 4. For the single herd, multi-herder system in (4.28) and (4.29), the herd converges to the ball of radius $\ell$ about the origin, $B_\ell$.

Proof. Consider the point offset $p$ located a distance $\ell$ away from the herd, defined in (4.23). It is equivalent to say that is the point offset converges to the origin, the herd converges to the ball $B_\ell$ about the origin.

By Proposition 3, we know that our mapping in (4.30) reduces the system to a 3D nonholonomic-like system. Additionally, by Lemma 1, we show that the controllers in (4.27) drives the nonholonomic vehicle to a ball $B_\ell$ about the origin. Thus, we apply these results and conclude that our multi-herder system converges to a ball of
radius $\ell$ about the origin.

In practice, we may want the herders to start away from the herd. Similar to the 2D case, we can use Assumption 1 that states the desired positions of the dogs evolve slowly enough compared to the speed of our dogs $\dot{d}_j$ that we can start the dogs anywhere in the environment and they will converge upon the ideal nonholonomic vehicle system. The tracking controller is written

$$\dot{d}_j = -K_d (d^*_j - d_j),$$

where $d^*_j$ is the desired position given in (4.32). Algorithm 2 summarizes the overall controller for herding in 3D.

**Algorithm 2** Herding Control in 3D

1: Calculate the controller for $\dot{p}$ (4.31)
2: Calculate ideal velocities $(v^*_x, \omega^*_x, \omega^*_y, \omega^*_z)$ (4.26)
3: Find ideal rotation matrix $AR^B*$ from $\Omega^*$
4: Find $\Delta^*_\alpha$ from $v^*$ (4.30)
5: Calculate desired dog positions $d^*_j$ (4.32)
6: Calculate tracking controller for $d_j$ (4.33)

### 4.4 Methods for Decentralization in Multi-Agent Herding

Ultimately, we would like to control the herders in a decentralized fashion. This section outlines an algorithm that combines tools from centroidal Voronoi control with our coverage algorithm. Since the herder control laws reduce to a few common parameters for all herders, we only require a few more steps to create a decentralized implementation. One simplistic idea might be to have every herder perform the controller calculations in parallel, as each dog only needs to know its index and the total group size to find its control action. However, this is a feed-forward design, and
does not account for potential performance and sensing variations among the dogs, so it will not be robust in practice.

Another approach is to utilize a centroidal Voronoi algorithm to achieve even spacing in a decentralized fashion. It can be shown that a centroidal Voronoi configuration in one dimension leads to all agents equally spaced [Drezner, 1995]. Applying a move-to-centroid algorithm [Cortes et al., 2004] on the parameterized line for the dogs would guarantee convergence to the desired equitable partitioning.

![Figure 4.7: Illustration of a 2D move-to-centroid algorithm on a radius about the herd. When the two leader dogs (pink) move in (b), the remaining dogs (blue) calculate their new centroids (black circles) and move towards the new positions in (c)-(d).](image)
Consider a heterogeneous group of dogs, where two dogs are responsible for deciding the control policy, and others follow the leader dogs. Here, the two leader dogs are $d_1$ and $d_m$. The leader dogs coordinate to choose the next control action based on the position of the sheep. Once they update their positions, all follower dogs use a move-to-centroid algorithm to update their position, based on that of their neighbors, demonstrated in Figure 4·7 for a 2D system and Figure 4·8 for the 3D system. By introducing this move-to-centroid algorithm, we reduce the information needed
by each herder to successfully control the sheep. Instead of requiring all herders to determine the herd’s position and their resulting control law, we let two dogs communicate the required information to their neighbors. Here we assume that all dogs can communicate with their neighbors, and that herders $d_1$ and $d_m$ are neighbors for communication.

**Algorithm 3 Decentralized Herding Control**

1: Lead dogs calculate position of sheep and desired controller $\dot{p}$ (4.31)
2: Ideal lead dog positions calculated via kinematic mapping
3: Remaining dogs employ centroidal Voronoi algorithm to calculate position

### 4.5 2D Simulations

The following simulations were performed in Matlab to demonstrate the capabilities of our herding algorithm. First, we present simulations illustrating the case of $n = 1$ sheep with $m$ dogs in 2D. Despite starting from random configurations, our system converges to the dynamics of the ideal unicycle-like vehicle, and we successfully relocate the herd to a ball around the origin. We also demonstrate our algorithm for multiple sheep, and investigate the effects of including additional inter-agent repelling and attracting forces among the sheep.

#### 4.5.1 Herding with $n = 1$ Sheep

Our first simulation shows the case of $m = 4$ dogs and a single sheep. Figure 4-9 illustrates the configuration and trajectories of all agents over time. In the figure, the green x represents the goal point, and the green circle denotes the goal region $B_\ell(g)$. The blue squares denote the dogs, and black circle and x are the sheep and point offset, respectively.

In Figure 4-9, the dogs do not start near the sheep, but converge to a circle around the sheep, which then drives the point-offset to the origin. To illustrate
the performance for a variety of initial conditions, Figure 4·10 compares the distance between the point offset ($\|p\|$) and the goal over 30 trials. The initial starting locations were randomized for each agent in each of the trials, yet we see in all simulations the point-offset converges to the origin, validating our claims in Proposition 2.
Figure 4.9: (a) Initial configuration of $m = 4$ herders (squares) and single sheep (circle) moving towards goal region. (b) Trajectory of the agents and the point offset, ending within the goal region.
Figure 4.10: Convergence of the point offset to the origin over 30 trials.

4.5.2 Herding with \( n > 1 \) Sheep

For the case of multiple sheep with \( m \) dogs, we add inter-agent forces between the sheep in the herd in addition to the repulsion forces the sheep experience from the dogs. For the purposes of these simulations, we use the flocking dynamics presented in Vaughan’s work ( [Vaughan, 1999, Vaughan et al., 2000] ) for inter-agent forces. Figure 4.11 shows two examples of controlling multiple sheep. The controllers use the herd mean \( \bar{s} \), as well as the radial controller presented in (4.20). For the simulations presented in Figure 4.11, the inter-herd forces have low repulsion relative to the distances to the dogs, meaning the sheep act as a cohesive unit. With these properties, the dogs are still able to control the group to some goal region using our point-offset controller on the mean of the herd.
Figure 4.11: (a) Trajectories of $m = 3$ dogs (squares) and $n = 10$ sheep (circles) moving towards goal region. (b) Trajectories of $m = 5$ dogs and $n = 4$ sheep moving towards goal region. In both cases, the flocking dynamics are cohesive enough to treat the herd as a single unit.
On the other hand, if we set the flocking dynamics to have higher repulsive forces between the herd members, the sheep have a greater tendency to disperse. Figure 4·12 shows the trajectories of two simulations where the sheep experience high repulsive forces. As seen in Figure 4·12, when the sheep have high repulsive forces between each other, the group spirals away from the mean. Surprisingly, the point offset from the mean still remains near the origin, as predicted by our controller. Note that the only metric in the radial controller is to adjust for the variance, but there is nothing in the current controller design to decrease the variance. As the sheep disperse under high repulsive forces, the dogs also disperse, but overall keep the mean of the herd near the origin. Although the controller is “successful” in relocating the point offset to the goal, the herd is not contained within a goal region, indicating the need for further control strategies to control the sheep.
Figure 4.12: Trajectories of (a) $m = 6$ dogs (squares) and single $n = 3$ sheep (circles) and (b) $m = 12$ dogs, $n = 10$ sheep with higher inter-herd repulsive forces relative to the distance to the dogs.
4.6 3D Simulations

This section presents simulations performed in Matlab to demonstrate the capabilities of our herding algorithm in 3D. First, we present a simulation illustrating the case of a single herd agent and \( m \) herders. Despite starting from random configurations, our system converges to the dynamics of a nonholonomic vehicle and we successfully relocate the group to a ball around the origin. Next, we present a simulation with multiple herd members and added noise on the measurement of the herd’s mean. Finally, we present the point-offset controller to track a moving goal target, demonstrating a more dynamic example and the flexibility of our controllers.

4.6.1 Relocation to Goal

Our first simulation shows the case of \( m = 10 \) herders and a single herd agent. Figure 4·13 shows the system evolving over time. In the figure, the green x denotes the goal point. The blue squares denote the herders, and the blue circle and x are the herd and the point offset, respectively.
Figure 4.13: Positions of the $m = 10$ herders (squares) and herd (circle) over time. The goal is denoted by the green x. Over time, the herd is relocated to the region around the goal.
Figure 4.14: Distance of the herd point offset over time. Once the herd reaches the goal, it maintains its position.

Similar to the 2D case, we extend our algorithm to a herd with flocking dynamics. Figure 4.15a shows a simulation with $n = 3$ herd members and $m = 7$ herders. The flocking dynamics create a noisy point offset from the mean, shown in 4.15b. Despite controlling multiple herd members with flocking dynamics, the herders successfully relocate the herd to a ball about the origin.
Figure 4.15: (a) Trajectories of the herd with flocking dynamics (squares) and herders (circles) over time. (b) Point offset from the mean of the herd.
4.6.2 Target Tracking

We can extend the applications of our controller to more than just relocation problems. This simulation illustrates the performance of our controllers for a moving goal point, similar to target tracking. Here, $m = 7$ herders continuously track the goal region. Figure 4.16(a) illustrates the trajectories of the goal point and point offset over time. Figure 4.16(b) shows the relative displacement between the goal and the point offset for the $x-$, $y-$, and $z-$ coordinates. As seen in the Figure, our point offset control successfully tracks a moving goal region.
Figure 4.16: (a) The trajectory of the goal (green) and point offset (blue) over time. The herders are able to track the goal. (b) Position of the goal (green) and point offset (blue) over time. Note that while the herd initial begins away from the goal, it is able to continuously track it over time.
4.6.3 Robustness to Noise

In this section, we investigate the robustness to noise in the 3D system. We look at two main sources of noise: noise added to the movement of the herd, and noise in dogs’ measurement of the herd’s position. For each case, we add varying levels of noise to examine the effect on the dogs’ ability to relocate the herd to the goal.

To model noise in the movement of the herd, we consider a group of \( m = 7 \) dogs and \( n = 1 \) sheep. Noise was added to the sheep’s dynamics (4.2) as

\[
\dot{s} = \sum_{j=1}^{m} -\frac{(d_j - s)}{\|d_j - s\|^3} + K_n X,
\]

where \( X \) is a random number drawn from a uniform distribution and \( K_n \) varies the magnitude of \( X \). The noise is not directly seen by the dogs during their calculation for \( \dot{d}_j \), who calculate their controller assuming the potential field model for the sheep’s dynamics. To study the effect of noise, we varied \( K_n \) and examined the distance of the point-offset from the goal at time \( t = 800 \) seconds. We chose this time to be well beyond the time required to relocate the herd to the origin, which will verify if the herders were successfully able to maintain the herd at the goal. For each value of \( K_n \), we performed 70 trials with randomized initial configurations. Figure 4·17 shows the mean of \( \|p\| \) at 800 seconds over each set of 70 trials. We compare these values to the ratio of \( \frac{K_n}{v_{\text{max}}} \), where \( v_{\text{max}} = -\frac{m}{r} \) is the maximum possible velocity for the herd induced by the dogs at a set control radius \( r \). From Figure 4·17, for values of \( \frac{K_n}{v_{\text{max}}} < 1 \), the herders are able to maintain the herd at the goal region. For \( 1 < \frac{K_n}{v_{\text{max}}} < 1.2 \), we notice the average value of \( \|p\| \) increases sharply. Table 4.1 summarizes the success rate of the herders relative to the noise value. Note that the transition between successful herding and failed attempts occurs around \( \frac{K_n}{v_{\text{max}}} > 1 \), once the dynamics of \( \dot{s} \) are dominated by the noise instead of the repulsion forces induced by the dogs.
Figure 4.17: Mean and error bars for final values of $\|p\|$ over varying values of noise in herd movement. The herders are able to control the herd to the goal region until $\frac{K_n}{v_{\max}} > 1$, when the noise dominates the herd movement.

<table>
<thead>
<tr>
<th>Noise Ratio</th>
<th>$\frac{K_n}{v_{\max}} &lt; 1$</th>
<th>$1 \geq \frac{K_n}{v_{\max}} \leq 1.2$</th>
<th>$\frac{K_n}{v_{\max}} &gt; 1.2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success Rate</td>
<td>99.7%</td>
<td>42.9%</td>
<td>0 %</td>
</tr>
</tbody>
</table>

Table 4.1: Success rate of trials as compared to the ratio $\frac{K_n}{v_{\max}}$.

To study the effect of noise in measurement, we add noise to the herder’s estimate of the herd’s position, with each herder using a different estimate of the herd. We use

$$\hat{h}_j = h + K_nX_j,$$

where $h$ is the true herd position, $X_j$ is a random number drawn from a uniform distribution, and $K_n$ is a gain on the magnitude of random noise. For each value of $K_n$, we ran 70 trials with randomized initial configurations at looked at the distance of the point offset to the goal at 800 seconds. Figure 4.18 summarizes the mean final
position $\|p\|$ at 800 seconds. Here, we compare to values of $\frac{Kn}{r}$, where $r$ is the control radius of the herd. From Figure 4.18 and Table 4.2, we notice a sharp transition in the success rate around $\frac{Kn}{r} = 1$. For $\frac{Kn}{r} < 1$, this implies the estimate of the herd’s position is within the control radius.

**Figure 4.18:** Mean and error bars for final values of $\|p\|$ for varying values of noise in measurement of the herd’s position. The herders are able to successfully relocate the herd to the goal region for values of $\frac{Kn}{r} < 1$.

<table>
<thead>
<tr>
<th>Noise Ratio</th>
<th>$\frac{Kn}{r} &lt; 0.9$</th>
<th>$0.9 \geq \frac{Kn}{r} \leq 1$</th>
<th>$\frac{Kn}{r} &gt; 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success Rate</td>
<td>97.8%</td>
<td>36.9%</td>
<td>0 %</td>
</tr>
</tbody>
</table>

**Table 4.2:** Success rate of trials as compared to the ratio $\frac{Kn}{r}$.

Overall, these simulation results show the system is inherently robust to noise. For noise on the herd’s movement, this implies that if the herd had unmodeled dynamics, but was reasonably close to a potential field repulsion model, our algorithm would be able to account for these changes. Similarly, the noise on the herder’s measurement
of the herd implies that the herders do not need to be perfectly aligned on the control radius to still effectively relocate the herd.

4.7 Experiments

To demonstrate our 2D algorithm, experiments were conducted in the Multi-Robot Systems Lab at Boston University. Our lab utilizes an OptiTrack\textsuperscript{1} system with IR cameras to track reflective markers and provide real-time localization. We use Pololu’s m3pi\textsuperscript{2} robot equipped with an mbed microcontroller and XBee\textsuperscript{3} radio. Position data is obtained from OptiTrack and sent to Matlab, which is then used to compute control laws and send information to the m3pi robots via the XBee radio. Due to the limitations of the mbed microcontroller, computation is done on a central computer and only updated velocity information is sent to the m3pi robots.

The biggest challenge during implementation was the culmination in system inefficiencies not present in simulation. While our simulations assume that all robots have holonomic dynamics, the m3pis are nonholonomic vehicles with noisy, lossy actuation. In addition, the floor mats in the lab introduce a friction force on the robots, requiring them to travel at a minimum speed. These unmodeled behaviors are hard to predict or quantify in simulation. Despite these challenges, we performed repeated successful experiments with the m3pi robots in the loop.

For this experiment, we use \( n = 1 \) m3pi “sheep”, and \( m = 3 \) m3pi “dogs.” Figure 4.19 illustrates the evolution of the system over time. The positions of the dogs (blue squares) and sheep (red circle) have been highlighted in each video frame. The goal region representing \( B_l(g) \) is indicated by the green circle. Over the course of the experiment, we see the herders are able to successfully relocate the sheep to the

\textsuperscript{1}www.naturalpoint.com/optitrack
\textsuperscript{2}www.pololu.com
\textsuperscript{3}www.digi.com/xbee
desired goal region.

Figure 4.20 displays the time history of the sheep and dogs over the course of the experiment. Here, the trajectories are noisier than those seen in simulations. The additional noise comes from the unmodeled dynamics, communication delays, and a low-level nonholonomic controller within the experimental system. Despite the added noise, we still achieve our goal of relocating the sheep to some desired region. This demonstrates an inherent robustness in our feedback controllers to tolerate uncertainty in our system. We can also assess the performance by looking at the distance of the sheep's point-offset $p$ from the goal. Figure 4.21 shows the results of five different trials with randomized initial configurations performing a relocation to the goal. Although there is chatter present, it does not impact the overall performance, and the herd was successfully relocated to the goal in every trial.
Figure 4.19: Images from the experimental video, illustrating the herding of $n = 1$ robot sheep (red circle) by $m = 3$ robot herders (blue squares).
Figure 4.20: Trajectories of the dogs (blue squares) and sheep (black circle). Over time, the dogs successfully relocate the sheep to the goal, despite the additional unmodeled noise.

Figure 4.21: Distance of the sheep’s point offset $p$ to the goal over five trials with varying initial configurations.
4.8 Summary

We consider the problem in which herders seek to control the location of a non-cooperative herd, analogous to sheep herding. The goal is for the herders to relocate the sheep to a region close to a goal point. Despite the highly nonlinear dynamics of the system, using the constraint that the dogs maintain some radius around the herd allows us to map these dynamics to a unicycle-like vehicle in 2D and a forward moving nonholonomic vehicle in 3D, for which a simple feedback controller can be formulated. Unlike previous work in herding, this is done with a single continuous control law and does not rely on switching or heuristic behaviors. For a single sheep with multiple dogs in 2D and 3D, we are able to prove with a Lyapunov-like proof that the sheep converge exponentially to the goal region. We also propose a control strategy for the general case of multiple sheep and multiple dogs. Although the strategy is centralized, we present a method for decentralization of the control strategy, as well as simulations of a target-tracking control law. The simulations and hardware experiments demonstrate the performance of these control strategies.
Chapter 5

Cooperative Multi-Quadrotor Pursuit of an Evader in an Environment with No-Fly Zones

In this chapter, we explore the problem of a cooperative group of pursuers tracking an evader through an environment. This chapter combines concepts from the Voronoi-based coverage control work in Chapter 3 with Robust Model Predictive control for a single pursuer and evader [Ataei and Paschalidis, 2015], and the results appeared in our 2016 ICRA paper [Pierson et al., 2016a]. Here, we consider multiple pursuers tracking a single evader. The evader is free to enter “no-fly zones,” which the pursuers must avoid. Once the evader enters the no-fly zone, the pursuers may not have information about its position. Furthermore, we remain agnostic as to why the evader is moving, whether it is passively moving through the environment or actively evading the pursuers.

When the evader is in free space, it is directly tracked by the pursuers. Once the evader enters a no-fly zone, we create a reachable set of all possible locations within the no-fly zone it could be located. We then use a Voronoi tessellation to divide up the reachable set of possible evader locations, with the goal of minimizing the “cost of capture” once the evader emerges from the no-fly zone. Using a virtual pursuer position, we employ a move-to-centroid algorithm to optimally divide the reachable set among the pursuers. Since the pursuers cannot move inside the no-fly zone, we drive them to the point on the boundary of the no-fly zone that is closest to the virtual
pursuer inside the no-fly zone. We show this minimizes a relevant “cost-to-capture” metric, and demonstrate our algorithm in hardware experiments using three KMEL Nano+ quadrotors pursuing an m3pi ground robot.

The remainder of this chapter is organized as follows. Section 5.1 provides a summary of related work pertinent to this chapter. In Section 5.2 we describe the algorithm for pursuing the evader in free space, and in Section 5.3 we consider the case when the evader is inside a no-fly zone. Section 5.4 describes the RMPC control method used to robustly execute the algorithms in the previous sections using a model of the quadrotors’ dynamics with modeling uncertainties. Section 5.5 describes our hardware experiments, and we present a summary in Section 5.6.

5.1 Related Work

There is no shortage of research on path planning and obstacle for mobile robots [Latombe, 1991, Goerzen et al., 2010], but most algorithms rely on simple approximations of the robot dynamics [Schlegel, 1998, Brock and Khatib, 1999, Ogren and Leonard, 2005, Minguez and Montano, 2004]. If the underlying robot is highly non-linear or is subject to uncertainties, these approximations may result in controller instability and collision with obstacles. Uncertainties can be incorporated into the design with a robust Model Predictive Control (MPC) framework using linear matrix inequalities (LMIs) [Kothare et al., 1996]. This was extended to guarantee obstacle avoidance for quadrotors in [Ataei and Paschalidis, 2015], which proposed a path planning algorithm that incorporated the RMPC technique for guaranteed collision avoidance in the presence of uncertainties.

We also draw inspiration from Voronoi-based coverage control. A Voronoi-based coverage strategy first proposed by Cortés et al [Cortes et al., 2004, Cortés, 2010], often referred to as the move-to-centroid controller, drives all robots continuously
towards the centroids of their Voronoi cells. This builds upon previous work in the optimal location of retail facilities [Drezner, 1995], and in data compression [Du et al., 1999]. The Voronoi-based control strategy can also be used to track intruders by a team of robots within an environment [Pimenta et al., 2010, Lee and Egerstedt, 2013]. However, these works do not consider our no-fly zone obstacles in the environment, nor do they use the guaranteed-safe RMPC tools presented in this chapter.

Voronoi-based strategies are also present in multi-agent pursuit-evasion. Other works [Huang et al., 2011, Pan et al., 2012, Liu et al., 2013] use the properties of the Voronoi cells to design a pursuer strategy that guarantees capture of an evader. These works will be further explored in Chapter 6 in studying the guaranteed capture of multiple evaders by one or more pursuers. While these strategies have been tested for convex and non-convex environments, they do not consider the no-fly zones that evaders can enter but pursuers must avoid.

Our strategy directs the pursuers around the no-fly zone boundary. In a similar fashion to our approach, Susca et. al use a Voronoi-based strategy to control a group of agents to monitor an evolving boundary [Susca et al., 2008], however, their work focuses on classifying the environment boundary. The purpose of navigating the pursuers around the obstacle is to find the best position once the evader emerges. In [Breitenmoser et al., 2010], the authors study the problem of Voronoi coverage in environments with obstacles, and employ a bug algorithm to navigate around objects. These works do not consider the pursuit-evasion problem, and don’t use the RMPC tools used here.

5.2 Zone-Aware Pursuit in Free Space

We first consider the case where the evader is in free space and summarize our path planning algorithm to steer the pursuers towards the evader while avoiding entering
any no-fly zone. The full details of this algorithm are presented in [Ataei and Paschalis, 2015]. In Section 5.3, we provide the tools necessary to track the evader when it is inside a no-fly zone. Throughout this chapter, we assume that the pursuers can determine the position of the evader when it is in free space, e.g. from an on-board camera or other sensing system, or from a tracking beacon on the evader. We also assume that the pursuers know their own positions, e.g., from GPS.

Let \( n_p \) be the number of pursuers in the group. Define \( p_i(k) \) and \( e(k) \) to be the \( xy \)-coordinates of the pursuer \( i \) and the evader at time \( k \), respectively. Similarly, let \( h_{pi}(i = 1, 2, \ldots, n_p) \) and \( h_e \) denote the altitude of pursuer \( i \) and the evader, respectively. As an additional safety measure to avoid collision between the pursuer quadrotors, we set \( h_{pi} \neq h_{pj} \) for all \( i \neq j \), however this may also be achieved with existing collision avoidance algorithms.

Consider the case where the evader is stationary and is close enough to the pursuer such that there exists an ellipsoid, completely outside any no-fly zone, that is centered at the position of the evader and includes pursuer \( i \) (see Fig. 5.1a). We refer to this ellipsoid as a “safety ellipsoid.” Pursuer \( i \) can safely reach the evader if there exists a controller which guarantees both that it never leaves its safety ellipsoid and also that it asymptotically converges to the center of the calculated ellipsoid. In Section 5.4.1, we review a robust MPC method which can be used to guarantee both conditions.
Figure 5-1: (a) Construction of the maximum-volume ellipsoid. Shaded areas represent the no-fly zones. (b) Schematic of the Path Planning Algorithm (Ataei and Paschalidis, 2015), Algorithm 2).
Let \( r_{\text{min}} \) and \( r_{\text{max}} \) be the lower and upper bounds on the semi-minor axis and the semi-major axis of the safety ellipsoid as shown in Fig. 5.1a. Let \( \upsilon_\kappa, \kappa = 1, \ldots, n_\upsilon \) denote a sampled set of no-fly zone boundary points at the distance of less or equal to \( r_{\text{max}} \) from the evader. For pursuers \( i = 1, \ldots, n_p \), define the maximum-volume safety ellipsoid as

\[
Q^i_k \triangleq \{ z \in \mathbb{R}^2 \mid (z - e(k))^T Q^i_{\varepsilon} (z - e(k)) \leq 1 \},
\]

where \( Q^i_{\varepsilon} \) solves the semidefinite programming problem

\[
\begin{align*}
\min & \quad \text{trace}(Q^i_{\varepsilon}) \\
\text{s.t.} & \quad (\upsilon_\kappa - e(k))^T Q^i_{\varepsilon} (\upsilon_\kappa - e(k)) \geq 1 + \varepsilon, \\
& \quad (p_i(k) - e(k))^T Q^i_{\varepsilon} (p_i(k) - e(k)) \leq 1 - \varepsilon, \\
& \quad \kappa = 1, \ldots, n_\upsilon, \quad \frac{1}{r_{\text{max}}^2} I \leq Q^i_{\varepsilon} \leq \frac{1}{r_{\text{min}}^2} I.
\end{align*}
\]

In the above problem, \( \varepsilon > 0 \) is a parameter that ensures neither the sampled boundary points nor \( p_i(k) \) fall on the boundary of \( Q^i_k \).

The optimization problem in (5.2) can occasionally become infeasible, e.g., when the distance between the evader and a pursuer is larger than \( r_{\text{max}} \) or when the direct line between a pursuer and the evader crosses a no-fly zone. In such scenarios, we search for a “dummy” evader which can move the pursuer in a direction towards the evader. Let \( e^*_i \) be the dummy evader corresponding to pursuer \( i \) and define \( Z_i(i = 1, \ldots, n_z) \), a no-fly zone, where \( n_z \) denotes the number of no-fly zones in the environment. Since the no-fly zones can be non-convex, we calculate a minimum-volume ellipsoid around them, which we use to navigate around a no-fly zone. Let \( E_{Z_r} \) be the minimum-volume ellipsoid around \( Z_r \) that is centered at the centroid of \( Z_r \) and is \( r_{\text{min}} \) distance from the closest boundary point of \( Z_r \).

Let \( \text{line}(p_i, e) \) to be the line segment from \( p_i \) to \( e \). Define \( \text{los}(p_i, e) \) to be the line-of-sight indicator function such that \( \text{los}(p_i, e) \) is 0 if \( \text{line}(p_i, e) \) crosses a no-fly zone.
and 1 otherwise. Consider the case where $\text{los}(p_i, e) = 1$. Note that the optimization problem (5.2) is infeasible if any point along $\text{line}(p_i, e)$ falls within $r_{\min}$ distance away from a no-fly zone boundary. Procedure 1 verifies if this problem exists and resolves it by assigning a new dummy evader ($e^*_i$) to pursuer $i$.

**Procedure 1** Proximity Verification Procedure [Ataei and Paschalidis, 2015]

| Input: | $p_i, e, Z_j, E_{Z_j} (j = 1, \ldots, n_z)$ |
| Output: | $e^*_i$ \hspace{1cm} $\triangleright$ position of the dummy evader for pursuer $i$ |

1: **procedure** PROXIMITY($p_i, e$)  
2: $(x_z, d_z, k) \leftarrow$ the closest point along the boundary of $Z_j$ to $\text{line}(p_i, e)$, its distance, and the corresponding no-fly zone number, respectively  
3: \hspace{1cm} if $d_z \geq r_{\min}$ then  
4: \hspace{2cm} $e^*_i \leftarrow e$  
5: \hspace{1cm} else  
6: \hspace{2cm} $e_z \leftarrow$ the nearest point on $E_{Z_k}$ to $x_z$ with $\text{los}(x_z, e_z) = 1$  
7: \hspace{2cm} $e^*_i \leftarrow e_z$  
8: \hspace{1cm} end if  
9: **end procedure**

Even when $\text{los}(p_i, e) = 1$ and $\text{line}(p_i, e)$ is not in $r_{\min}$ proximity of any no-fly zone, the optimization problem in (5.2) can become infeasible due to the shape of no-fly zones and the distance between the evader and pursuer $i$. In such scenarios, we create a dummy evader and gradually move it towards pursuer $i$ until (5.2) is feasible (see Fig. 5.1b and Algorithm 4). Finally, if $\text{los}(p_i, e) = 0$, we choose the dummy evader $e^*_i$, to be a point on the minimum-volume ellipsoid of the closest intersecting no-fly zone such that $\text{los}(p_i, e^*_i) = 1$ (see Step 11 Algorithm 4).

Algorithm 4 provides a complete path planning algorithm for a given pursuer $i$. Each pursuer track its assigned target evader, $e_i$. When the evader is outside any no-fly zone, all target evaders are set to $e(k)$. However, when the evader is inside a no-fly zone, each pursuer is assigned a different target evader to improve coverage of the no-fly zone as described in Section 5.3. Details on the RMPC controller are provided in Section 5.4.
Algorithm 4 Path Planning Algorithm for Pursuer $p_i$ [Ataei and Paschalidis, 2015]

**Input:** $p_i(k), e_i(k), Z_j, E_{Z_j}$ $(j = 1, \ldots, n_z)$

**Output:** $e^*_i(k), Q^i_\varepsilon$ △ position of dummy evader for $p_i$

1: Pick $0 < \theta_p < 1$
2: $e^*_i \leftarrow e_i(k)$ △ $e_i(k) = e(k)$ if $e(k)$ is in free space
3: $L_1 \leftarrow 1, L_2 \leftarrow 1$ △ feasibility indicator for (5.2)

4: while $L_1 = 1$ do
5: if $e^*_i$ is inside a no-fly zone $r$ then
6: $e^*_i \leftarrow$ nearest point to $e^*_i$ on the boundary of $Z_r$
7: end if
8: if los($p_i, e^*_i$) = 1 then
9: $e^*_i \leftarrow$ PROXIMITY($p_i, e^*_i$)
10: end if
11: if los($p_i, e^*_i$) = 0 then
12: $r \leftarrow$ index of closest obstacle along line($p_i, e^*_i$)
13: $z \leftarrow$ nearest point to $e^*_i$ on $E_{Z_r}$
14: $x_{\{CW,CCW\}} \leftarrow$ nearest point to $z$ on $E_{Z_r}$ in CW and CCW directions with
   los($p_i, x_{\{CW,CCW\}}$) = 1
15: $e^*_i \leftarrow$ Pick $x_{CW}$ or $x_{CCW}$ that yields a shorter arc to $z$
16: $e^*_i \leftarrow$ PROXIMITY($p_i, e^*_i$) △ $x_P$ in Fig. 5.1b
17: else
18: while $L_2 = 1$ do
19: $Q^i_\varepsilon \leftarrow$ Construct max-volume ellipsoid (5.2)
20: if $Q^i_\varepsilon = \emptyset$ then
21: $x_d \leftarrow \theta_p p_i + (1 - \theta_p)e^*_i$
22: else
23: $L_1 \leftarrow 0, L_2 \leftarrow 0$
24: end if
25: end while
26: end if
27: end while

5.3 Pursuing an Evader in a No-Fly Zone

The previous section summarized the multi-robot pursuit algorithm when the evader is outside a no-fly zone, and the pursuers know the evader’s current position. In this section we address the case when the evader enters a no-fly zone. Once inside, the pursuers may not have reliable information about the evader’s position, so the pur-
suers construct a reachable set of possible positions, based on the evader’s maximum velocity and its entry point. The pursuers arrange themselves around the perimeter of the no-fly zone to be ready to capture the evader when it emerges. We adapt strategies from Voronoi-based coverage control to partition the reachable set of the evader, and position the pursuers as close as possible to a centroidal Voronoi configuration over the reachable set, without entering the no-fly zone. We note that the evader could choose the strategy to stay inside a no-fly zone so that it can never be captured. In this case, the pursuers distribute themselves around the perimeter, thereby effectively containing the evader within the no-fly zone.

Consider the no-fly zone \( \mathcal{Z}_j \), where \( q \in \mathcal{Z}_j \) is an arbitrary point in the zone. The set \( \mathcal{Z}_j \) need not be convex. Let \( v_{\text{max}} \) be the maximum speed of the evader. Suppose an evader enters the no-fly zone at some time \( t = \tau \). The entry point is denoted as \( e(\tau) \). For some time \( t \geq \tau \), we can find the reachable set of the evader locations as a ball of radius \( (v_{\text{max}}(t - \tau)) \) centered at \( e(\tau) \), written \( \mathcal{B}(e(\tau), v_{\text{max}}(t - \tau)) \). This ball may include points outside the no-fly zone, so we define \( \mathcal{R}_j(t, \tau, v_{\text{max}}) \) to be the part of the reachable set that is inside the no-fly zone, written

\[
\mathcal{R}_j(t, \tau, v_{\text{max}}) = \mathcal{B}(e(\tau), v_{\text{max}}(t - \tau)) \cap \mathcal{Z}_j.
\]

We then define the indicator function \( \rho(q, t) \) such that \( \rho(q, t) = 1 \) if \( q \in \mathcal{R}_j(t, \tau, v_{\text{max}}) \) at time \( t \), and 0 otherwise. We can write \( \rho(q, t) \) as

\[
\rho(q, t) = \begin{cases} 
1, & \text{if } q \in \mathcal{R}_j(t, \tau, v_{\text{max}}), \\
0, & \text{otherwise}.
\end{cases}
\]

If the pursuers were free to move inside the no-fly zone, we would want to distribute the pursuers to minimize the average distance to any possible location of the evader. Let \( \bar{p}_i \) be the unrestricted desired position of a pursuer. We can also define the
Voronoi tessellation of the reachable set $R_j$ based on these desired positions as

$$V_i = \{ q \in R_j \mid \|q - \bar{p}_i\|^2 \leq \|q - \bar{p}_k\|^2 , k = 1, 2, ..., n_p \} .$$

This allows us to formulate a “cost of capture,” modeled after previous Voronoi-based coverage cost functions introduced in [Cortes et al., 2004, Cortés, 2010] and summarized in Chapter 2. We define our cost of capture function as

$$V(\bar{p}_1, ..., \bar{p}_n) = \sum_{i=1}^{n_p} \int_{V_i} \|q - \bar{p}_i\|^2 \rho(q, t) dq.$$  (5.3)

Intuitively, we see that a low value of $V$ indicates a good configuration of the pursuers. It is also useful to define a “mass” and “centroid” of each Voronoi cell $V_i$, analogous to physical masses and centroids, written

$$M_{V_i} = \int_{V_i} \rho(q, t) dq, \quad C_{V_i} = \frac{1}{M_{V_i}} \int_{V_i} q \rho(q, t) dq.$$  (5.4)

As discussed in Chapter 2, the local minimum of $V$ occurs when all robots are at the centroids of their Voronoi cells, or $\bar{p}_i = C_{V_i}$ for all $i = 1, \ldots, n_p$. In our scenario, if the pursuers could enter the no-fly zone, a move-to-centroid controller like the one presented in [Cortes et al., 2004] would position them to locally minimize the cost of capture of the evader. However, given that our pursuers must remain outside the no-fly zone, we must adapt the move-to-centroid algorithm to this constrained case.

**Remark 8.** Note the above approach can easily be extended to scenarios that utilize predictions about the evader position. This can be captured in either the reachable set $R_j$ or in the indicator function representing the probability of the position estimate for $q \in R_j$.

Let $d_i = p_i - \bar{p}_i$ be the vector from the ideal pursuer position (if the pursuer could enter the no-fly zone) to the actual pursuer position (constrained to lie outside the no-fly zone). Consider minimizing the cost of capture in this case, $V(p_1, ..., p_n)$,
with \( p_i = \bar{p}_i + d_i \), and with \( p_i \) constrained to lie outside \( Z_j \). For \( i = \{1, \ldots, n_p\}, \)
\( j = \{1, \ldots, n_z\} \), we can re-write the cost as a function of the vectors \( d_i \) as

\[
H(d_1, \ldots, d_n) = \sum_{i=1}^{n_p} \int_{V_i} \|q - \bar{p}_i - d_i\|^2 \rho(q, t) dq,
\]

where \( \bar{p}_i = C_{V_i} \), are the locally optimal positions if the pursuers could move inside the no-fly zone. Our optimization problem is then to solve

\[
\min_{(d_1, \ldots, d_n)} H(d_1, \ldots, d_n) \quad \text{s.t.} \quad d_i + \bar{p}_i \notin Z_j.
\]

We find constrained positions to optimize the cost of capture function (5.5) with the following proposition.

**Proposition 5 (Projected Centroids).** Given \( \bar{p}_i = C_{V_i} \) for all \( i = 1, \ldots, n_p \), the locations \( p_i \) that minimize the cost to capture (5.5) are given by \( p_i = \bar{p}_i + d_i^* \), where

\[
d_i^* = \arg \min_{d_i + \bar{p}_i \in \partial Z_j} \|d_i\|,
\]

and \( \partial Z_j \) denotes the boundary of the no-fly zone \( Z_j \).

**Proof.** To see that (5.7) minimizes (5.5) given our constraints, we first expand the cost function as

\[
H = \sum_{i=1}^{n_p} \int_{V_i} \left( \|q + \bar{p}_i\|^2 - 2d_i^T (q - \bar{p}_i) + \|d_i\|^2 \right) \rho(q, t) dq.
\]
Breaking the above equation into three parts, we find $\mathcal{H} = \mathcal{H}_1 + \mathcal{H}_2 + \mathcal{H}_3$, where

$$
\mathcal{H}_1 = \sum_{i=1}^{n_p} \int_{V_i} (\|q - \bar{p}_i\|^2) \rho(q, t) \, dq,
$$

$$
\mathcal{H}_2 = -2 \sum_{i=1}^{n_p} d_i^T \int_{V_i} (q - \bar{p}_i) \rho(q, t) \, dq,
$$

$$
\mathcal{H}_3 = \sum_{i=1}^{n_p} \|d_i\|^2 \int_{V_i} \rho(q, t) \, dq = \sum_{i=1}^{n_p} \|d_i\|^2 M_{V_i}.
$$

We see that $\mathcal{H}_1$ is the same as the unconstrained cost function (5.3), and is independent of $d_i$, so letting $\bar{p}_i = C_{V_i}$ yields a local minimum of $\mathcal{H}_1$ [Drezner, 1995]. When $\bar{p}_i = C_{V_i}$, it follows from (5.4) that $\mathcal{H}_2 = 0$. It remains then to minimize $\mathcal{H}_3$, which is accomplished by minimizing $\|d_i\|$, since $M_{V_i}$ does not depend on $d_i$. Given that the pursuer cannot enter the no-fly zone, $\|d_i\|$ is minimized when $p_i$ is the closest point to the centroid $C_{V_i}$ on the boundary of the no-fly zone $\mathcal{Z}_j$.

**Remark 9.** We assume that the pursuers can navigate to any point on the perimeter of an obstacle. For environments with multiple obstacles, this implies that the obstacles are spaced wide enough that a pursuer can move between the obstacles. If the corridor between obstacles is too narrow, we would treat the group of obstacles as a larger obstacle.

**Remark 10.** Our choice of cost function is formulated based on the expected capture time. In simulations and experiments, this policy works well for the pursuers to track an evader. However, this is not the only cost function that can be used to derive the pursuer policy. If the environment had challenging obstacles, such as a U-shaped or spiral obstacle that could trap a pursuer, it may be necessary to modify the cost function with a “trapping cost” parameter.

Note that our Path Planning Algorithm in effect ensures that $e_i^* = d_i$, where $e_i^*$ denotes the position of the dummy evader for pursuer $i$. Therefore, each centroid $C_{V_i}$ can be treated as an evader target position assigned to a pursuer.
Figure 5.2 illustrates the evolution of our no-fly zone planning algorithm. Once the evader enters a no-fly zone, the pursuers generate an estimate of the reachable set $R_j$, updated at each time step. Between time steps, the pursuers use a move-to-centroid algorithm that projects their desired position to the center of their Voronoi cell (see Algorithm 5). A simple target assignment, using the well-known Hungarian algorithm, matches each centroid point $\bar{p}_i = C_{V_i}$ with a pursuer. The Path Planning Algorithm is then applied to each pursuer and its associated dummy evader. Due to the safety ellipsoids used to prevent collisions, the pursuers will never sit directly on the boundary, but move to the closest allowable point given safety constraints.

**Algorithm 5 No-Fly Zone Planning Algorithm**

**Input:** $e(\tau), t, Z_j, v_{\text{max}}$

**Output:** $\bar{p}_i$

1. Calculate $B(e(\tau), v_{\text{max}}(t - \tau))$ and $R_j(e(\tau), v_{\text{max}}(t - \tau))$
2. Compute Voronoi tessellation about $\bar{p}_i$
3. while $\bar{p}_i \neq C_{V_i} \forall i$ do
   4. Assign $\bar{p}_i = C_{V_i}$
   5. Recompute Voronoi tessellation
4. end while
7. Assign $\bar{p}_i$ as target points for $p_i$

### 5.4 Quadrotor Modeling and Control

Up to this point, we have assumed that we can control the pursuer positions to a desired waypoint, while staying inside an ellipse. Here we describe the low-level controller used to accomplish this, included for completeness. We first control the rotational dynamics of the quadrotor to a desired attitude, which then allows us to control the translational dynamics to a desired location [Bouabdallah and Siegwart, 2007]. The translational sub-system controls the position and velocity of the quadrotor by generating the total rotor output and the desired roll and pitch angles. The rotational sub-system regulates the attitude of the vehicle to the desired values
Figure 5.2: Simulation of three pursuers tracking the evader (red) while executing the no-fly-zone planning algorithm. At each step, the pursuers move to the nearest feasible point to the centroids of the Voronoi cells.
computed by the translational sub-system.

In [Ataei and Paschalidis, 2015], the authors designed a robust MPC controller for the translational sub-system and an LQR controller for the rotational sub-system. Here, we assume the rotational sub-system is regulated through an on-board controller and consider the control design only for the translational dynamics of the aircraft. We borrow elements from the translational controller in [Ataei and Paschalidis, 2015] and expand it to a group of quadrotors. While the relevant theorems and propositions are included here for completeness, we refer the reader to [Ataei and Paschalidis, 2015] for their proof.

Let $U_i$ be the total output of the rotors for pursuer $i$ and define $U^*$ to be the total output required for hovering. We denote the control inputs to the translational sub-system corresponding to the desired roll and pitch angles of the pursuer $i$ by $\phi_{d_i}, \theta_{d_i}$, respectively. We treat the desired yaw angle as a parameter set by the user, and without loss of generality set it to zero. Let $\xi_i = (p_i, h_{p_i})$, where $p_i$ denotes the $xy$-coordinates of pursuer $i$ and $h_{p_i}$ denotes its altitude. Define $\hat{x}_i = (\xi_i, \dot{\xi}_i)$ and $\hat{u}_i = (U_i, \phi_{d_i}, \theta_{d_i})$ to be the translational state space and control input vectors.

Suppose $e^*_i$ is the dummy evader for pursuer $i$ as defined in Section 5.2 and set $\xi^*_i = (e^*_i, h_e)$. Let $x^*_i = (\xi^*_i, 0_3)$ be a hovering position with $u^* = (U^*, 0, 0)$. A discrete-time linear model of the translational sub-system can be derived by linearizing the nonlinear system dynamics around $x^*$ and $u^*$, which yields [Ataei and Paschalidis, 2015]

$$x_i(k+1) = A_i x_i(k) + B_i u_i(k),$$

where $x_i = \hat{x}_i - x^*_i$ and $u_i = \hat{u}_i - u^*$. The state and input matrices in (5.8) are given by,
\[ A_i = \begin{bmatrix} I_3 & Ts I_3 \\ 0_3 & I_3 \end{bmatrix}, \quad B_i = Ts \begin{bmatrix} 0_3 \\ 0 & 0 \\ 0 & g \\ 0 \end{bmatrix}, \]

where \( T_s \) is sampling time and \( g \) is gravitational acceleration.

The linearized system in (5.8) is subject to modeling errors induced by system linearization. These effects become especially dominant when the quadrotor performs more aggressive maneuvers with large roll and pitch angles. Furthermore, the uncertainty caused by disturbance and measurement errors in \( p_i \) and \( e \) can also be captured as uncertainties in the system. We can therefore treat the system as a linear time-variant system with uncertain \( A_i \) and \( B_i \) matrices. Since the algorithm requires each pursuer to remain inside its safety ellipsoid, we wish to construct a controller which not only achieves stability but also guarantees that a pursuer trajectory never leaves its corresponding safety ellipsoid even in the presence of uncertainties. To this end, we use the following robust MPC technique proposed in our collaborator’s earlier work in [Ataei and Paschalidis, 2015] and include it here for completeness.

### 5.4.1 Robust Model Predictive Control

To improve the readability throughout this section, we use superscript \( \kappa \) to denote a variable corresponding to pursuer \( \kappa \). Consider the discrete-time linear time-variant system in (5.8) for pursuer \( \kappa \) with the output vector \( y^\kappa \) defined as,

\[
\begin{align*}
    x^\kappa(k+1) &= A^\kappa(k)x^\kappa(k) + B^\kappa(k)u^\kappa(k), \\
    y^\kappa(k) &= C^\kappa x^\kappa(k),
\end{align*}
\]

(5.9)

where \( x(k) \in \mathbb{R}^{n_x} \), \( u \in \mathbb{R}^{n_u} \) and \( y \in \mathbb{R}^{n_y} \). Let \( D^\kappa \triangleq [A^\kappa | B^\kappa] \) and define the uncertainty set as

\[
\mathcal{U} \triangleq \{ D \in \mathbb{R}^{n_x \times (n_x+n_u)} | |D^\kappa_{ij} - \bar{D}_{ij}| \leq \Delta^\kappa_{ij}, \forall i, j \},
\]

(5.10)
where $\bar{D} = [\bar{A} | \bar{B}]$ denotes the matrix corresponding to the nominal values of $A^\kappa$ and $B^\kappa$, and $\Delta^\kappa = [\Delta^\kappa_{ij}]$ is a matrix with non-negative elements.

Let $\delta$ be the vector containing the non-zero elements of $\Delta^\kappa$. For the $l$-th element of $\delta$, let $i_{\delta_l}$ and $j_{\delta_l}$ denote the row and column index of $\delta_l$ in $\Delta$, respectively (i.e. $\Delta_{i_{\delta_l}j_{\delta_l}} = \delta_l$). Define $U^\kappa(i, j, \Delta^\kappa_{ij})$ to be a matrix whose $ij$-th element is set to $\Delta^\kappa_{ij}$ and the rest are zero. We can now write $D$ as,

$$
D^\kappa = \bar{D} + \sum_{l=1}^{n_\delta} \zeta_l U^\kappa(i_{\delta_l}, j_{\delta_l}, \Delta^\kappa_{i_{\delta_l}j_{\delta_l}}),
$$

(5.11)

where $|\zeta_l| \leq 1$. Note (5.11) effectively covers the set of uncertain $A^\kappa$ and $B^\kappa$ matrices with polytopic uncertainties.

Let $x^\kappa(k+i|k)$ be an estimate of $x^\kappa$ at sampling time $k+i$ based on the measurements obtained at time $k$. For brevity, we denote $x^\kappa(k+i|k)$ for all $i \geq 0$ by $x^\kappa(k+i)$ and extend the same notation for other variables. Consider an MPC problem with an infinite prediction horizon, where at each sampling time $k$, a control law $u^\kappa(k+i) (i \geq 0)$ is designed to solve the min-max optimization problem

$$
\min_{u^\kappa(k+i), i \geq 0} \max_{(A^\kappa, B^\kappa) \in \mathcal{U}} J^\kappa_\infty(k),
$$

(5.12)

where $Q_\infty \succ 0$, and $R_\infty \succeq 0$.

Suppose there exists a feedback control law $u^\kappa(k+i) = F^\kappa x^\kappa(k+i)$ and a quadratic function $V^\kappa(x) = (x^\kappa)^T P^\kappa x^\kappa$ with $P > 0$ such that for all $i \geq 0$,

$$
V^\kappa(x^\kappa(k+i)) - V^\kappa(x^\kappa(k+i+1)) \geq x^\kappa(k+i)^T Q_\infty x^\kappa(k+i) + u^\kappa(k+i)^T R_\infty u^\kappa(k+i).
$$

(5.14)

Then, $V^\kappa(x^\kappa(k))$ is a Lyapunov function for the optimization problem (5.13) [Ca-
macha and Alba, 2013]. Note that the control law should satisfy (5.14) for all \((A^K, B^K) \in U^K\). This, in general, will require solving the optimization problem over all vertices of \(U^K\) which grow exponentially with the dimension of \(U^K\). In [Ataei and Paschalidis, 2015], the authors show an alternative approach where the size of the problem grows linearly with the dimension of \(U^K\).

**Theorem 6** ([Ataei and Paschalidis, 2015]). Consider the uncertain discrete system in (5.9), where uncertainties are defined by (5.10). Let \(u^K(k + i) = F^K x^K(k + i)\) be the control action for time \(k + i\) for all \(i \geq 0\). Consider the following LMI problem:

\[
\begin{aligned}
\min_{\gamma^K, Q^K, G^K, Z^K_{\delta l} \mid l = 1, \ldots, n_\delta} & \quad \gamma^K \\
\text{s.t.} & \quad \begin{bmatrix} 1 & x^K(k) & Q^K \end{bmatrix} \succeq 0, \\
& \quad L^K_0 \succeq 0, \\
& \quad \sum_{l=1}^{n_\delta} Z^K_{\delta l} \succeq L^K_0, \\
& \quad Z^K_{\delta l} \succeq L^K_{\delta l}, \quad Z^K_{\delta l} \succeq -L^K_{\delta l}, \quad \forall l = 1, 2, \ldots, n_\delta
\end{aligned}
\]  

(5.15)

where \(Z^K_{\delta l} = (Z^K_{\delta l})^T\),

\[
L^K_0 = \begin{bmatrix}
AQ^K + BG^K & Q^K & * & * & * \\
Q_{\infty}^2 & 0 & \gamma^K I & * & * \\
R_{\infty}^2 & 0 & 0 & \gamma^K I
\end{bmatrix},
\]

\[
L^K_{\delta l} = \begin{bmatrix}
A^K_{\delta l} Q^K + B^K_{\delta l} G^K & 0 & * & * & * \\
0 & 0 & * & * & * \\
0 & 0 & 0 & 0
\end{bmatrix},
\]

\(A^K_{\delta l}\) and \(B^K_{\delta l}\) are matrices with the same dimensions as \(A^K\) and \(B^K\), respectively, whose values are extracted from \(U^K(i_{\delta l}, j_{\delta l}, \Delta^K_{i_{\delta l}, j_{\delta l}}) = [A^K_{\delta l} \ B^K_{\delta l}]\) in (5.11). If the above optimization problem has a solution, then \(J^K_\infty\) in the worst-case scenario is bounded from above by \(\gamma^K\) and the minimizing feedback control gain is given by,

\[
F^K = G^K (Q^K)^{-1}.
\]

(5.16)
While the above theorem guarantees stability of the quadrotor system for the given uncertainty set, it does not ensure that the trajectories will always remain inside the safety ellipsoid. Let $Q^\kappa_\varepsilon$ be the safety ellipsoid for pursuer $\kappa$ calculated from (5.2). The following proposition guarantees that the pursuer trajectory does not leave the safety ellipsoid.

**Proposition 6** (Output Constraints [Ataei and Paschalidis, 2015]). Consider the uncertain discrete system in (5.9), the feedback control law $u^\kappa(k+i) = F^\kappa x^\kappa(k+i)$, where $F^\kappa$ is given by (5.16), and the safety ellipsoid $Q^\kappa$ in (5.1). Let $C = [I_2 | 0_{2\times4}]$. The pursuer $\kappa$ is guaranteed to remain in the safety ellipsoid $Q^\kappa$ if the following LMI feasibility problem has a solution,

$$
\begin{align*}
\text{find } & \tilde{Z}_{\delta_l}^\kappa, l = 1, \ldots, n_{\delta}, \\
\text{s.t. } & M_0^\kappa \succeq 0, \\
& \sum_{l=1}^{n_{\delta}} \tilde{Z}_{\delta_l}^\kappa \preceq M_0^\kappa, \\
& \tilde{Z}_{\delta_l}^\kappa \succeq M_{\delta_l}^\kappa, \quad \tilde{Z}_{\delta_l}^\kappa \succeq -M_{\delta_l}^\kappa,
\end{align*}
$$

where $\tilde{Z}_{\delta_l}^\kappa = (\tilde{Z}_{\delta_l}^\kappa)^T$, $l = 1, \ldots, n_{\delta}$, and

$$
\begin{align*}
M_0^\kappa &= \begin{bmatrix} Q^\kappa & (Q^\kappa_\varepsilon)^{-1} \\ C(AQ^\kappa + BG^\kappa) & 0 \end{bmatrix}, \\
M_{\delta_l}^\kappa &= \begin{bmatrix} 0 & * \\ C(A_{\delta_l}^\kappa Q^\kappa + B_{\delta_l}^\kappa G^\kappa) & 0 \end{bmatrix}.
\end{align*}
$$

To ensure stability and safety conditions are satisfied simultaneously, we add the constraints in Proposition 6 to Theorem 6 and solve the resulting optimization problem.

### 5.5 Experiments

In this section, we present experimental results demonstrating our proposed tracking algorithm. For the pursuers, we used three KMEL Nano+ quadrotors equipped with
Kbee radios for communication. Localization was performed with NaturalPoint’s OptiTrack system. Matlab was used to perform all calculations, and the updated waypoints were transmitted to the quadrotors. Two scenarios are presented. The first experiment uses a simulated evader following a pre-planned trajectory. The pursuers do not know the pre-planned trajectory, but react to the evader in real time. The second experiment uses a Pololu m3pi robot manually driven with a joystick, and pursuers react in real time. A video with both experimental runs can be viewed on the MSL website\textsuperscript{1}. In both experiments, we see the pursuers successfully track the evader as it moves in and out of the no-fly zones.

5.5.1 Simulated Evader

In the first experiment, three quadrotors track a simulated evader as it moves throughout the environment. The trajectories can be seen in Figure 5·3(a), with the evader shown in red and final positions denoted by the circles. Over time, we see that the trajectories of the pursuers remain outside the no-fly zones, demonstrating a successful implementation of the path-planning algorithm. Figure 5·3(b) shows the minimum distance from any pursuer to the evader over time. The shaded areas indicate when the evader was within a no-fly zone, and the red dashed line corresponds to the maximum distance the evader could achieve given its entry point and maximum velocity. By employing the centroidal Voronoi algorithm, we find that the minimum distance remains relatively small, despite the pursuers not knowing the true evader position. By the end of the experiment, the pursuers are within 10 cm of the evader. Figure 5·4 shows stills from the experimental video demonstrating the Voronoi-based coverage strategy while the evader is in the no-fly zone. Over time, we see the pursuers spread out as $R_j$ grows. In the final frame, the evader emerges from the top of the no-fly zone, with a pursuer nearby.

\textsuperscript{1}http://msl.stanford.edu
Figure 5·3: (a) Trajectories of the pursuers and evader (red). (b) Min. distance from any pursuer to the evader. The red dashed line shows the maximum possible distance to the evader inside the no-fly zone. Our strategy keeps the distance well below the max.
Figure 5.4: Stills from the experimental video illustrating the control strategy while an evader is in a no-fly zone. The pursuers (yellow squares) arrange themselves around the boundary, waiting for the evader to emerge in (d).
5.5.2 Human-Controlled Evader

For our second experiment, we controlled an m3pi robot with a joystick to create a “live” evader for our three quadrotors to track. The trajectories over time are shown in Figure 5-5(a). Again, we see the pursuers never enter the no-fly zone. Figure 5-5(b) plots the minimum distance from any pursuer to the evader, as well as the maximum possible distance in red. Our Voronoi-based control strategy keeps the distance to the evader relatively small, even though its position is unknown. Stills from the experimental video are shown in Figure 5-6. As with the simulated evader, we see the pursuers distributing themselves around the boundary of $Z_j$ while the evader remains inside the no-fly zone.
Figure 5.5: (a) Trajectories of the pursuers and evader (red). (b) Min. distance from any pursuer to the evader.
Figure 5.6: Stills from the experimental video illustrating the no-fly-zone planning. The pursuers are denoted by the yellow squares, and the evader is circled in red.

5.6 Summary

This chapter presents a series of algorithms to coordinate a group of pursuers tracking an evader while avoiding no-fly zones. While the evader remains outside the no-fly zone, we assume the pursuers know the evaders position. Once the evader enters a no-fly zone, an estimate of the reachable set of all evader positions is generated based on the entry point and the maximum velocity of the evader. Each pursuer is then
assigned to the centroid of a Voronoi cell, which is used to distribute the pursuers about the zone’s boundary. Through experiments, we show that as a result of the coordinated pursuit, the quadrotors remain in close proximity of the evader even when it enters a no-fly zone.
Chapter 6

Guaranteed Capture of Multiple Evaders with Multiple Pursuers

In this chapter, we examine the problem of capturing multiple evaders in a convex, bounded environment. In the previous chapter, the goal of the pursuers was to track an evader while avoiding obstacles, but not necessarily capture it. Here, the pursuers attempt to capture all evaders in the environment without obstacles. Our approach finds a control policy for the pursuers that guarantees capture in finite time of all evaders, regardless of the evaders’ strategy. The pursuers first compute a Voronoi tessellation with the evaders, and then move to the midpoint of the shared Voronoi boundary between the pursuer and targeted evader. We present a global algorithm that guarantees capture of all evaders in finite time, as well as a decentralized heuristic more suited for implementation. Performance is verified for 2D and 3D environments in simulation, and experiments were conducted with ground robots running the algorithm onboard in real time. The results are currently under review in both Robotics and Automation Letters (RA-L) and for the 2017 ICRA conference [Pierson et al., 2017].

The remainder of the chapter is organized as follows: Section 6.2 defines the multi-pursuer, multi-evader problem. In Section 6.3, we present generalized capture guarantees of a single evader in convex, bounded environments in $\mathbb{R}^N$. Section 6.4 extends these results to the coordinated pursuit of multiple evaders. Simulations of multi-evader pursuit in 2D and 3D environments are presented in Section 6.5. Finally,
results from our decentralized hardware implementation with ground vehicles are presented in Section 6.6.

6.1 Related Work

For multi-agent pursuit-evasion problems, one common approach is to formulate them as a differential game and solve the associated Hamilton-Jacobi-Isaacs (HJI) partial differential equation [Isaacs, 1999]. In this game, the evader(s) attempt to maximize their capture time, while the pursuers minimize the capture time of the evaders. Optimal trajectories with respect to capture time are found by either solving the equations directly or with numerical approximations [Falcone and Ferretti, 2002, Mitchell et al., 2005, Ramana and Kothari, 2015]. In practice, the differential games approach poses several challenges. Many techniques require backwards-solving from a terminal condition, which is difficult to determine from initial conditions. Furthermore, the computational complexity of these methods creates a limitation on the number of agents in the game.

Another approach is to formulate the pursuit-evasion problem on a graph and solve for the movement between nodes [Parsons, 1978]. This approach decomposes the environment to reduce the possible actions of pursuers, greatly reducing the computational complexity. For visibility-based pursuit evasion games, where a pursuer can only target an evader within line-of-sight, decomposing the environment allows the pursuers to explore and locate evaders [Isler et al., 2005, Stiffler and O’Kane, 2016].

In games with multiple pursuers, coordination among the pursuers can lead to a more efficient capture of the evader(s). Several non-optimal techniques have been proposed to coordinate the pursuers. One approach is the “sweep-pursuit-capture” strategy, where pursuers form a chain to simultaneously sweep the environment and
encircle evaders [Bopardikar et al., 2007]. This chain formation can also be used to prevent evaders from escaping narrow environments [Kolling and Carpin, 2010], and is relevant to the problem of clearing environments in search and rescue applications [Hollinger et al., 2010]. For a probabilistic model of the evader, pursuers can employ a greedy pursuit strategy that becomes computationally feasible [Vidal et al., 2002].

Another approach uses Model Predictive Control (MPC) to prevent pursuers from colliding with agents or obstacles in the environment while tracking an evader [Ataei and Paschalidis, 2015, Pierson et al., 2016a]. Chapter 5 presented a cooperative pursuit strategy for quadrotors tracking an evader as it moves through the environment. Here, we focus on guaranteeing capture of the evaders, and there are no obstacles in the environment.

Our work builds upon pursuer-evader problems that leverage Voronoi tessellations in designing the pursuer control strategy. When the evader’s dynamics are known, the pursuers can tessellate the environment and determine an optimal target assignment [Bakolas and Tsiotras, 2010]. For an evader with unknown dynamics, Huang et. al proposed an “area-minimization” policy that decreases the safe-reachable area of a single evader to guarantee capture in the plane [Huang et al., 2011, Zhou et al., 2016]. This area-minimization strategy has been extended to contain evaders in unbounded environments [Pan et al., 2012] or in nonholonomic systems [Kothari et al., 2014]. In this chapter, we extend the “area-minimization” strategy in several ways. First, we present a generalized formulation to capture an evader in $\mathbb{R}^N$. Next, we present a global strategy that guarantees multiple evaders are captured in finite time. Finally, we present a decentralized heuristic for capturing multiple evaders better suited for implementation.
6.2 Problem Formulation

Consider a group of \( n_p \) pursuer and \( n_e \) evader agents in a bounded, convex environment \( Q \subset \mathbb{R}^N \), with points in \( Q \) denoted \( q \). We denote the positions of the pursuer agents as \( x_j^p \in Q \), for \( j \in \{1, \ldots, n_p\} \) and the evader agents as \( x_i^e \in Q \) for \( i \in \{1, \ldots, n_e\} \). The entire group of \( n = n_p + n_e \) agents is denoted \( x^T = [\ldots, x_j^p T, \ldots, x_i^e T, \ldots] = [\ldots, x_k^T, \ldots] \) for \( k \in \{1, \ldots, n\} \). We assume that all agents have integrator dynamics,

\[
\dot{x}_e^i = u_e^i, \quad \|u_e(t)\| \leq v_{\text{max}}, \\
\dot{x}_p^j = u_p^j, \quad \|u_p(t)\| \leq v_{\text{max}},
\]

where \( u_e \) and \( u_p \) are the control inputs moving the pursuers and evaders through the environment, subject to the same maximum speed \( v_{\text{max}} \). Without loss of generality, we let \( v_{\text{max}} = 1 \) for the remainder of this section.

The pursuers’ goal is to capture all evaders in the environment. If the pursuers capture an evader at time \( t_c \), it remains captured for all \( t > t_c \). An evader \( i \) is captured at \( t_c \) when

\[
\min_{j \in n_p} \|x_e^i(t_c) - x_p^j(t_c)\| < r_c,
\]

where \( r_c > 0 \) is the capture radius. The capture radius can be arbitrarily small and defined based on the pursuers and evaders.

While we do not know the evader’s control policy, we know that it can only avoid capture by moving within its safe-reachable set. At any point in time, the safe-reachable set is defined as all points in \( Q \) that an evader \( i \) can reach before any other agent. For agents with equal maximum velocities and integrator dynamics, this set is equal to the Voronoi partition, defined

\[
V_i = \{ q \in Q \| q - x_i \| \leq \| q - x^j \|, \forall j \neq i, \quad i, j \leq n \}.
\]
We define Voronoi neighbors as agents that share a Voronoi boundary, and denote the set of neighbors $N_i$. In the following sections, we present the pursuer’s area-minimization strategy that reduces the evader’s reachable set until it is captured.

6.3 Pursuit of a Single Evader

In [Huang et al., 2011, Zhou et al., 2016], the authors propose their “area-minimization strategy” for the pursuit of a single evader by multiple pursuers in $\mathbb{R}^2$. They prove that by driving to the midpoint of the shared Voronoi boundary between a pursuer and evader, the pursuer is guaranteed to capture the evader in finite time. Here, we present our proof of guaranteed capture of a single evader for environments in $\mathbb{R}^N$. Consistent with previous work, we find that the pursuers will drive to the centroid of the shared Voronoi boundary with the evader.

To prove guaranteed capture in finite time, we present our proof in a similar structure to [Huang et al., 2011]. Despite these parallels in structure, our derivation and proof techniques are different in that we represent edges and center points, facets and centroids, etc, in a general integral form that is independent of dimension. For consistency with existing literature, we denote $A_e$ as the safe-reachable area of the evader and the pursuers’ strategy as the “area-minimization” policy for $\mathbb{R}^N$.

6.3.1 Area-Minimization Policy

This section described the area-minimization strategy for the pursuers from [Huang et al., 2011], but re-cast in our integral notation. The safe-reachable area of an evader, $A_e$, is defined as the points in $Q$ that the evader can reach before any other agent. For agents with equal speeds, this reduces to the Voronoi cell, $V_e$, of the evader. The area $A_e$ is calculated

$$A_e = \int_{V_e} dq,$$  \hspace{1cm} (6.1)
where $V_e$ is the Voronoi cell of the evader in $\mathbb{R}^N$. The dynamics of $A_e$ are

$$\dot{A}_e = \frac{\partial A_e}{\partial x_e} \dot{x}_e + \sum_{j=1}^{n_p} \frac{\partial A_e}{\partial x_p} \dot{x}_p^j.$$ 

The pursuers choose a strategy that will decrease the area over time. From this formulation, we can decouple the pursuers into their individual contributions $\frac{\partial A_e}{\partial x_p} \dot{x}_p^j$.

For each pursuer, let

$$u_p^j = -\frac{\frac{\partial A_e}{\partial x_p}}{\left\| \frac{\partial A_e}{\partial x_p} \right\|}.$$ 

This policy follows the gradient of $A_e$, moving in the direction of the fastest decrease of the area. We refer to this strategy as the “area-minimization” strategy for consistency with existing literature. Note that in 2D this will be an area, and in 3D $A_e$ represents a volume.

**Lemma 2.** For the evader $x_e$ and its safe-reachable area $A_e$ in (6.1) and pursuer $x_p^j$, the gradient $\frac{\partial A_e}{\partial x_p}$ is equivalent to

$$\frac{\partial A_e}{\partial x_p^j} = \frac{L_j}{\|x_p^j - x_e\|} \left( x_p^j - C_{b_j} \right).$$

where $b_j$ is the shared Voronoi boundary between $x_p^j$ and $x_e$, $L_j$ is the area of the boundary, and $C_{b_j}$ is the centroid of the boundary.

**Proof.** For the evader area $A_e$ defined in (6.1), using Leibniz Integral Rule, the derivative of $A_e$ reduces to

$$\dot{A}_e = \sum_{j \in \mathcal{N}_e, b_j} \int \left[ \frac{(x_p^j - q)^T \dot{x}_p^j}{\|x_p^j - x_e\|} - \frac{(x_e - q)^T \dot{x}_e}{\|x_p^j - x_e\|} \right] dq,$$

where $\mathcal{N}_e$ is the set of the evader’s Voronoi neighbors and $b_j$ is the Voronoi boundary between the evader and pursuer $j$. Define

$$L_j = \int_{b_j} dq, \text{ and } C_{b_j} = \frac{1}{L_j} \int_{b_j} q dq,$$
noting that $L_j > 0$ and $C_{b_j}$ is the centroid of the boundary $b_j$. For 2D environments, $b_j$ is a line and $L_j$ is the length of the line, while in 3D environments, $b_j$ is a face and $L_j$ is the area. Thus, $\dot{A}_e$ reduces to

$$\dot{A}_e = \sum_{j \in N_e} \frac{L_j (x^j_p - C_{b_j})^T}{\|x^j_p - x_e\|} \dot{x}^j_p - \sum_{j \in N_e} \frac{L_j (x^j_e - C_{b_j})^T}{\|x^j_p - x_e\|} \dot{x}_e$$

$$= \sum_{j \in N_e} \frac{\partial A_e}{\partial x^j_p} \dot{x}^j_p + \frac{\partial A_e}{\partial x_e} \dot{x}_e. \tag{6.4}$$

Using Lemma 2 and plugging (6.3) into (6.2) the pursuer control policy reduces to

$$u^j_p = \frac{(C_{b_j} - x^j_p)}{\|C_{b_j} - x^j_p\|}. \tag{6.4}$$

Note this policy directs the pursuers towards the centroid of the shared Voronoi boundary between the pursuer and the evader. In 2D, the centroid is equivalent to the midpoint of the shared Voronoi boundary edge (as found by different means in [Huang et al., 2011, Zhou et al., 2016]), and in 3D, it is the centroid of the shared Voronoi boundary face.

### 6.3.2 Proof of Guaranteed Capture

The pursuers’ strategy in (6.4) is designed to decrease the evader’s safe-reachable area $A_e$. To prove this guarantees capture, we parallel the proof technique from [Huang et al., 2011, Zhou et al., 2016]. We first show that under our pursuer strategy $A_e$ is always non-increasing. When $\dot{A}_e = 0$, we show the distance between the pursuer and evader is strictly decreasing. We then show that bounds on these dynamics lead to guaranteed capture in finite time. For the remainder of this section, we use a single pursuer in our proof of guaranteed capture, and it is easily seen that the results hold with multiple pursuers.
Lemma 3. Consider a single pursuer, single evader in $Q$. Under the proposed pursuer strategy (6.4), the area $A_e$ satisfies $\dot{A}_e \leq 0$ for any admissible evader control strategy. Furthermore, $\dot{A}_e = 0$ if and only if the evader uses the following controller:

$$u_e^* = \frac{(C_b - x_e)}{\|C_b - x_e\|}, \quad (6.5)$$

where $C_b$ is the centroid of the shared Voronoi boundary between pursuer $x_p$ and evader $x_e$.

**Proof.** For a single pursuer, single evader scenario, the dynamics of $A_e$ reduce to

$$\dot{A}_e = \frac{L}{\|x_p - x_e\|} \left[ (x_p - C_b)^T \dot{x}_p - (x_e - C_b)^T \dot{x}_e \right], \quad (6.6)$$

where $L$ is the area of the shared boundary $b$ between the pursuer and evader. Plugging in (6.4) into (6.6),

$$\dot{A}_e = \frac{L}{\|x_p - x_e\|} \left[ -\|x_p - C_b\| - (x_e - C_b)^T \dot{x}_e \right].$$

We see that for this single pursuer case, $\dot{A}_e \leq 0$ and furthermore, for $\dot{A}_e = 0$, we find

$$\dot{x}_e = \frac{(C_b - x_e)}{\|C_b - x_e\|}.$$

By Lemma 3, the only evader policy to keep the area constant is to move towards the shared centroid $C_b$ of the Voronoi boundary. Define $z$ as the distance between the pursuer and evader,

$$z = \|x_p - x_e\|^2 = (x_p - x_e)^T (x_p - x_e).$$

The following proves that $z$ is strictly decreasing when $\dot{A}_e = 0$.

**Lemma 4.** For the pursuit strategy, (6.4), if $\dot{A}_e = 0$, then

$$\dot{z} = -\frac{2\|x_p - x_e\|^2}{\|C_b - x_p\|} < 0.$$


Proof. The dynamics of $z$ are

$$
\dot{z} = 2 (x_p - x_e)^T (\dot{x}_p - \dot{x}_e).
$$

For $\dot{A}_e = 0$, by Lemma 3, we know that the evader dynamics are given by (6.5). Since $C_b$ exists on the shared Voronoi boundary, $\|C_b - x_p\| = \|C_b - x_e\|$. Plugging in our pursuer strategy (6.4) and evader strategy (6.5),

$$
\dot{z} = 2 (x_p - x_e)^T \left( \frac{C_b - x_p - C_b - x_e}{\|C_b - x_p\|} \right),
$$

$$
= -\frac{2\|x_p - x_e\|^2}{\|C_b - x_p\|}.
$$

By Lemmas 3 and 4, we have that the evader’s area $A_e$ is non-increasing, and that when $\dot{A}_e = 0$, $z$ is strictly decreasing. However, there is a possibility that although $A_e$ may be decreasing, $z$ increases and remains within the range $[r_c^2, \ell_{\text{max}}^2]$, where $\ell_{\text{max}}$ is the maximum distance between any two points in $Q$. If the evader were to remain in this range, it would never be captured. The following lemma proves this cannot happen.

**Lemma 5.** Under the pursuer strategy $u_p$ in (6.4), if $\dot{A}_e \geq -\beta$ for some constant $\beta > 0$, then $\dot{z} \leq -f(\beta)$, where $f(\beta)$ is given by

$$
f(\beta) = \ell_{\text{max}} - \frac{\ell_{\text{max}}}{\ell_{\text{min}}} \beta.
$$

Proof. First, examine the case when $\dot{A}_e \geq -\beta$, thus

$$
-(x_e - C_b)^T \dot{x}_e \geq \frac{-\beta\|x_p - x_e\|}{L} + \|C_b - x_p\|.
$$

Rearranging this expression, we find

$$
(x_e - x_p)^T \dot{x}_e \leq \frac{-\beta\|x_p - x_e\|}{L} \leq \frac{\ell_{\text{max}}}{\ell_{\text{min}}} \beta,
$$

where $\ell_{\text{min}}$ is a lower bound on $L$, defined by the geometry of $Q$. Substituting this
expression into \( \dot{z} \),

\[
\dot{z} \leq \frac{\ell_{\text{max}}}{\ell_{\text{min}}} \beta - (x_p - x_e)^T \dot{x}_p \leq \frac{\ell_{\text{max}}}{\ell_{\text{min}}} \beta - \ell_{\text{max}}.
\]

\[\Box\]

Lemma 5 also implies that when \( \dot{A}_e < -\beta \), then \( \dot{z} > f(\beta) \). We assume that \( \beta \) is chosen such that \( f(\beta) > 0 \). We now present our theorem proving that under the pursuer strategy (6.4), the evader is captured in finite time. Define an “cost-to-capture” function of the system,

\[
E = kA_e + z,
\]

(6.7)

where \( k = \frac{4\ell_{\text{max}} + f(\beta)}{\beta} > 0 \). For capture to occur, \( E = 0 \), either as \( A_e \) or \( z \) goes to zero.

**Theorem 7.** For the cost-to-capture function in (6.7) and pursuer strategy (6.4), if capture has not occurred before time \( t_0 \), then for \( t > t_0 \),

\[
E(t) < E(t_0).
\]

Proof. Lemma 5 gives us the following conditions, which must be true at any given time:

**Condition 1:** \( \dot{A}_e \geq -\beta \) and \( \dot{z} \leq -f(\beta) \), or

**Condition 2:** \( \dot{z} > f(\beta) \) and \( \dot{A}_e < -\beta \).

The derivative of \( E \) is

\[
\dot{E} = k\dot{A}_e + \dot{z},
\]

and we know \( \dot{A}_e \leq 0 \). Under Condition 1, \( \dot{E} \leq -f(\beta) \). Under Condition 2, we see that \( \dot{A} < -\beta \) and \( \dot{z} > f(\beta) \), however, since the agents are restricted by maximum speeds, \( \dot{z} \leq 4\ell_{\text{max}} \). Thus, for \( k = \frac{4\ell_{\text{max}} + f(\beta)}{\beta} \),

\[
\dot{E} < -k\beta + 4\ell_{\text{max}} < -f(\beta),
\]

which implies the cost to capture decreases to zero in finite time, ensuring capture of the evader. \[\Box\]
6.4 Extension to Multiple Evaders

Next, we present our algorithms for extending the cooperative pursuit to multiple evaders in the environment. In Section 6.3, we presented the area-minimization strategy to reduce the safe-reachable area of a single evader, which guarantees capture in finite time. Here, we present a global area-minimization strategy that guarantees the capture of all evaders in finite time. We also present a decentralized version of the algorithm that we later implement in hardware.

Let \( A_{e_i} \) refer to the safe-reachable area of a single evader, and

\[
A_e = \sum_{i=1}^{n_e} A_{e_i},
\]

be the safe-reachable area of all evaders in the environment. Its derivative is

\[
\dot{A}_e = \sum_{i=1}^{n_e} \frac{\partial A_{e_i}}{\partial x_i} \dot{x}_i + \sum_{i=1}^{n_e} \sum_{k \in \mathcal{N}_{e_i}} \frac{\partial A_{e_i}}{\partial x_k} \dot{x}_k + \sum_{j=1}^{n_p} \sum_{i \in \mathcal{N}_{p_j}} \frac{\partial A_{e_i}}{\partial x_j} \dot{x}_j.
\]

6.4.1 Naive Approach

When designing the pursuer control policy, it may be tempting to extend the results from Section 6.3 and allow the pursuers to minimize the area of all neighboring evaders, ie,

\[
u_p^j = -\sum_{i \in \mathcal{N}_{p_j}} \frac{\partial A_{e_i}}{\partial x_i}.
\]

Under this policy, the pursuer attempts to simultaneously reduce all neighboring evaders’ safe areas. For the single-evader case, this control policy guaranteed capture of the evader in finite time. However, in the multi-evader case, this policy cannot guarantee capture, and generally performs poorly. Consider when the evaders are arranged symmetrically around the pursuer, then the pursuer can be caught in a “symmetry trap” as the components of \( u_p^j \) sum to zero. Figure 6.1 illustrates a
symmetry trap scenario wherein the pursuers fail to capture any evaders.

**Figure 6.1:** Symmetry trap when pursuers (triangles) attempt to minimize the area of all evaders (circles). The pursuers begin with an initial configuration (a), but get stuck in a final configuration (b).
6.4.2 Global Policy with Guaranteed Capture

To avoid the symmetry trap, we propose a policy, requiring global information, in which pursuers target their nearest evader, and coordinate with other pursuers that share the same target. This effectively turns the multi-evader problem into parallel single-evader pursuit problems. Using this strategy, we can guarantee the capture of all evaders in finite time. Algorithm 6 presents a high-level overview of the strategy.

**Algorithm 6 Global Multi-Evader Pursuit**

1. Calculate nearest evader $x_e^\kappa$
2. Coordinate with other pursuers to determine $P_\kappa$
3. while $(\min_{j \in P_\kappa} \| x_p^j - x_e^\kappa \|) > r_e$ do
   4. Compute new Voronoi tessellation $\bar{V}_\kappa$ (6.8)
   5. Pursue $x_e^\kappa$ using (6.9)
4. end while
5. Once $x_e^\kappa$ is captured, update target

Instead of pursuing all neighboring evaders, each pursuer targets only its nearest evader. For a targeted evader $\kappa$, denote the set of pursuers assigned to that target as $P_\kappa$. Let $\bar{V}_\kappa$ denote the Voronoi partition between evader $\kappa$ and its pursuers, defined

$$\bar{V}_\kappa = \{ q | \| x_e^\kappa - q \| \leq \| x_p^j - q \|, \ j \in P_\kappa \}. \quad (6.8)$$

The pursuers then use $\bar{V}_\kappa$ in their control policy,

$$u_p^j = \frac{(C_{b_{\kappa j}} - x_p^j)}{\| C_{b_{\kappa j}} - x_p^j \|}, \quad (6.9)$$

where $b_{\kappa j}$ is the Voronoi boundary between $x_e^\kappa$ and $x_p^j$ in $\bar{V}_\kappa$ and $C_{b_{\kappa j}}$ is the centroid of that shared boundary. Note that multiple pursuers may target the same evader, and there may be evaders that are initially not targeted, but this does not prevent the capture of all evaders. Proposition 7 formalizes this notion.

**Proposition 7.** By Algorithm 6, using control law (6.9), the pursuers can capture all evaders in finite time.
Proof. First, consider an evader $\kappa$ initially targeted by pursuers $j \in P_\kappa$. To calculate (6.10), pursuer $j$ ignores all other evaders $i \neq \kappa$ and the pursuers $j \notin P_\kappa$ to calculate $V^\kappa$. This becomes a single-evader sub-problem for $j \in P_\kappa$ pursuing evader $\kappa$. By Theorem 7, pursuers $j \in P_\kappa$ will capture evader $\kappa$ in finite time.

Now consider an evader that is not initially targeted by any pursuers. Since all targeted evaders will be captured in finite time, there exists some time $\tau < \infty$ that a non-targeted evader will be targeted. Thus, all evaders in the environment will be captured in finite time. \hfill \Box

In designing our global algorithm, we do not claim it is optimal, only that it guarantees capture. The only known method to find an optimal strategy in this setting is HJI, known to be intractable for large numbers of agents.

6.4.3 Decentralized Local Policy

In the previous section, we proposed a global policy for the pursuers that guarantees capture of all evaders in finite time. However, this policy requires coordination and communication among all pursuers, making it difficult to implement. Here, we present a decentralized, local heuristic more suited for implementation that performs comparably to the global algorithm in practice.

Instead of requiring pursuers to calculate $\bar{V}_\kappa$ for $u_j^{p}$, consider instead that each pursuer calculates its Voronoi cell based on all agents, which can be computed in a decentralized fashion [Cortes et al., 2004]. From $V_j$, the pursuer then chooses the nearest evader from its Voronoi neighbors as its target $x^\kappa_e$. Overall, the control policy is

$$u_p^j = \frac{(C_{b_{\kappa j}} - x_j^p)}{\|C_{b_{\kappa j}} - x_j^p\|}$$

where $C_{b_{\kappa j}}$ is the centroid of the shared Voronoi boundary between $x^\kappa_e$ and $x_j^p$. If there are no evader Voronoi neighbors, then the pursuer moves directly towards the nearest evader in the environment. Note that there will always be at least one pursuer with an evader neighbor, so seeking non-neighbor evader targets is not a requirement.
There are two key differences between the decentralized local policy and the global policy:

1. Pursuer $x_p^j$ calculates $V_j$ and $C_{b,j}$ using all agents in the environment.

2. Pursuers are allowed to switch targets over time.

These two relaxations of the policy ensure that at every time step, a pursuer $j$ only needs local information about its Voronoi neighbors to compute its control law. Furthermore, since it includes all nearby pursuers in calculating $V_j$, the pursuer policies are cooperative with one another. Under this heuristic, we cannot guarantee capture in finite time, however, in all simulations and experiments we have observed that capture is achieved. We know of one theoretical counter-example for circular environments when all agents start equally spaced on a constant radius and move uniformly, creating a symmetry trap. This appears to be highly unlikely in practice, and we have not found counter-examples for non-circular environments. Simulations demonstrate its performance for a variety of randomized configurations. Algorithm 7 summarizes the main steps.

**Algorithm 7 Local Area-Minimization Policy**

1: Calculate Voronoi tessellation with all agents
2: Determine nearest evader from Voronoi neighbors, $e^*$
3: Compute $\dot{x}_\beta^j$ (6.10)
4: If no neighbor evader exists, move directly towards nearest evader

### 6.5 Simulations

Simulations were performed in Matlab to verify the behavior of our algorithm. We also compare our performance to a baseline strategy wherein the pursuer directly chases the evader [Alexander et al., 2006]. Although our algorithm works for any evader policy, we need to implement an evader control for simulations. If the evaders
simply try to run away from the pursuers, they will get trapped against a wall. If the evader attempts to maximize its area \( A_e \), by Lemma 3 that drives the evader directly towards the pursuer (clearly a bad idea!). Similarly, an area-minimization policy only helps the pursuers. With these considerations, based on a comparison of several policies, we choose a “move-to-centroid” control, common in Voronoi-based multi-agent literature [Cortes et al., 2004]. Here,

\[
  u^i_e = \frac{(C_{V_i} - x^i_e)}{\|C_{V_i} - x^i_e\|},
\]

where \( V_i \) is the Voronoi cell of the evader calculated using all agents, and \( C_{V_i} \) is the centroid. Intuitively, this policy drives the evader away from edges of its cell, balancing the threat of neighboring pursuers and avoiding environmental boundaries. We choose this policy for its simple, decentralized nature as well as its natural threat-avoidance properties. In Section 6.6, we also test our policy against a human-controlled evader, which has superior planning abilities.

### 6.5.1 Multi-Evader Pursuit in 2D

Figure 6·2 shows a simulation with \( n_p = 4 \) pursuers and \( n_e = 8 \) evaders over time. The pursuers are using the decentralized area-minimization policy from Algorithm 7, which targets their nearest Voronoi evader neighbor. In Figure 6·2, the evader’s safe-reachable area is shaded in blue. Over time, we see the evader’s area shrinking, with successful capture of all agents by the end of the simulation. Figure 6·2f plots the minimum distance an evader is to any pursuer over time, with the black horizontal line showing the capture radius. Although this value may temporarily increase, over time every evader is captured.
Figure 6.2: (a)-(e) Simulation of 4 pursuers (triangles) and 8 evaders (red circles), with their safe-reachable area shaded in blue. Captured evaders are marked as x’s. (f) Minimum distance of evaders to any pursuer. The black line marks the capture radius.
6.5.2 Multi-Evader Pursuit in 3D

Our generalized problem formulation presented in Section 6.2 allows us to extend the algorithm to 3D. Here, the safe-reachable area is the volume of the 3D Voronoi cell, and the pursuers target the centroid of the shared boundary face. Figure 6.3 shows a 3D simulation of $n_p = 4$ and $n_e = 2$ evaders. The Voronoi cells of the evaders are shaded in blue, showing the total volume of the evader’s safe-reachable area. The pursuers use the decentralized heuristic from Algorithm 7, and over time, successfully capture both evaders.
Figure 6.3: (a)–(e) Simulation of 4 pursuers (triangles) and 2 evaders (circles), with their safe-reachable area shaded in blue and captured evaders denoted as x’s. (f) Minimum distance of evaders to any pursuer over time. The black line marks the capture radius.
6.5.3 Comparison to Other Methods

To analyze the performance of our global and local area-minimization policies, we conducted trials across a variety of scenarios in 2D and 3D. As a baseline, we compare these policies to two other comparable deterministic pursuit strategies. A summary of all pursuer policies is given in Table 6.1. We do not compare to HJI-based strategies, which are intractable for large groups.

The first baseline strategy we call the “Direct Charge (DC)” strategy. Here, a pursuer drives directly towards its nearest evader, \( e^\kappa \), guaranteeing capture in closed, simply connected domains [Alexander et al., 2006]. For a target \( \kappa \) assigned by

\[
\kappa = \arg \min_{i \in n_e} \| x^j_p - x^i_e \|,
\]

the pursuer control law is

\[
u^j_p = \frac{(x^\kappa_e - x^j_p)}{\| x^\kappa_e - x^j_p \|}.
\] (6.12)

A limitation of this policy is a lack of cooperation between pursuers, which leads to the pursuers overlapping and performing duplicate actions. The next strategy we refer to as the “Hungarian Direct Charge” policy. Here, the pursuers are assigned a target \( \kappa \) with a Hungarian algorithm, then pursue targets with (6.12). Under this policy, pursuers will only target the same evader once \( n_p > n_e \).

<table>
<thead>
<tr>
<th>Policy</th>
<th>Assignment</th>
<th>Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Charge (DC)</td>
<td>Nearest Evader</td>
<td>(6.12)</td>
</tr>
<tr>
<td>Hungarian DC</td>
<td>Hungarian Alg.</td>
<td>(6.12)</td>
</tr>
<tr>
<td>Global Area-Min</td>
<td>Nearest Evader</td>
<td>(6.9)</td>
</tr>
<tr>
<td>Local Area-Min</td>
<td>Nearest Neighbor</td>
<td>(6.10)</td>
</tr>
</tbody>
</table>

Table 6.1: Summary of pursuer policies.
Figure 6-4a summarizes the mean final capture time for seven different pursuer-evader combinations in 2D. For each combination, 100 trials with randomized initial configurations were run for each pursuer strategy. From Figure 6-4a, both the global and local area-minimization policies dominate the baseline strategies. Furthermore, note that our decentralized local policy performs just as well as the global policy in almost all scenarios. The one exception is for the combination of $n_p = 1$ pursuer chasing $n_e = 4$ evaders. Here, the local policy performs worse, but in practice, the global policy is already “decentralized” and would be the preferred implementation.

Figure 6-4b summarizes the mean final capture time for seven different pursuer:evader scenarios in 3D. For each scenario, 50 randomized trials were run for each pursuer strategy, comparing Algorithm 7 with the Direct Charge baseline policy. From these comparisons with other policies, we find that the decentralized area-minimization heuristic in Algorithm 7 is an effective pursuer policy to capture multiple evaders in a convex, bounded environment.
Figure 6.4: Mean final capture time for the pursuer policies in Table 6.1 over randomized trials. The local policy from Algorithm 7 is shown with the pink x.
6.6 Experiments

Here, we present our experimental results, which demonstrate our decentralized area-minimization policy in 2D. The experiments were conducted in the Autonomous Systems Lab (ASL) at Stanford University, with both autonomous evaders and a human-controlled evader. The environment is 4m×3m, as shown in Figure 6·6 and Figure 6·8. For the pursuers, we used four custom Ouijabots design in our lab [Wang et al., 2016]. The evaders are the Dexter Industries GoPiGo robots\(^1\). Both robotic platforms are equipped with a Raspberry Pi 2 running Linux and ROS, allowing us to implement Algorithm 7 and the evaders’ “move-to-centroid” controller completely onboard. Localization is performed with Vicon\(^2\), with position data broadcast over the ROS network. Except for this position broadcast, no other communication occurs between any two robots, and the robots have no knowledge about other robots’ policies other than determining if a neighbor is a pursuer or evader. During the experiments, the maximum velocity for all agents is capped at 0.2m/s.

To visualize when an evader is captured during experiments, the GoPiGos are equipped with status flags, pictured in Figure 6·5. While free, an evader’s flag is down. Once an evader is captured, the flag is raised.

\(^1\)http://www.dexterindustries.com/gopigo/
\(^2\)https://www.vicon.com/
Figure 6.5: (a) Picture of Ouijabot (left) and GoPiGo (right) robots used in the experiment. (b) The capture status of the GoPiGos are indicated via flags.

6.6.1 4 Autonomous Evaders

In the first experiment, all agents are autonomous and utilizing their onboard controllers. Still frames from the experiment video are shown in Figure 6.6, and the full video can be found on the MSL website\textsuperscript{3}. Initially, all pursuers start in one corner of the environment, with the evaders randomly placed. This configuration gives the

\textsuperscript{3}http://msl.stanford.edu
greatest advantage to the evaders. Despite noisy actuation and network delays, the experiment performs as expected, with all pursuers capturing all evaders. Figure 6.7 plots the minimum distance to any pursuer from the evaders over time.

**Figure 6.6:** Still frames of the experiment over time for all autonomous evaders. The environment boundary is marked in white.
The evaders in the first experiment use a decentralized move-to-centroid control policy. The policy performs well, but one may wonder if a human-controlled evader, with greater planning and predication capabilities, can do better. For this experiment, we convert one evader into a human-controlled evader. Here, we tele-operate the robot with a joystick, while all other robots remain autonomous. Unlike the other evaders, a human-controlled evader has full knowledge of the system, including how the pursuers react. Despite these advantages, the pursuer agents still successfully capture all evaders. Figure 6.8 shows still frames from our experiment video, and Figure 6.9 plots the minimum distance between any pursuer and the evaders over time. Note that in Figure 6.9, the human-controlled evader is not the last evader to be captured. Finally, Figure 6.10 plots the final capture time over the different autonomous and human-controlled trials.
Figure 6-8: Still frames of the experiment over time. The human-controlled evader is circled in red.

Figure 6-9: Minimum distance of each evader to any pursuer over time. The capture radius is denoted by the dotten line.
Figure 6.10: Final capture time of evaders over different trials of the experiment. Each trial was initialized with random positions for $n_p = 4$ pursuers and $n_e = 4$ evaders.

6.7 Summary

In this chapter, we presented our algorithm to control multiple pursuers to capture multiple evaders in a bounded, convex environment in $\mathbb{R}^N$. We first formulated a global algorithm using an “area-minimization” strategy to guarantee the capture of all evaders in finite time. Next, we extended this to a distributed version of our algorithm, which is shown to perform similarly to the global policy in simulations. Simulations in 2D and 3D demonstrate the performance and compare it with other algorithms. Experiments were conducted with the distributed algorithm driving Ouijabots to pursue GoPiGo evaders. In the experiments, we included a human-controlled evader that was unable to escape capture.
Chapter 7

Conclusions and Future Work

This dissertation examines several scenarios of multi-agent systems where cooperation and performance are not guaranteed. We classify these scenarios based on the interactions between agents and situate them within a spectrum of cooperation. Each type of problem requires a different control technique, but by using geometric methods for design insight, we propose controllers for the agents and use Lyapunov-based stability theory to analyze their behavior. Simulations and experiments on ground and aerial vehicles are used to validate the performance.

We first consider the problem of Voronoi-based coverage control, and examine the result of performance variations between agents in Chapter 3. By using weighted Voronoi cells, we can assign each agent a performance weight, such that lower-performing agents will be responsible for a smaller portion of the environment. We propose an online weightings adaptation law to adjust the agents’ weights based on relative performance variations within their neighbors. The weightings adaptation law requires only local information, and runs in parallel to the move-to-centroid positional control law. Through simulations and experiments with m3pi robots, we verify the algorithm performs as expected. One example application is for a group of agents equipped with cameras deployed over an environment to take pictures. Our algorithm adjusts for variations in the camera, such as focus, or debris on the lens, allowing the combined photos are a higher quality than without adjustments.

Next, we examine the problem of multi-agent herding, akin to shepherding, in
Chapter 4. Here, a group of “dog” herder agents must relocate a group of “sheep” herd agents to a goal region in the environment. We have direct control over the herders, but the herd agents are only reactionary. We consider this a non-cooperative system, as the sheep agents are not inclined nor opposed to cooperate with the dogs. Our key insight in designing the control laws is to enforce geometric relationships that map the combined system dynamics to a single nonholonomic vehicle. From there, we use Lyapunov’s direct method to verify the herd converges to a goal region. We demonstrate our algorithm in 2D and 3D simulations, with extensions to multiple herd members and noisy dynamics. Ground experiments were also performed with m3pi robots. Applications of our herding algorithm may include managing wildlife populations, or even human crowd control.

In Chapter 5, we investigate the cooperative pursuit of an evader by a group of quadrotors in an environment with no-fly zones. The pursuers’ goal is to track the evader as it moves through the environment while avoiding the no-fly zone obstacles. The evader is free to move within these zones, and when it enters a no-fly zone, the pursuers do not have reliable position information. We employ tools from Voronoi-based coverage control to distribute the pursuers around the no-fly zone’s boundary and minimize the capture time once the evader emerges. The quadrotors use a robust MPC controller to guarantee collision avoidance between both obstacles and other quadrotor pursuers. Experiments were conducted with three KMEL Nano+ quadrotor pursuers tracking a human-controlled m3pi evader. As hobby drones become increasingly popular, this algorithm could be applied to track an athlete, such as a mountain biker, as they move in and out of forested areas.

Finally, Chapter 6 examines a multi-pursuer, multi-evader game in a convex, bounded environment. Here, the pursuers and evaders do not cooperate, and it is the goal of the pursuers to capture all evaders. We present an algorithm that guaran-
tees the capture of all evaders in finite time by one or more pursuers. We refer to this strategy as the “area-minimization” strategy, although it generalizes to $\mathbb{R}^N$. From a Voronoi tessellation of the environment, the pursuers move towards the midpoint of the shared boundary with the evader, which reduces the safe-reachable area of the evader until it is captured. We present both a global and decentralized algorithm, and verify performance for 2D and 3D environments in simulation. We also implemented the algorithm onboard Ouijabot and GoPiGo ground vehicles, demonstrating a decentralized experiment with all robots computing their controllers in real time. Applications of these results may be useful in patrolling restricted airspace around airports and military bases.

While we have modeled several examples of how cooperation impacts multi-agent systems, there are many other scenarios not included in this dissertation. Future work may examine further scenarios along this spectrum of cooperation. Our work on Voronoi-based coverage control assumed all agents were generally cooperative and working towards the same goal. Future extensions may examine the impact of malicious agents within the system. It would first be necessary to quantify “how bad” a malicious agent would be, how it interacts with the other agents, then determine the loss in performance of the group. For example, if the malicious agents could disable the coverage-control agents, our work in Chapter 6 suggests that a malicious agent could disable the entire group.

For the herding problem discussed in Chapter 4, there are several directions for future work. One straightforward application would be to extend the work to navigating environments with obstacles. Our results only guarantee convergence to the goal for a single sheep, and future extensions may analyze the convergence in a multi-sheep case, or error analysis on the trajectory-tracking results. Beyond the relocation problems, there are other multi-agent problems related to herd behavior. Other ex-
tensions could examine the problem of consolidating sparse groups of sheep, or culling members from the group. Additionally, the herder agents may need to protect the herd from an external threat.

Chapters 5 and 6 examined two versions of a pursuit-evasion problem. In Chapter 5, the pursuers had to navigate obstacles while tracking the evader. Extensions of this work may look at limited sensing ranges for the pursuers, such as maintaining line of sight or a probabilistic model for the evader. Another direction is to track multiple evaders, which may borrow tools from the algorithm presented in Chapter 6. While we were able to guarantee capture of multiple evaders in a convex, bounded environment, future extensions may study both non-convex and unbounded environments.

Overall, our work provides insight into designing control strategies for multi-agent systems when cooperation is not guaranteed. Depending on the interactions between agents, we can use geometric methods to derive individual controllers for the agents. We provide a series of scenarios that encompass the spectrum of cooperation in multi-agent systems, from mild performance variations to adversarial pursuer-evader problems. For each of our proposed controllers, we use Lyapunov-based stability theory to analyze their behavior, and verify performance through simulations and experiments. We hope the work presented here will drive further research and help in integrating multi-agent systems into real-world applications.
Appendix A

Index of Notation

Here we include the notation used in each chapter.

A.1 Voronoi-Based Coverage Control

\[ Q \ldots \text{Bounded, convex environment} \]
\[ q \ldots \text{Points in } Q \]
\[ \phi(q) \ldots \text{Information density function} \]
\[ p_i \ldots \text{Location of agent } i \]
\[ V_i \ldots \text{Voronoi Cell for agent } i \]
\[ W_i \ldots \text{Weighted Voronoi Cell} \]
\[ M_{W_i} \ldots \text{Mass of } W_i \]
\[ C_{W_i} \ldots \text{Centroid of } W_i \]
\[ N_i \ldots \text{Set of Voronoi neighbors} \]
\[ H_W \ldots \text{Sensing cost function} \]
\[ \gamma_i(\cdot) \ldots \text{Sensor data quality function} \]
\[ h_i \ldots \text{Sensor health} \]
\[ w_i \ldots \text{Performance weight (used in } W_i) \]
\[ u_i \ldots \text{Control input for } p_i \]
\[ \Delta_i \ldots \text{Actuation errors} \]
\[ K_i \ldots \text{Matrix for true actuation performance, } \dot{p}_i = K_i u_i \]
\[ \hat{K}_i \ldots \text{Agent’s estimate of } K_i \]
\[ b_{ij} \ldots \text{Voronoi boundary between } i \text{ and } j \]
\[ d_{ij} \ldots \text{Length of } b_{ij} \]
\[ L \ldots \text{Weighted Laplacian of neighbor graph} \]
A.2 Herding

\[ d_j \ldots \text{Positions of “dog” herders} \]
\[ s_i \ldots \text{Positions of “sheep” herd members} \]
\[ u_j \ldots \text{Control input for } d_j \]
\[ B_\ell(g) \ldots \text{Goal region} \]
\[ A_R^B \ldots \text{Rotation matrix from local } B \text{ frame to global } A \text{ frame} \]
\[ b_x, b_y, b_z \ldots \text{Basis vectors in global coordinates} \]
\[ v \ldots \text{Forward velocity of vehicle} \]
\[ \omega \ldots \text{Angular velocity of vehicle} \]
\[ \phi \ldots \text{Heading of 2D vehicle} \]
\[ \phi, \theta, \psi \ldots \text{ZYX Euler angles for 3D vehicle} \]
\[ p \ldots \text{Point offset from } s \]
\[ \ell \ldots \text{Length of point offset in } b_x \text{ direction} \]
\[ \Delta \ldots \text{Desired separation of } d_j \text{ in 2D} \]
\[ \Delta_{\alpha}, \Delta_{\beta} \ldots \text{Desired separation of } d_j \text{ in 3D} \]
\[ r \ldots \text{Control radius for } d_j \]

A.3 Cooperative Pursuit

\[ p_i \ldots \text{Position of pursuer } i \]
\[ e \ldots \text{Position of evader} \]
\[ h_{p_i}, h_e \ldots \text{Altitude of agent} \]
\[ \mathcal{Z}_j \ldots \text{No-fly zone obstacle} \]
\[ \partial\mathcal{Z}_j \ldots \text{Boundary of } \mathcal{Z}_j \]
\[ Q^i \ldots \text{Maximum-volume safety ellipsoid about } p_i \]
\[ \mathcal{E}_\mathcal{Z}_j \ldots \text{Minimum-volume ellipsoid around } \mathcal{Z}_j \]
\[ \epsilon^i \ldots \text{Dummy evader target for } p_i \]
\[ \mathcal{R}_j \ldots \text{Evader’s reachable set inside no-fly zone} \]
\[ \rho(q,t) \ldots \text{Indicator function from } \mathcal{R}_j \]
\[ \bar{p}_i \ldots \text{Unrestricted pursuer position} \]
\[ \mathcal{V}_i \ldots \text{Voronoi Cell for agent } i \text{ inside } \mathcal{R}_j \]
\[ M_{\mathcal{V}_i} \ldots \text{Mass of } \mathcal{V}_i \]
\[ C_{\mathcal{V}_i} \ldots \text{Centroid of } \mathcal{V}_i \]
\[ d^*_i \ldots \text{Target position along } \partial\mathcal{Z}_j \]
A.4 Guaranteed Capture of Multiple Evaders

\( x_p^j \) ... Position of pursuer \( j \)
\( x_e^i \) ... Position of evader \( i \)
\( r_c \) ... Capture radius
\( V_i \) ... Voronoi cell of agent \( i \)
\( A_{e_i} \) ... Safe-reachable area of an evader
\( \kappa \) ... Index of a targeted evader
\( P_\kappa \) ... Set of pursuers that share \( \kappa \) target
\( b_{\kappa j} \) ... Shared boundary between pursuer and targeted evader
\( C_{b_{\kappa j}} \) ... Centroid of shared boundary
References


Curriculum Vitae

Alyssa Pierson  
people.bu.edu/pierson  
pierson@bu.edu

Education

2017  |  PhD, Mechanical Engineering, Boston University
      |  Visiting Student, Stanford University
2016  |  MS, Mechanical Engineering, Boston University
2010  |  BS, Engineering, Harvey Mudd College
      |  Study Abroad, University of Western Australia

Research Experience

2012-2016  |  Research Assistant, Multi-Robot Systems Lab  
      |  Advisor: Prof. Mac Schwager  
      |  Designed online control algorithms for nonlinear, distributed, and 
      |  heterogeneous multi-robot systems. Hardware implementations with 
      |  quadrotors and ground vehicles.
2008-2011  |  Flight Analysis of a Turning Pigeon, Harvey Mudd College  
      |  Rigorous kinematic analysis of a pigeon’s body movements through 
      |  90-degree turn. Joint project with the Concord Field Station and 
      |  Harvard University.

Honors & Awards

2016  |  Best Conference Paper Finalist, ICRA
2012-2014 |  Clare Boothe Luce Fellowship
2010  |  Graduated with Distinction, Harvey Mudd College
2007-2010 |  Dean’s List, Harvey Mudd College
2006  |  Valedictorian, Steamboat Springs High School
Work Experience

2010 - 2012 | Cobham Graduate Development Program, Cobham, plc
Rotational program to gain exposure across Cobham business units. Worked as a design engineer and program manager in Orchard Park, NY and Nashua, NH.

2008 | Software QA Engineer Intern, Laserfiche
Wrote test protocols and training seminars for QA engineers and the help manuals for general users. Tested the user interface of the Laserfiche product suite.

Project Experience

2009-2010 | University of Iceland Global Clinic, Harvey Mudd College
Designed small-scale Organic Rankine Cycle system to generate electricity from a low-temperature heat source for use as “backyard waste heat reclamation”

2008 | Nike Clinic, Harvey Mudd College
Designed shoes that are manufactured as independent parts and assembled at the retail level. Focus on consumer customization, small-scale manufacturing processes.

Journal Publications


Conference Publications


