COOPERATIVE HYBRID CONTROL OF ROBOTIC SENSORS FOR PERIMETER DETECTION AND TRACKING

By

JUSTIN DELANEY CLARK

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

2002

Submitted to the Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the degree of
MASTER OF SCIENCE
May, 2005

COOPERATIVE HYBRID CONTROL OF ROBOTIC SENSORS FOR PERIMETER DETECTION AND TRACKING

Thesis Approved:

Dr. Rafael Fierro - Thesis Advisor

Dr. Martin Hagan

Dr. Eduardo Misawa

Dr. A. Gordon Emslie - Dean of the Graduate College

Acknowledgments

My genuine thanks to my thesis advisor, Dr. Rafael Fierro, for all of his encouragement, support, advice, discussions, and for allowing me to be a part of the MARHES laboratory.

I would like to thank Dr. Martin Hagan and Dr. Eduardo Misawa for serving on my committee and for sparking my interest in control systems. I am honored to have had the opportunity to have taken classes from you.

I am indebted to all of the members of the MARHES Laboratory including: Daniel Cruz for his extensive help with the Gazebo simulations, the mobile platform, and the experiments; Kenny Walling for designing the CAN control unit and CAN sensor nodes and for the many discussions/arguments we had concerning future plans; Chris Flesher and Omar Orqueda for their help with vision; Carlo Branca for his interest in my work and James McClintock for all of his help relating to this work.

This work is dedicated to my parents for their support and patience over the last two years. Also, to my brother for living and putting up with me this last year. Finally, to Angela, for your support and all of your help editing and improving the flow of this thesis.

Last, but definitely not least, my gratitude to the sponsors for supporting this work, specifically, NASA and Dr. Andrew Arena, Deputy Director of the NASA Oklahoma Space Grant program, for NASA Oklahoma Space Grant Fellowships. Also, the National Science Foundation under Embedded and Hybrid Systems grant #0311460 and CAREER Award #0348637 and the U.S. Army Research Office under grant DAAD19-03-1-0142 (through the University of Oklahoma).

Contents

List of Figures

Chapter 1

Introduction

The goal of this work is to provide experimental results of a real-world multi-vehicle coordination application, ***perimeter detection and tracking***, not to prove optimality, convergence, stability, etc. *Perimeter detection and tracking* means the following: A robotic swarm cooperatively searching for, locating, and tracking a dynamic perimeter while avoiding collisions. The assumption is that a chemical spill, *i.e.*, oil, is the dynamic perimeter and that the general, but not exact, location of the dynamic perimeter is already known by some means, *i.e.*, an unmanned aerial vehicle (UAV). Other groups have used the terms *perimeter* and *boundary* interchangeably, but in this thesis, there is a distinct difference. The perimeter is the *chemical substance* being tracked, while the *boundary* is the limit of the exploration area.

In this thesis, the cooperative system is modeled as a hybrid system (a dynamical system composed of continuous dynamics and decision logic). A library of simple controllers, *i.e.*, bang-bang, proportional, etc., have been designed; they are combined to form a hybrid system where group behavior emerges allowing the system to accomplish the perimeter detection and tracking application.

More formally, a decentralized coordination algorithm is presented that allows a robotic swarm to locate and track a dynamic perimeter. A cooperative communication scheme is used by the team to rapidly detect a perimeter. Collision-free cycling behavior emerges by composing simple reactive control laws. The decentralized framework could potentially allow the algorithm to scale to many robots. Extensive

simulation results in Matlab and Gazebo (a more realistic simulator) and experiments verify the validity and scalability of the proposed cooperative control scheme [1].

1.1 Motivation

Many applications of multi-robot cooperation have been studied including area coverage, search and rescue, manipulation, exploration and mapping, and perimeter detection [2, 3, 4, 5, 6, 7, 8]. The reader is referred to [9] for a survey on cooperative mobile robotics.

Perimeter detection could have a wide range of uses in several areas, including: (1) Military, *i.e.*, locating minefields or surrounding a target, (2) Nuclear/Chemical industries, *i.e.*, tracking radiation/chemical spills, (3) Oceans, *i.e.*, tracking oil spills, and (4) Space, *i.e.*, planetary exploration. In many cases, humans are used to perform these dull and/or dangerous tasks; however, robotic swarms replacing humans would be more beneficial.

A perimeter is an area enclosing some type of substance. Consider two types of perimeters: (1) static and (2) dynamic. A static perimeter does not change over time, *i.e.*, possibly a minefield. Dynamic perimeters are time-varying and expand/contract over time, *i.e.*, a radiation leak. See Fig. 1.1 for examples of perimeters, an oil spill[1] and a concentration function representing a perimeter.

In perimeter detection tasks, a robotic swarm locates and surrounds a substance, while dynamically reconfiguring as additional robots locate the perimeter. All robots must be equipped with sensors capable of detecting whatever substance they are trying to track. Substances could be airborne, ground-based, or underwater.

The ability of a robot to track a perimeter depends on the robot's kinematic and dynamic constraints, along with the characteristics of the perimeter. A robot's limitations and several different types of perimeters that may be encountered while

[1] "The IXTOC I exploratory well blew out on June 3, 1979 in the Bay of Campeche off Ciudad del Carmen, Mexico. By the time the well was brought under control in 1980, an estimated 140 million gallons of oil had spilled into the bay. The IXTOC I is currently #2 on the all-time list of largest oil spills of all-time, eclipsed only by the deliberate release of oil, from many different sources, during the 1991 Gulf War." Courtesy of the NOAA Office of Response and Restoration.

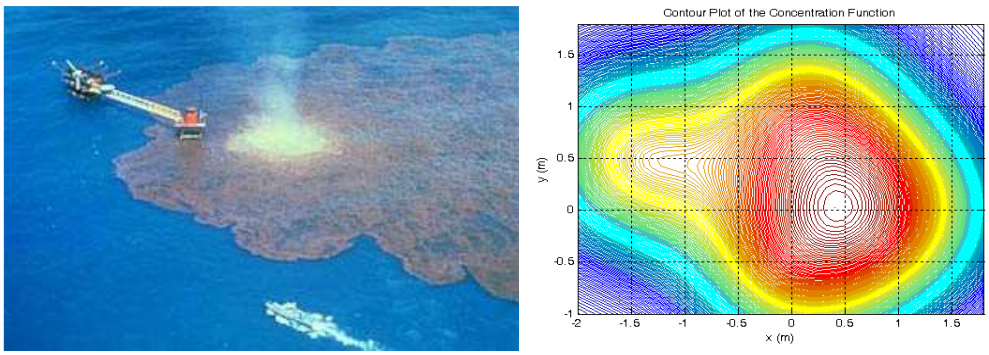

Figure 1.1: Example perimeters: (a) Oil spill and (b) Concentration function.

performing perimeter detection and tracking are described below.

The physical constraints of a robot must be taken into account, as several things may deter a robot from being able to track a perimeter including the robot's maximum speed and limited turning radius. If a dynamic perimeter is expanding faster than the robot's maximum speed, the perimeter cannot be tracked. A perimeter can only be accurately tracked if there are no abrupt turns required. Specifically, if a required turn is greater than the robot's radius of curvature, some error will occur or the robot will completely miss a section of the perimeter and may not be able to recover.

The perimeters are assumed to be contiguous, convex, and trackable. See Fig. 1.2 for some examples of different types of perimeters. In Fig. 1.2(a), the perimeters are

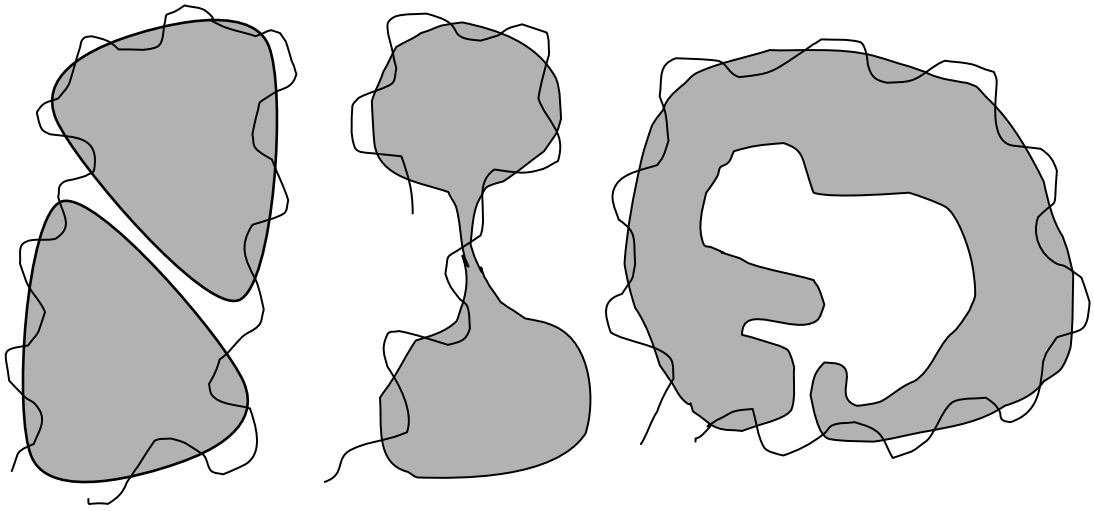

Figure 1.2: Perimeters: (a) Separate, (b) Pinched, and (c) Concave.

not contiguous. The robots may recognize these two perimeters as one large perimeter

depending on the space in between the individual perimeters. In Fig. 1.2(b), the perimeter is pinched in the middle, potentially causing the robots to track the wrong portions of the perimeter. In Fig. 1.2(c), because the perimeter is concave, the robots may be unable to track the interior of this perimeter. The aforementioned examples are primary illustrations of perimeters that may not be trackable.

1.2 Contributions

The contributions of this thesis are two-fold: The design and implementation of a hybrid system capable of executing perimeter detection and tracking.

A decentralized, cooperative hybrid system is developed utilizing biologically-inspired emergent behavior. Emergent behavior is defined as, "when a number of simple entities operate in an environment, forming more complex behaviors as a collective" [2]. The algorithm has been verified by simulations and through experiments.

A library of simple controllers compose the hybrid system, and each controller consists of finite state machines. Controllers are developed to search for, move to, and track a dynamic perimeter. The proportional tracking controller developed is unique; it utilizes the difference between color blobs seen by an inexpensive camera to control the angular velocity, thus, allowing the robot to smoothly track the perimeter. A communication scheme is implemented permitting the robots to coordinate to reach the perimeter once one robot initially detects the perimeter.

This thesis can be used as a basis for a formal analysis of the perimeter detection and tracking hybrid system or for a system capable of dynamic perimeter estimation. Furthermore, it can be used as a reference if the researcher is interested in extending the results to an outdoor environment or wishes to use the results from the tracking controller for another application.

[2] http://encyclopedia.thefreedictionary.com/Emergent behavior

1.3 Thesis Outline

Hereafter, the organization of the thesis is as follows: A literature review of some relevant work is discussed in Chapter 2. Additionally, summaries of other applications are given along with applications closely related to perimeter detection and tracking.

A brief overview of hybrid systems theory is presented in Chapter 3. A detailed description of the hybrid system is described, which includes the model used to represent the platform and descriptions of each controller agent.

Chapter 4 contains simulation results for static and dynamic perimeters. Specifically, Matlab was used as a proof of concept to verify the hybrid system followed by the use of Gazebo to create a more realistic environment for simulation purposes. The advantages and disadvantages of each simulator are discussed.

Provided in Chapter 5 are a brief description of the experimental testbed, a list of hardware difficulties, and a discussion of detailed experimental results for static and dynamic perimeter tracking with three robots.

Chapter 6 presents the concluding remarks of this thesis as well as ideas for future work for improving the algorithm. Specifically, estimating the evolution of a dynamic perimeter and a formal analysis of the system behavior.

Chapter 2

Literature Review

2.1 Introduction

Multi-robot cooperation includes a variety of applications such as area coverage, search and rescue, manipulation, exploration and mapping, and perimeter detection and tracking.

This literature review is inexhaustive and subjective. Particularly, some of the algorithms developed could fall into multiple applications, each of which were grouped into applications deemed appropriate. Other applications are discussed, followed by applications more closely related to perimeter detection and tracking.

2.2 Other Applications

To accomplish perimeter detection and tracking, the results from numerous other applications might be extended and combined. The following is an inexhaustive list of previously reviewed approaches: area coverage, cyclic pursuit, formation control, and gradient tracking.

2.2.1 Area Coverage

Area coverage occurs when a mobile sensor network is trying to cover an area efficiently. Detailed below are some approaches to solving this application.

In [7], the algorithm uses virtual potential fields and graph theory (Delaunay triangulation and Voronoi diagrams) to allow a sensor network to efficiently cover an area. Delaunay triangulation is an optimal partioning of the space around a set of irregular points (robots) into non-overlapping triangles and their edges[1]. On the other hand, a Voronoi diagram is a special kind of decomposition of a metric space determined by distances to a specified discrete set of objects in the space[2]. The Voronoi diagram and the Delaunay triangulation are duels to each other. These methods allow each robot's relationship (distances) with its neighbors to be uniquely defined, which allows the virtual potentials to be applied, and the network to cover the area. See Fig. 2.1 for a simulation of 50 robots covering an area.

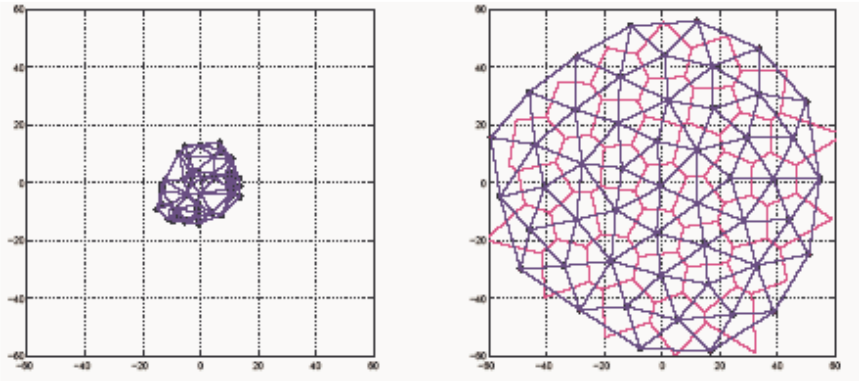

Figure 2.1: Self-deployment of 50 mobile robots in an open space (Delaunay triangulation in blue, Voronoi diagram in red): (a) Initial deployment and (b) Final deployment [7].

In [10], robots cover a plane in a specified pattern such as in Fig. 2.2. Each robot communicates only with its nearest neighbors and when a robot fails, a new group of nearest neighbors form. Optimization is used to minimize the error between neighboring robots.

2.2.2 Cyclic Pursuit

Cyclic pursuit can be represented as individual agents following the next agent resulting in a cyclic group behavior. Cyclic pursuit is studied for holonomic robots in [11], in which each vehicle follows the vehicle in front of it (1 follows 2, 2 follows 3,

[1]http://cwx.prenhall.com/bookbind/pubbooks/clarke/chapter3/custom2/deluxe-content.html
[2]http://en.wikipedia.org/wiki/Voronoi_diagram

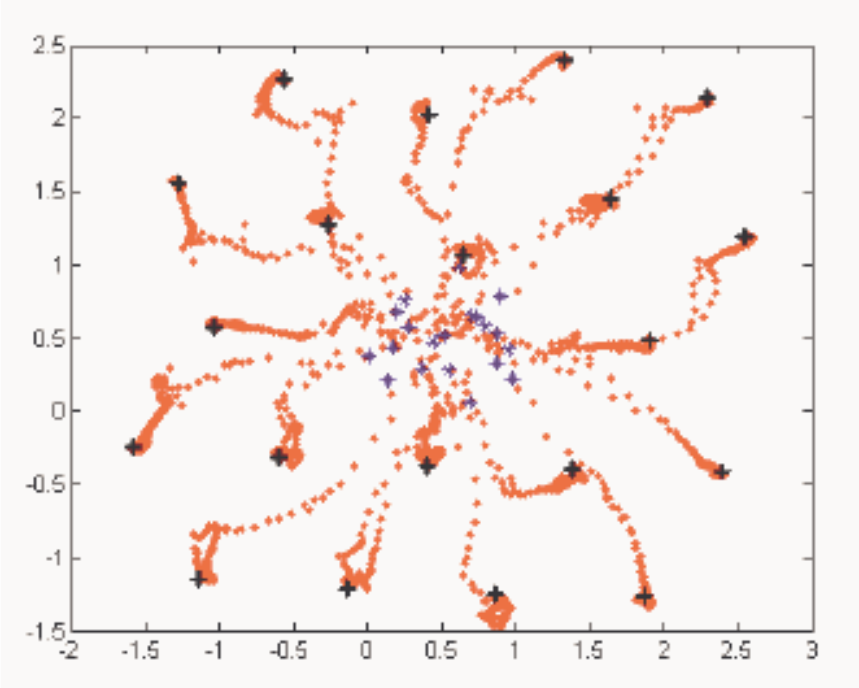

Figure 2.2: 20 robots spreading out to cover a plane [10].

etc.), hence, forming regular formations (polygons). The vehicles move with constant speed and use a proportional controller to handle orientation. Perfect sensing is assumed, collisions are ignored, and the result is extended to the unicycle model. The algorithm is distributed, scalable, and requires local communication (each agent only needs information from one other agent). See Fig. 2.3 for an example of a cyclic pursuit simulation.

2.2.3 Formation Control

Formation control is a multi-agent application where the objective is for the robotic network to move into a desired formation (particular pattern, *i.e.*, triangular) and to accomplish the desired task. Several examples of formation control are described below.

In [12], a control law is described that guarantees obstacle avoidance by using gyroscopic forces. Gyroscopic forces denote forces that do *not* do any work and create coupling between different degrees of freedom [13]. Gyroscopic forces act like steering forces and do not affect the total energy of the system. A braking force is also

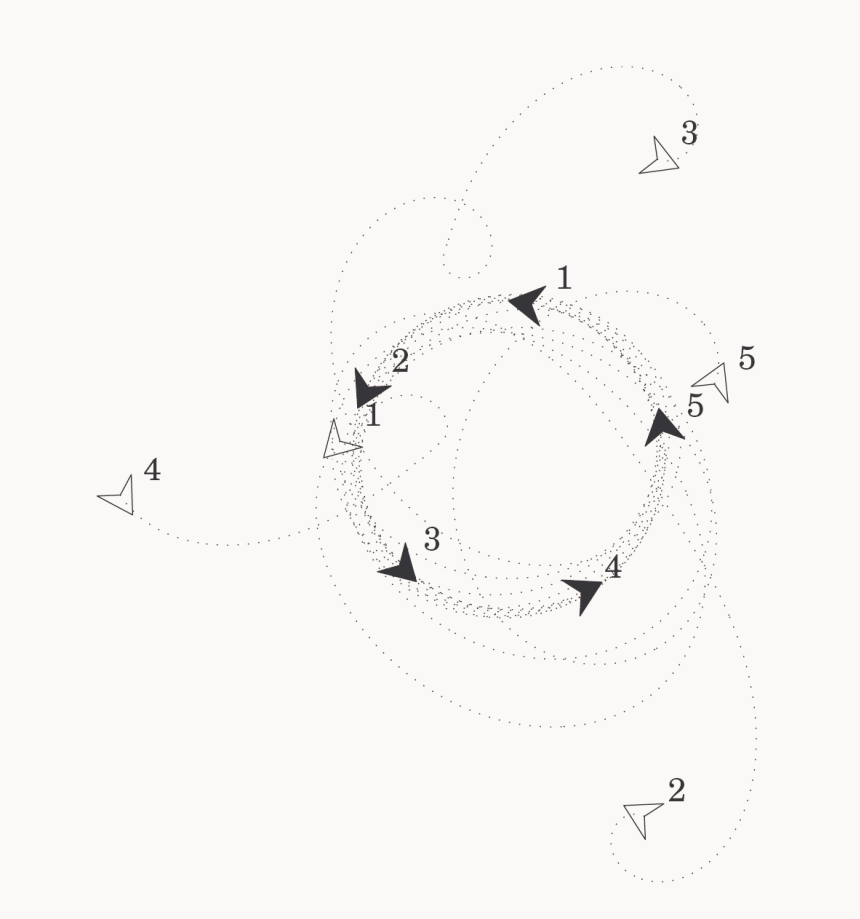

Figure 2.3: Five vehicles in cyclic pursuit [11].

used that allows the vehicles to slow down in order to turn, hence, avoiding a collision. The braking forces also do not affect the total energy of the system. Each vehicle can sense neighboring obstacles or vehicles only in its detection shell. If multiple obstacles are inside the detection shell, the vehicle only reacts to the closest obstacle. A benefit of this method is obstacles behind the vehicle are ignored, even if they are inside the detection shell. A simulation of a group of 25 vehicles accomplishes the split/rejoin maneuver in which the vehicles' target is a virtual vehicle moving in a straight line in Fig. 2.4.

In [14], the development of a decentralized framework allows a nonholonomic sensor network to navigate using only an omnidirectional camera sensor. See Fig. 2.5 for an example of formation switching.

In [15], a hybrid control system is designed and implemented for formation con-

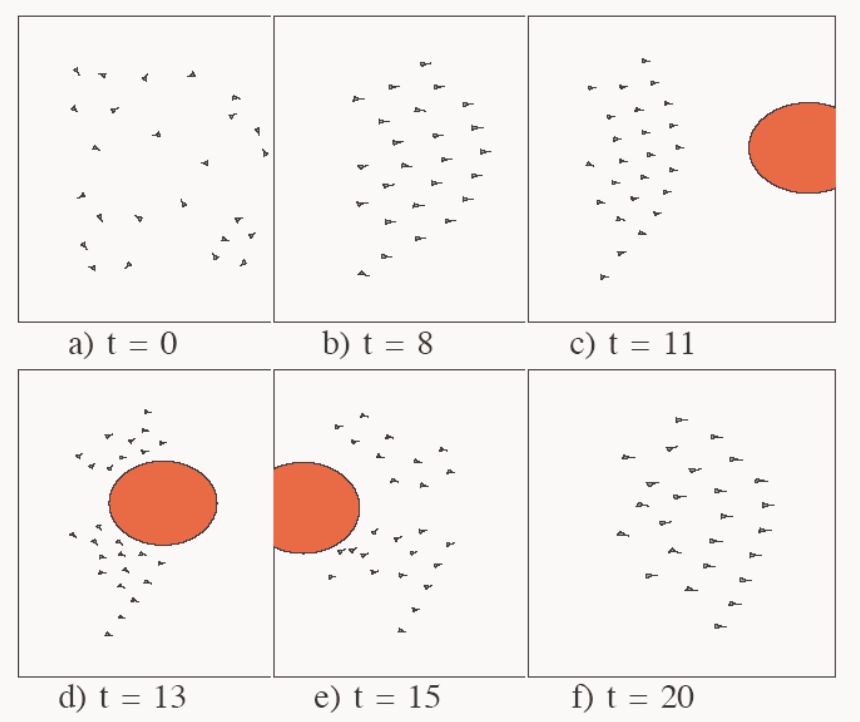

Figure 2.4: Split/rejoin maneuver for a group of 25 vehicles [12].

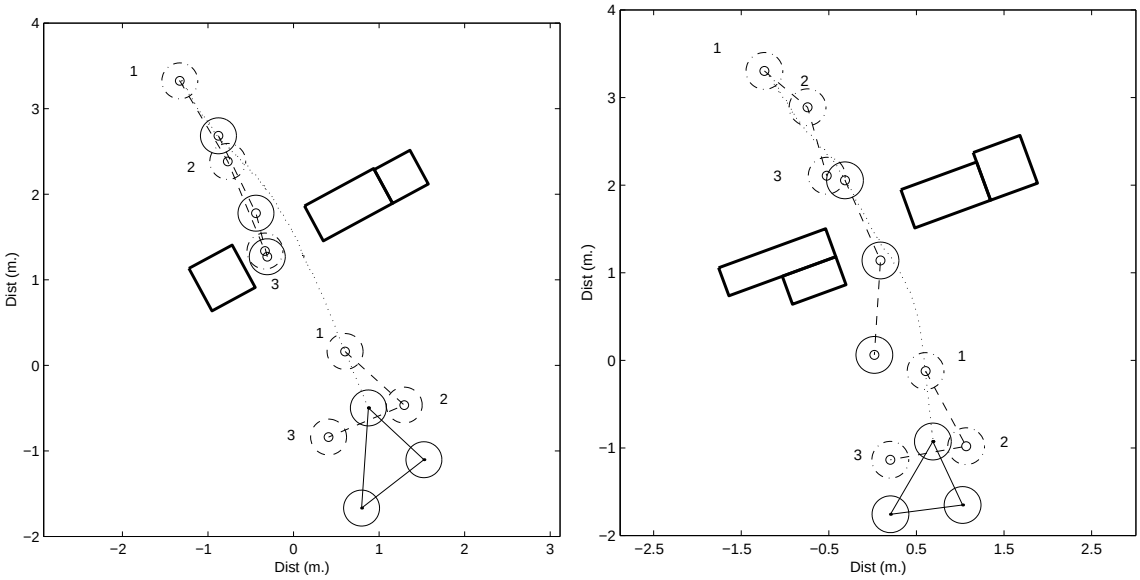

Figure 2.5: Experimental formation switching to avoid obstacles with three robots [14].

trol that does not require communication. Nonlinear controllers, along with optimal estimation techniques (extended Kalman filter) are used instead of communication. See Fig. 2.6 for a simulation of three robots maintaining a formation while avoiding obstacles.

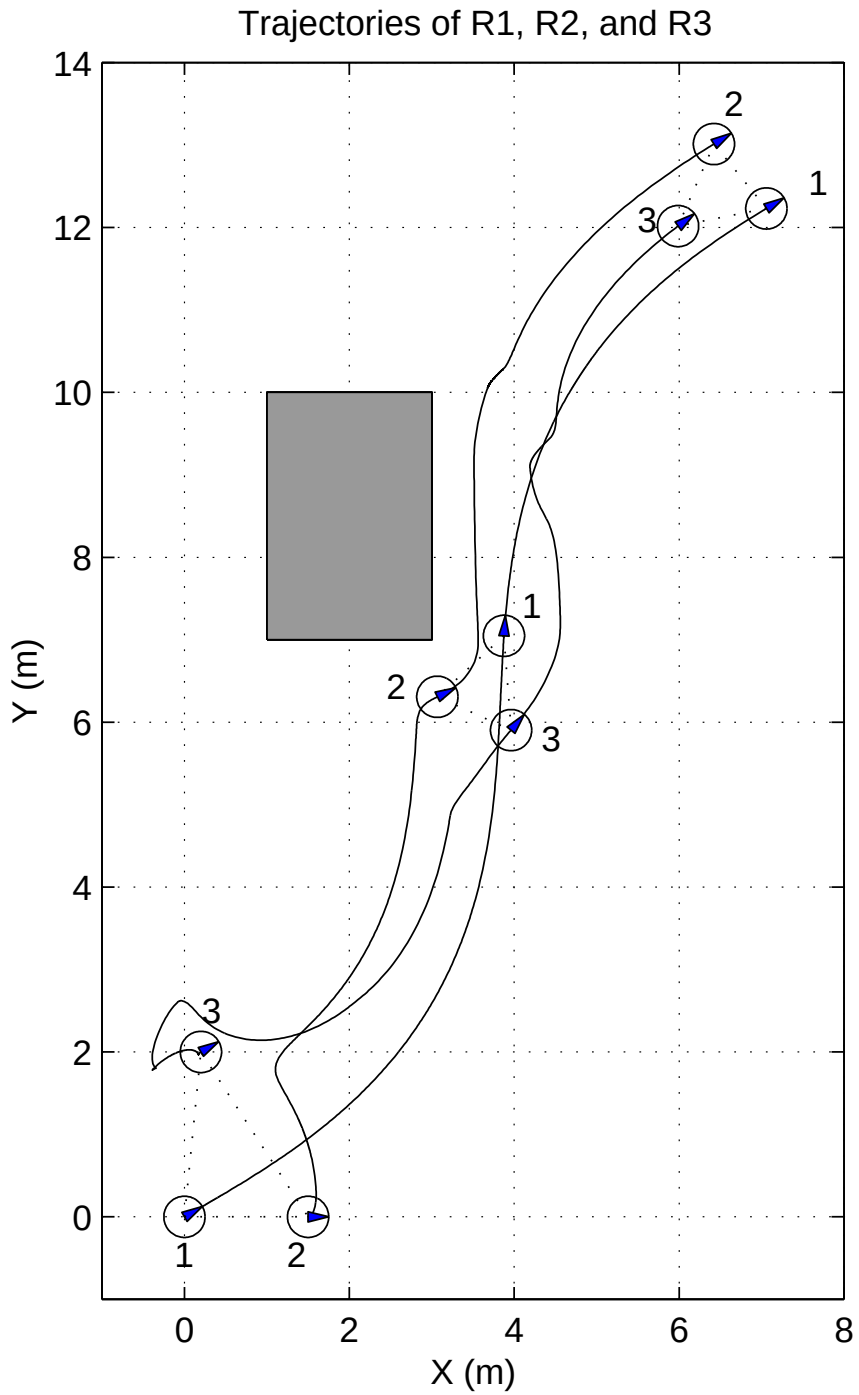

Figure 2.6: Three robot formation control with obstacle avoidance [15].

A sensor network is able to avoid obstacles while maintaining a formation in [16]. A simulation is presented in Fig. 2.7 where three robots are able to navigate through a difficult set of obstacles while maintaining a triangular formation. One drawback is a leader must be selected. The dynamic unicycle model, which includes torque, is used. Graph theory and input-to-state stability are used to maintain formations, while

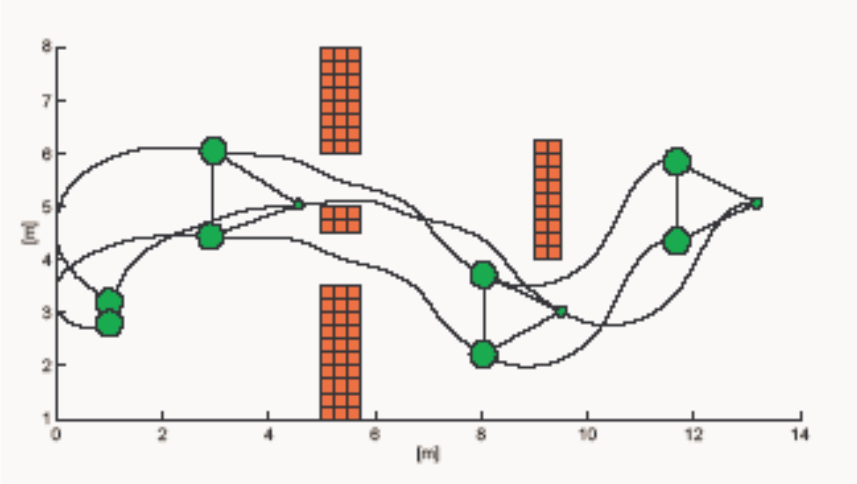

Figure 2.7: Three robots in formation while avoiding obstacles [16].

a revised Dynamic Window Approach (DWA) is used for obstacle avoidance. The DWA is a combination of model predictive control and a control Lyapunov function (proper, positive definite function where, if it is possible to make the derivative less than zero at every point, the system can be stabilized [17]), which the authors proved is convergent in a previous work.

2.2.4 Gradient Tracking

Gradient tracking can be defined as a robotic swarm moving towards a target or source by using some form of the gradient method. Gradient descent/climbing methods are useful for surrounding a target or tracking an environmental plume (pollutants released from a point source[3]). Current approaches to solving this problem are summarized below.

In [18], a simulation shows a sensor network climbing/descending an environmental gradient. With limited sensing (each vehicle is only able to measure the gradient along its path) and local communication (nearest neighbors), a vehicle network is able to find the global minimum of a system, while individual vehicles are not. Three vehicles form a triangle while moving towards the minimum in Fig. 2.8.

Gradient descent algorithms are developed in [3]. A simulation is presented in

[3]http://www.dictionary.com

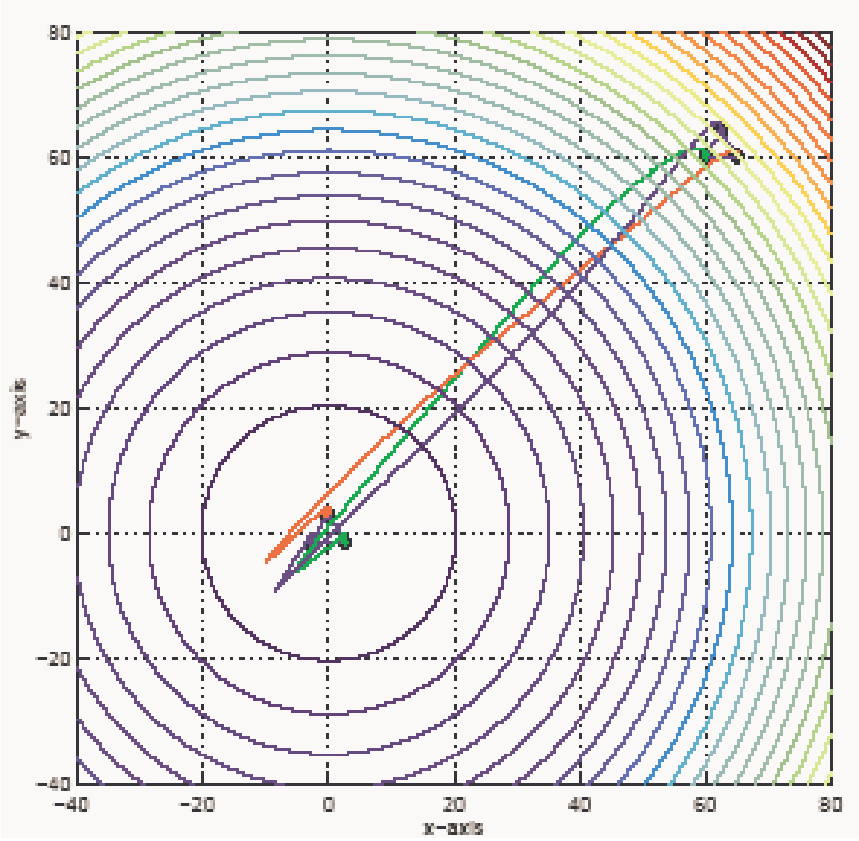

Figure 2.8: Three robots converging to a source [18].

Fig. 2.9 that allows a group of 32 vehicles to converge and surround a target while avoiding collisions.

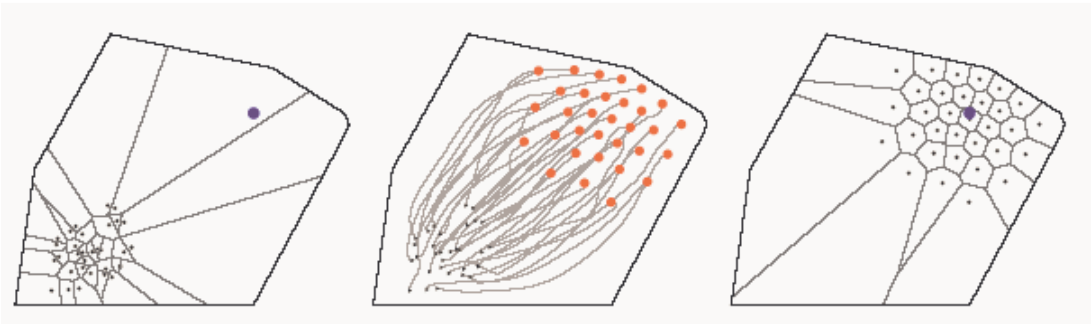

Figure 2.9: Coverage control with 32 vehicles: (a) Initial deployment, (b) Trajectories, and (c) Final deployment [3].

In [8], a method using artificial potentials and virtual bodies is shown. The vehicles are modeled as point masses with fully actuated dynamics. It is possible to extend the results to underactuated vehicles. Inter-vehicle potentials, based on natural swarms, are used for low-level control, and perfect sensing is assumed. This approach is

centralized, as the virtual bodies require intensive processing. It is assumed that the field the robots are tracking is unknown, but can be measured along the path the vehicle takes. Simulations are shown for three to eight vehicles in which the networks form regular polygons upon uniformly surrounding the target in Fig. 2.10.

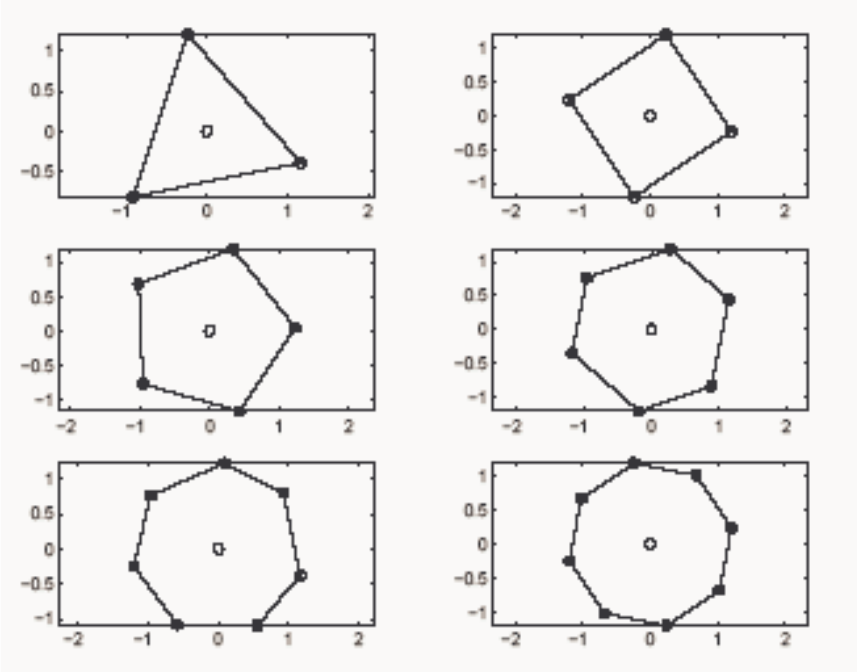

Figure 2.10: Formation control with three to eight vehicles [8].

A Swarm Intelligence summary is presented and an implementation applying odor tracking to mobile robots is shown in [19]. A spiral search pattern is used for initially searching an area that does not optimally cover an area (as compared to a coordinated search, which is considered the best in [20]), but is nonetheless, effective. When an odor is detected, the robots move upwind for a constant distance. If the odor is still detected, they will repeat this maneuver until the source is lost, if ever. If lost, they will start the spiral search again. Simple communication is used, *i.e.*, if one robot locates the source, it will chirp, and all robots downwind or that have not located an odor will move towards it. The testing area is shown in Fig. 2.11.

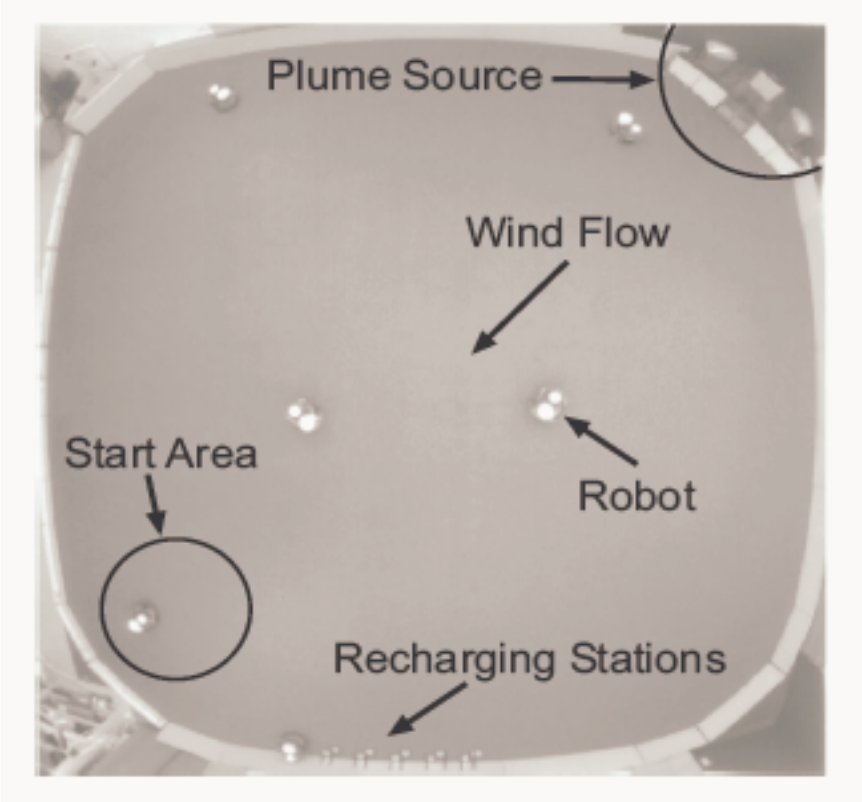

Figure 2.11: Plume tracking testbed [19].

2.3 Related Applications to Perimeter Detection and Tracking

Some of the previous and ongoing work related to perimeter detection and tracking is briefly reviewed below.

In [2], an interesting approach to swarm behavior rooted in biology is presented. Social potential fields are implemented using IR, chirps, and light sensing. Emergent behaviors, *i.e.*, searching and perimeter formation, occur as the robots respond to light and sound changes. Traditional sensors such as encoders, GPS, sonar, etc. were abandoned in favor of touch, light, and IR sensors, along with a speaker and microphones. Some of the robots use RF transmissions to receive commands from a human operator. These robots then use IR sensors to transmit this information to the others. A group of GrowBots were able to autonomously locate and surround a water spill in Fig. 2.12.

Figure 2.12: Robots deployed in a cluttered DOE facility form a perimeter around a water spill [2].

Perimeter surveillance is experimentally accomplished outdoors over a large area using a group of vehicles that investigates alarms from intrusion detection sensors in [5]. The vehicles spread uniformly around the perimeter and communicate using differential GPS and RF radios. When an alarm is tripped, the closest vehicle will investigate, while the others adapt to maintain the perimeter. Note that the sensors are broadcasting a signal, whereas if the perimeter were a chemical, this broadcast would not occur. In regards to perimeter detection as described in this thesis, the robot must be equipped with a sensor that can detect the perimeter, as an intrusion sensor will not be in place. A formal stability analysis is shown for a large-scale system. The sensor network was able to surround a facility and uniformly space around it over large distances in Fig. 2.13.

In [21], modified snake (energy-minimizing curves) algorithms, adapted from image processing, are constructed for a group of vehicles. A *snake* (vehicles) could be thought of as a "rubber band" that inflates or contracts until it fits around the perimeter. A snake is controlled by internal forces and external constraint forces. The agents are assumed to be equipped with sensors capable of detecting the chem-

Figure 2.13: Four vehicles surrounding the backside of a facility [5].

ical they are tracking. Represented by a concentration function in Fig. 2.14 is a simulation of the snake surrounding the perimeter. Two different forms of updating are applied: (1) frequent communication where each robot is given the positions of its neighbors, and (2) infrequent communication where each robot must know the positions of all robots. Shown by simulation, the algorithm can overcome large amounts of agents failing, while sacrificing accuracy. Perfect sensing, communication, and a static perimeter is assumed. This method may be suitable for tracking static perimeters, but may have difficulty tracking dynamic perimeters because the gradient of the concentration may change rapidly.

The research that most closely relates to the tracking aspect of the work described in this thesis is in [22]. An algorithm is presented in which vehicles equipped with binary sensors turn left and right along circular paths to track a static perimeter (see Fig. 2.15). The linear velocity is adjusted to distribute the group around the perimeter. The drawbacks to this method are the vehicles start on the perimeter and the large amount of error while tracking the perimeter. Furthermore, the vehicles repeated entry of the perimeter can create a problem if the perimeter is for instance,

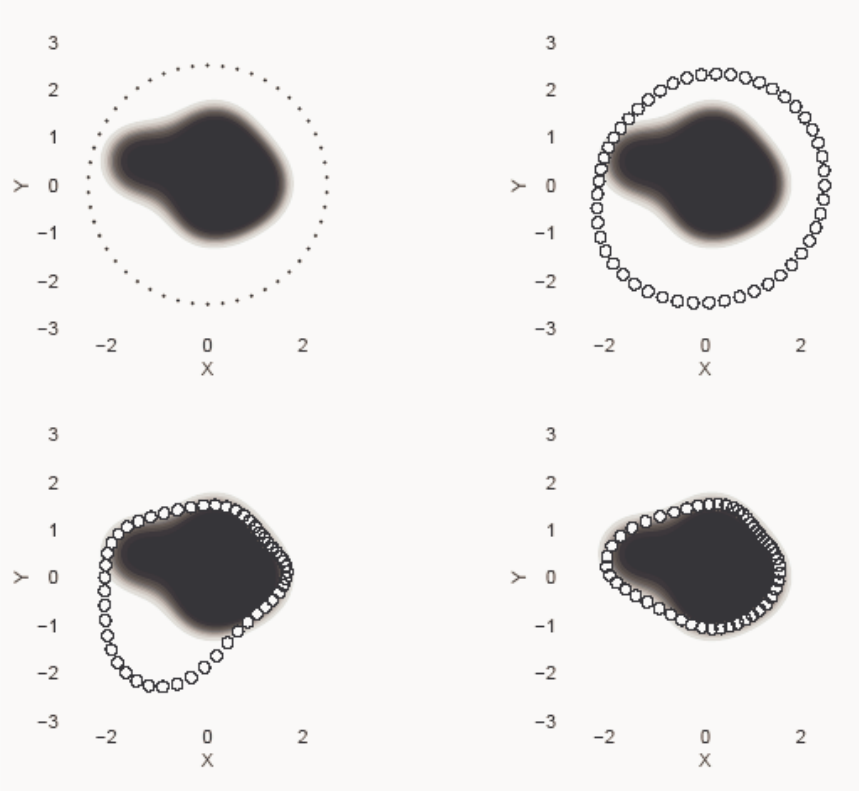

Figure 2.14: Snake contracting to surround a static perimeter: (a) The initial placement of agents is on a circle of radius 2.5, (b) Agents begin to move towards the perimeter, (c) Agents partially surrounding the perimeter, and (d) Final configuration where the perimeter is completely surrounded [21].

oil that causes wheel slippage or a minefield destroying the vehicle. Lastly, no results are shown for dynamic perimeters.

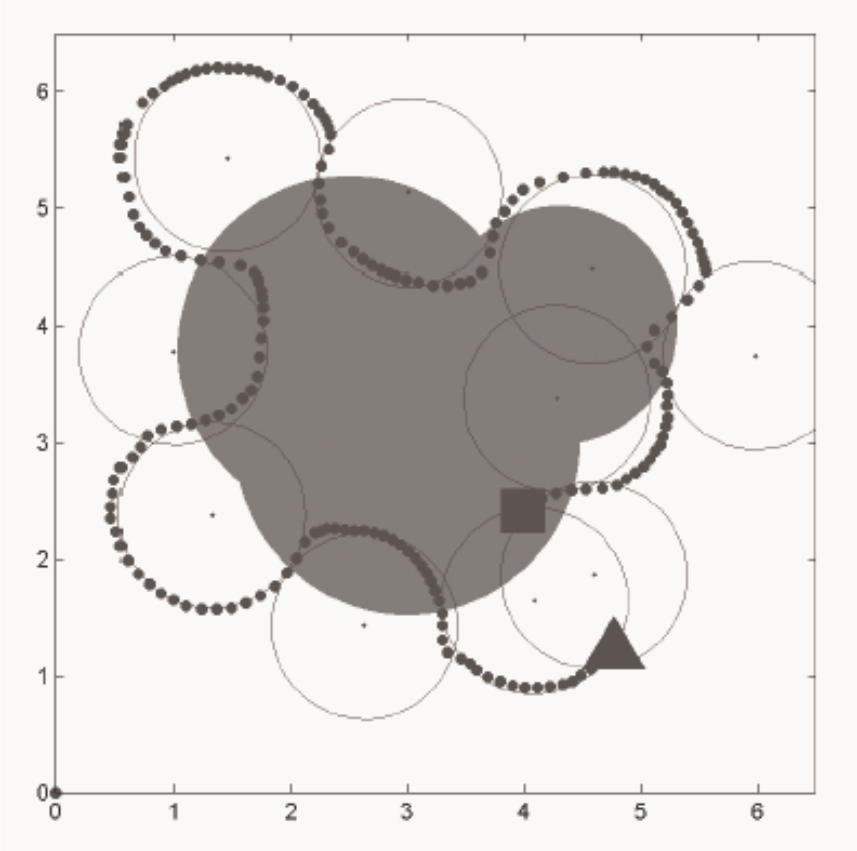

Figure 2.15: Single vehicle tracking a static perimeter (note the tracking error) [22].

Chapter 3

Hybrid System for Perimeter Detection and Tracking

3.1 Introduction

This chapter presents a brief summary of hybrid systems as an introduction to this relatively new area of research. In addition, a hybrid system designed specifically for perimeter detection and tracking is detailed.

3.2 Overview of Hybrid Systems

The following summaries of hybrid systems theory and hybrid automata were guided by the work of [23].

3.2.1 Introduction to Hybrid Systems

A hybrid system is a dynamical system that consists of continuous dynamics and decision logic (discrete states). The system continuously evolves unless there is a change in the decision logic. Therefore, a time elapse or a discrete switch can alter the hybrid system state.

Majors areas of research in hybrid systems are modeling and analysis. Modeling consists of how to model and describe real systems (almost any analog plant controlled

by a digital program) using hybrid systems theory. Two areas of hybrid systems analysis are stability and reachability analysis. An example of stability analysis is finding the conditions to guarantee system stability for any sequence of discrete transitions, whereas reachability analysis is verifying if a specific state is reachable from a set of initial conditions.

3.2.2 Hybrid Automata

Finite state machines can usually be used to model a hybrid system. Each finite state machine is equipped with a set variables that can change according to the evolution laws defined by the specific discrete state. Each discrete state has a set of guards and assignments, where, if the boolean condition of the guard, *i.e.*, $x > 21$, is satisfied, a transition occurs and the assignment, *i.e.*, on or off, executes. To prevent deadlock, where no transitions are enabled for the current state, an invariant condition must be true whenever the system is executing in that state. The invariant condition could be thought of as a fail-safe mechanism, where if the guard condition fails to trigger a state change, the invariant condition should execute a state change before the system becomes unstable.

A hybrid automaton (self-operating mechanism[1]) is a finite state machine and a finite number of variables. These variables can change continuously, through differential equations, or discretely, according to a specified assignment. For an example of a hybrid automaton, a thermostat, see Fig. 3.1 [24]. In Fig. 3.1, the system starts in

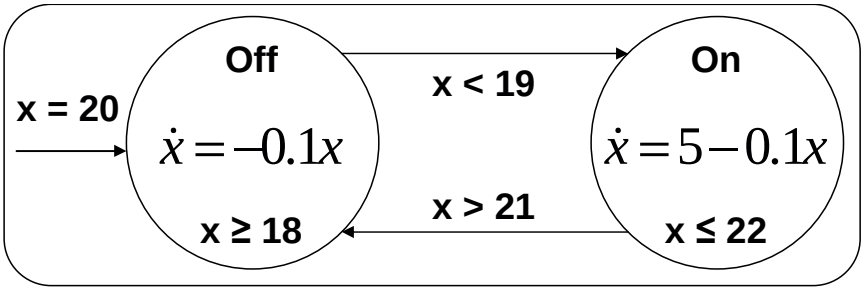

Figure 3.1: Thermostat hybrid automaton.

[1]http://www.dictionary.com

the Off mode with $x = 20$ degrees. In this mode, the temperature decreases according to $\dot{x} = -0.1x$. Due to the guard condition $x < 19$, the heater should switch to the On state as soon as the temperature falls below 19 degrees. At the latest, the heater will move to the On state when the temperature falls below 18 degrees according to the invariant condition $x \geq 18$ degrees. In the On control mode, the temperature increases according to $\dot{x} = 5 - 0.1x$ and the control jumps to the Off mode when the temperature is greater than 21 degrees. Also, the invariant condition, $x \leq 22$ degrees, guarantees that at the latest the heater will move to the Off state when the temperature becomes greater than 22 degrees.

3.3 Hybrid Automata for Perimeter Detection and Tracking

The theory of hybrid systems [25] offers a convenient framework to model the multi-robot system engaged in a perimeter detection and tracking task. In previous work [26], an object-oriented software architecture was developed that supports hierarchical composition of robot agents and behaviors or modes. Key features of the software architecture are summarized below.

- **Architectural hierarchy:** The building block for describing the system architecture is an *agent* that communicates with its environment via shared variables and also communication channels. In this application, the team of mobile sensors defines the *group* node. The group node receives information about the area *i.e.*, boundary where the perimeter is located.

- **Behavioral hierarchy:** The building block for describing a flow of control inside a node is a *mode*. A mode is basically a hierarchical state machine, that is, a mode can have submodes and transitions connecting them. Modes can be connected to each other through entry and exit points. We allow the instantiation of modes so that the same mode definition can be reused in multiple contexts.

- **Discrete and continuous variable updates:** Discrete updates are specified by *guards* labeling transitions connecting the modes. Such updates correspond to mode-switching, and are allowed to modify variables through assignment statements.

The state of a robot node is given by $\boldsymbol{x} \in \mathbb{R}^n$, its evolution is determined by a set of differential equations:

$$\dot{\boldsymbol{x}} = f_q(\boldsymbol{x}, \boldsymbol{u}) \tag{3.1}$$

$$\boldsymbol{u} = k_q(\boldsymbol{x}, \boldsymbol{z}) \tag{3.2}$$

where $\boldsymbol{u} \in \mathbb{R}^m$ is the control vector, $q \in \mathbb{Q} \subset \mathbb{Z}$ is the control mode for the node, $\mathbb{Q}$ is a finite set of control mode indices, $\mathbb{Z}$ denotes the set of positive integers, and $\boldsymbol{z} \in \mathbb{R}^p$ is the information about the external world available either through sensors or through communication channels. The robot node contains modes describing behaviors that are available to the robot. A robot performing a perimeter detection and tracking task is abstracted by a hybrid automaton below.

The overall finite automaton consists of three states: (1) *Random Coverage Controller* (RCC), (2) *Potential Field Controller* (PFC), and (3) *Tracking Controller* (TC). The conglomeration of these three controllers is such that the sensor/robot network is able to locate and track a perimeter. See Fig. 3.2 for a hierarchical state diagram of the cooperative hybrid system developed herein. In the next sections, details of the model used to represent the platform and the controller agents forming the hybrid system are presented.

3.4 Platform and Model

The platform and model used to represent the platform are introduced in this section.

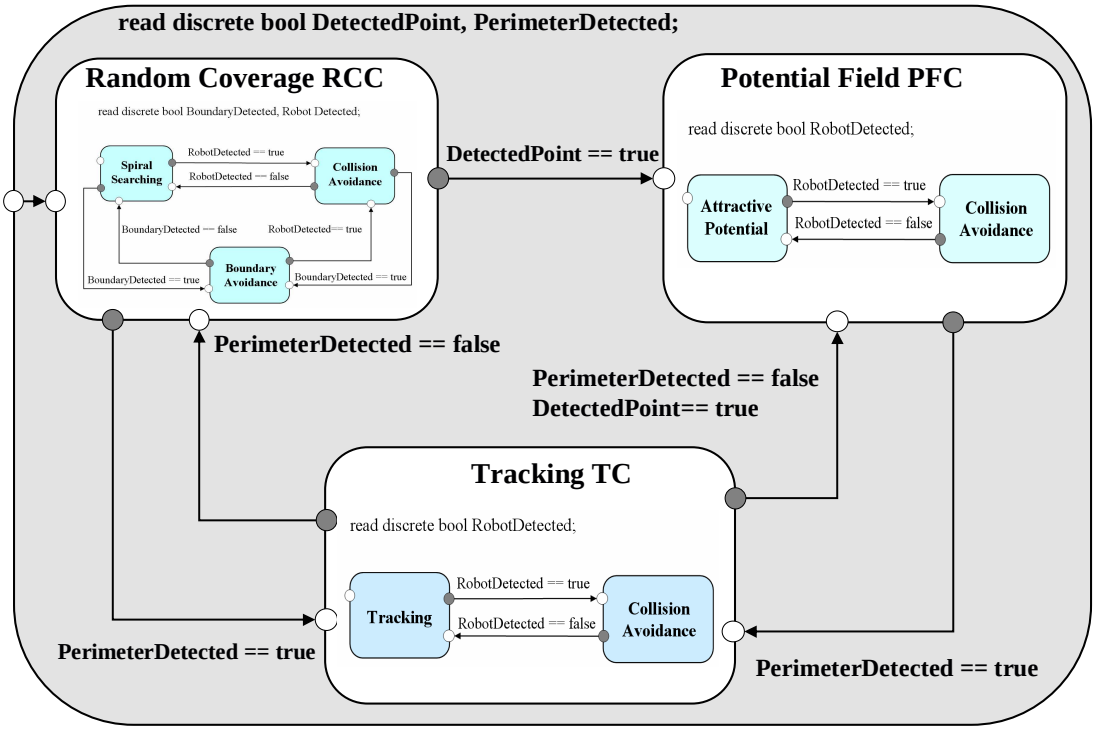

Figure 3.2: Overall finite automaton.

For simplicity, the platform was modeled with the unicycle model:

$$\begin{aligned} \dot{x_i} &= v_i \cos\theta_i \\ \dot{y_i} &= v_i \sin\theta_i \\ \dot{\theta_i} &= \omega_i, \end{aligned} \tag{3.3}$$

where x_i, y_i, θ_i, v_i, and ω_i are the x-position, y-position, orientation angle, linear velocity, and angular velocity of robot i, respectively. v_i ranges from $-2 \leq v_i \leq 2\ m/s$ and this comes from extensive tests of the platform shown in Fig. 3.3 along with the model used to represent it.

Assuming double-steering, the maximum possible angular speed has been calculated using the following formula from [27].

$$\omega_{max} = \frac{v}{r_{min}} = \frac{v}{0.5} = 2v \tag{3.4}$$

where r_{min} is the minimum radius of curvature and $r_{min} = 0.5\ m$ was experimen-

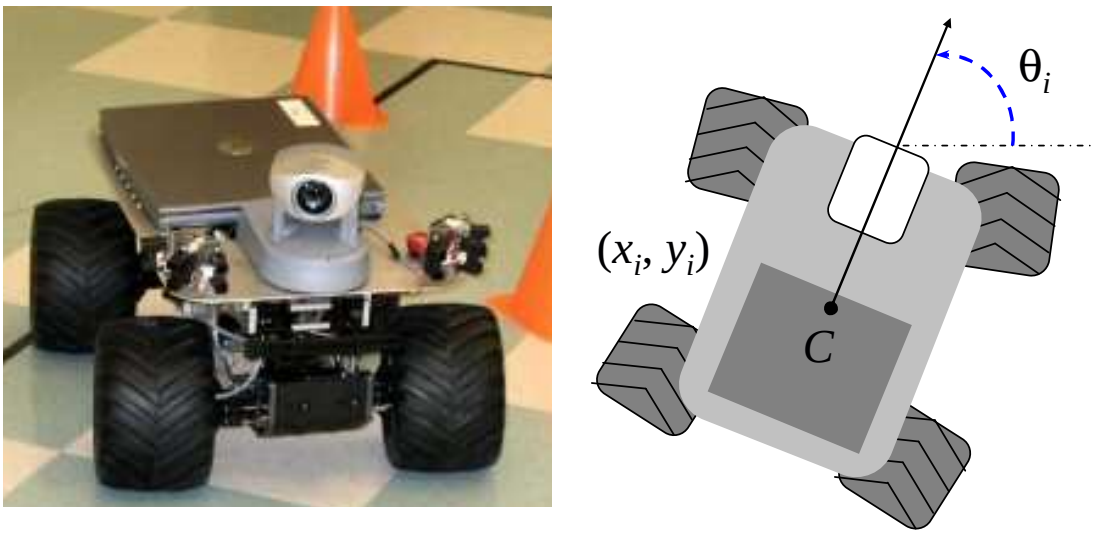

Figure 3.3: (a) Platform and (b) Unicycle model.

tally determined by the author. For this application, the rear servo was removed. Consequently, each robot had single-steering instead of double-steering, in which case $r_{min} = 1\ m$, yielding the following result:

$$\omega_{max} = \frac{v}{r_{min}} = \frac{v}{1} = v \tag{3.5}$$

The actual results varied slightly because equation (3.5) is an approximation of the platform. So, the angular velocity range is approximately $-v_i \leq \omega_i \leq v_i\ rad/s$.

3.5 Controller Agents

The following controllers that create the hybrid system are detailed in this section:

1. Random Coverage Controller (RCC)
2. Potential Field Controller (PFC)
3. Tracking Controller (TC)

3.5.1 Random Coverage Controller

The goal of the Random Coverage Controller (RCC) is to efficiently cover as large an area as possible while searching for the perimeter and avoiding collisions. The robots move fast in this state to quickly locate the perimeter. The RCC consists of three states: (1) *spiral search*, (2) *boundary avoidance*, and (3) *collision avoidance*. Refer to 3.4 for the RCC finite state machine. The spiral search is a random search for

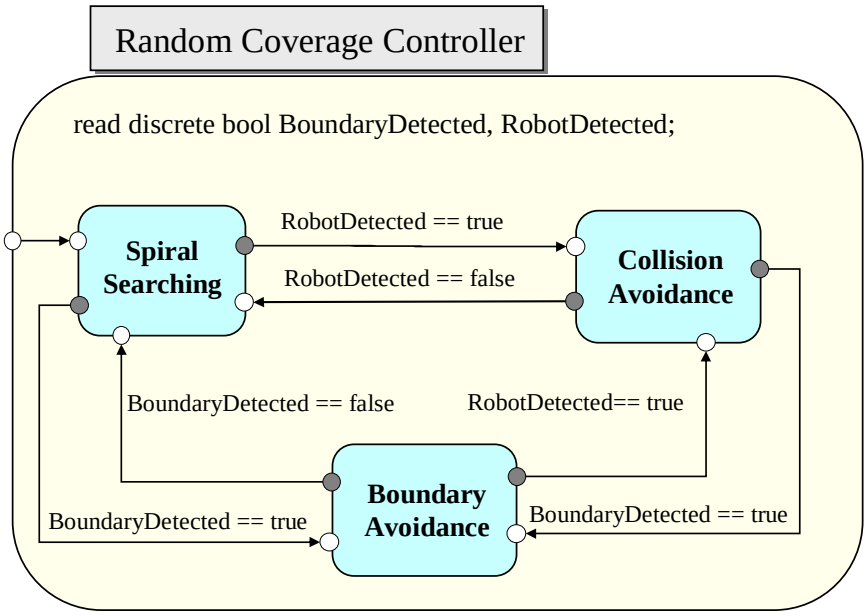

Figure 3.4: Random coverage controller (RCC) finite state machine.

effectively covering the area. The boundary and collisions are avoided by adjusting the angular velocity.

The logarithmic spiral, seen in many instances in nature, is used for the search pattern. In [19], a spiral search pattern such as that used by moths is utilized for searching an area. The spiral search is not optimal, but effective, as shown in [20]. Some examples are hawks approaching prey, insects moving towards a light source, sea shells, spider webs, and so forth. See Fig. 3.5 for an example of a logarithmic spiral in a sea shell[2].

[2]http://en.wikipedia.org/wiki/Logarithmic_spiral

Figure 3.5: Example of logarithmic spiral in nature.

The linear and angular velocity controllers are:

$$v_i = v_s\left(1 - e^{-t}\right) \tag{3.6}$$

$$\omega_i = ae^{b\theta_i}, \tag{3.7}$$

where v_s is a positive constant, a is a constant and $b > 0$. If $a > 0$ (< 0), then the robots move counterclockwise (clockwise). The robots spiral clockwise in this thesis, such as in Fig. 3.6.

The linear velocity is approximately constant when the robot is using the RCC, such as in Fig. 3.7, while the angular velocity is slowly increasing in Fig. 3.8. The errors between the actual and desired velocity are reasonable considering this experiment was performed outdoors on grass. See Figs. 3.9 and 3.10.

Collision and *boundary* (limit of the exploration area here) avoidance are handled in simulation by sharply turning, while in experiments, the robots back up and turn, then go forward. Fig. 3.11 shows an experimental plot of a robot performing obstacle avoidance against a wall. The robot starts at (3.5,0). It senses the wall, backs up,

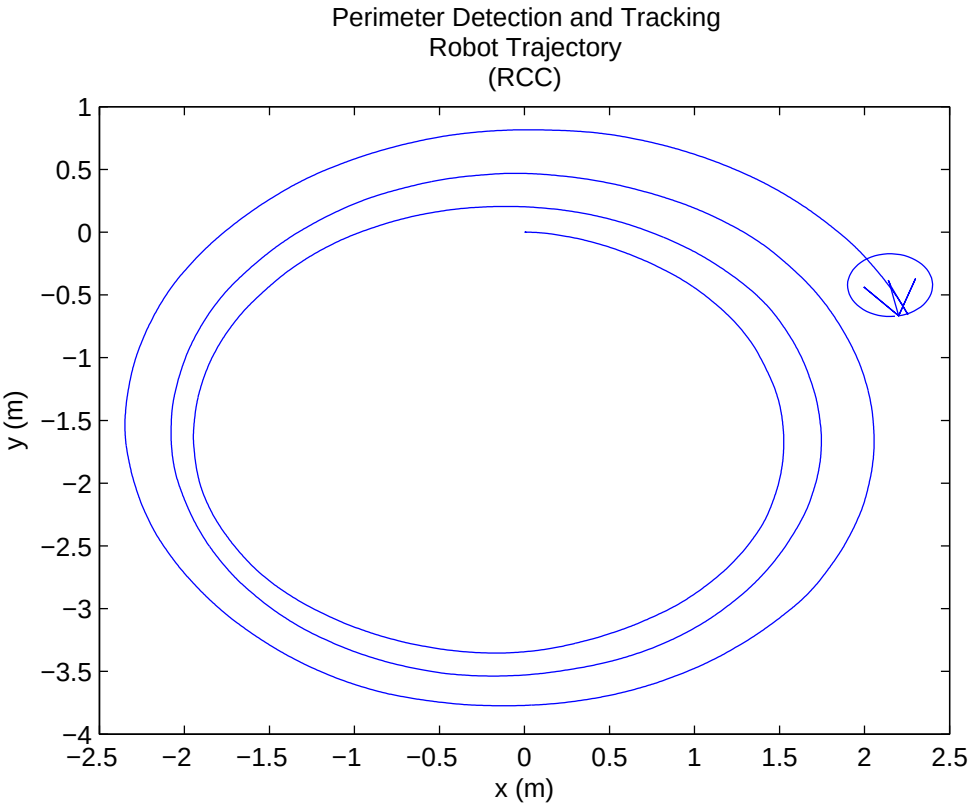

Figure 3.6: Logarithmic spiral search.

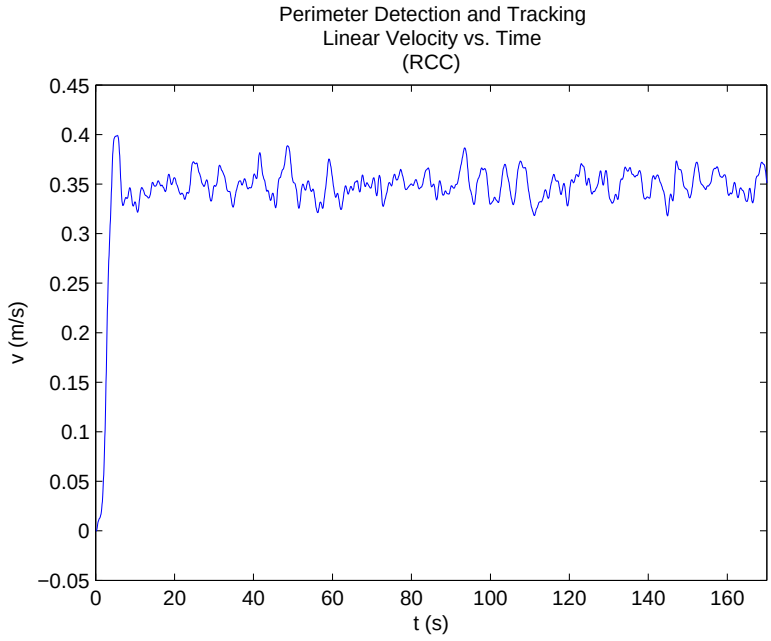

Figure 3.7: Nearly constant linear velocity.

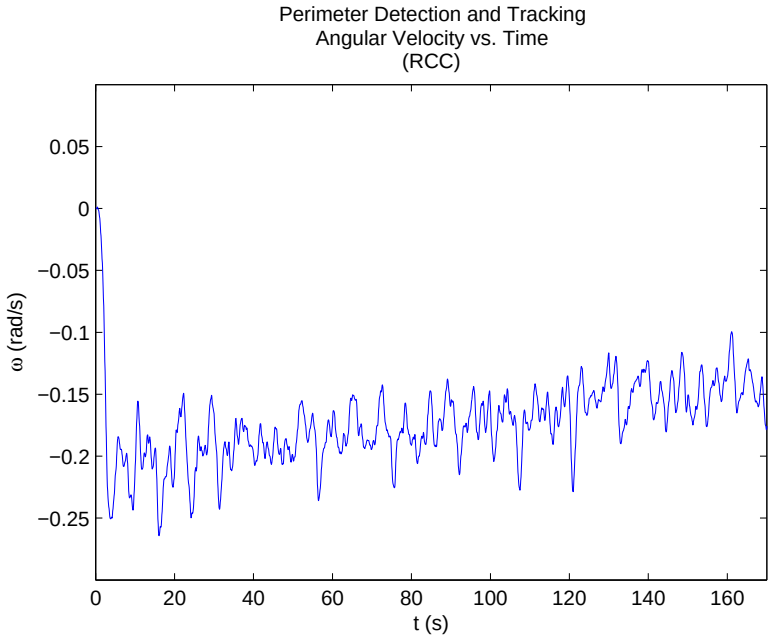

Figure 3.8: Angular velocity slightly increasing.

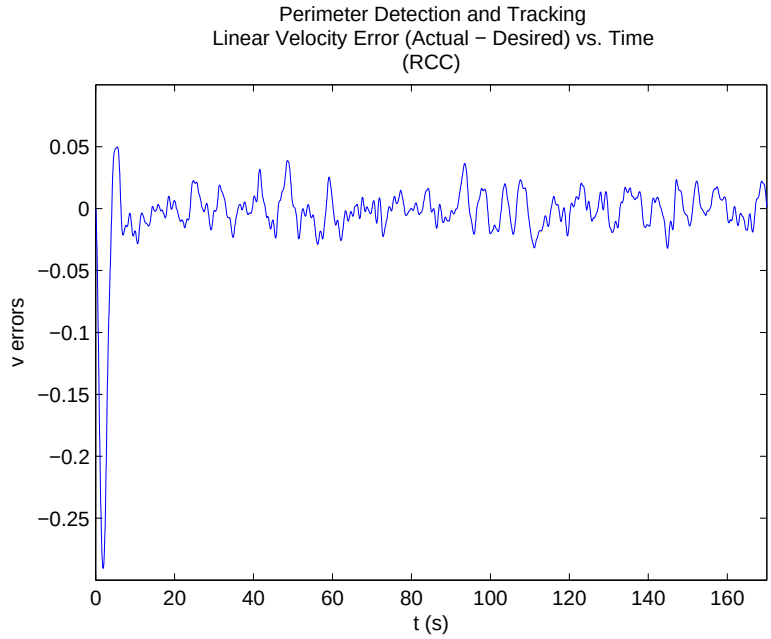

Figure 3.9: Linear velocity error close to zero.

moves forward, and repeats this maneuver until it escapes the wall, upon which the robot begins the spiral search.

The linear velocity is positive and negative in Fig. 3.12, indicating the robot is moving forwards and in reverse. In Fig. 3.13, the robot is always turning right or

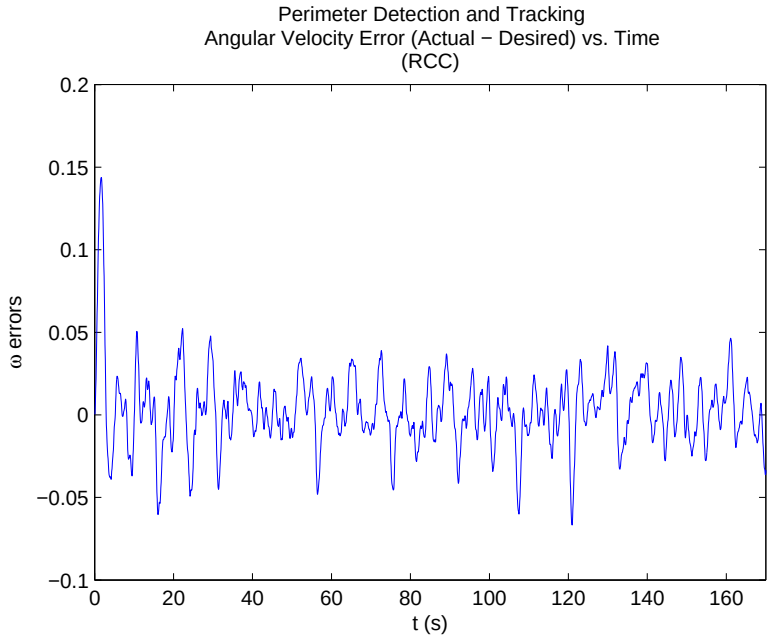

Figure 3.10: Slight angular velocity error.

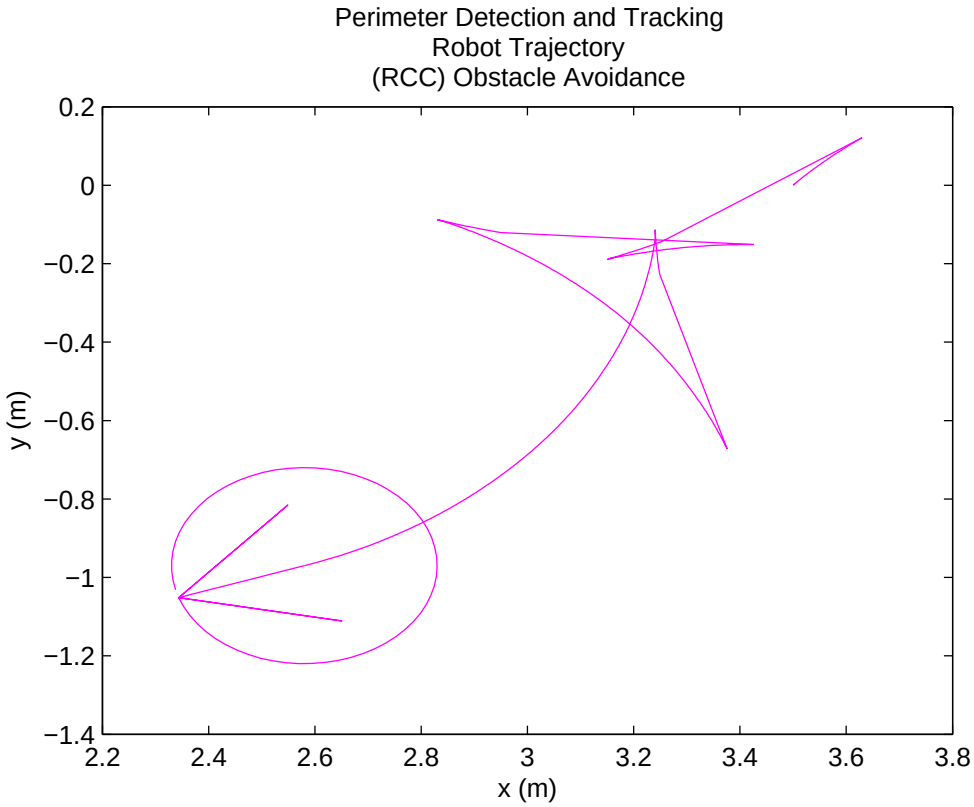

Figure 3.11: Robot avoiding a wall at x=3.8 m.

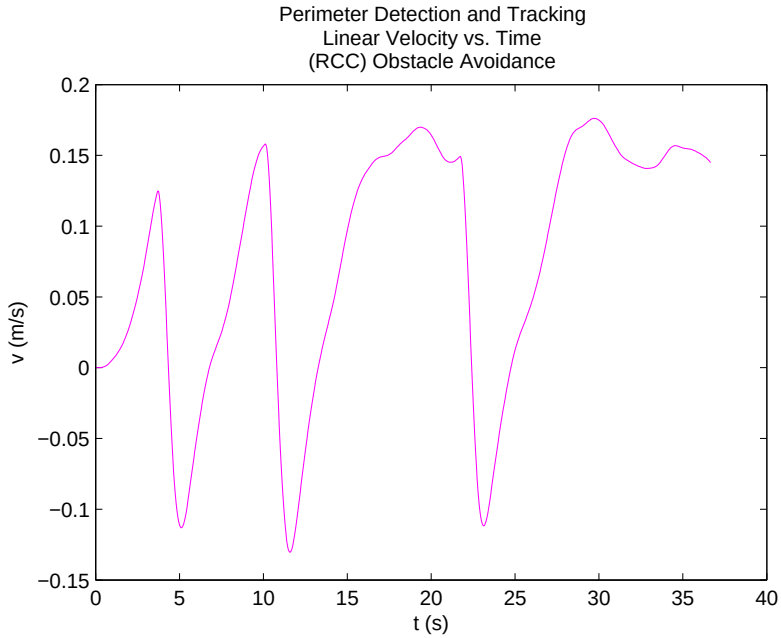

Figure 3.12: Linear velocities indicate forward and reverse movement.

going backwards, so the angular velocity is mostly negative.

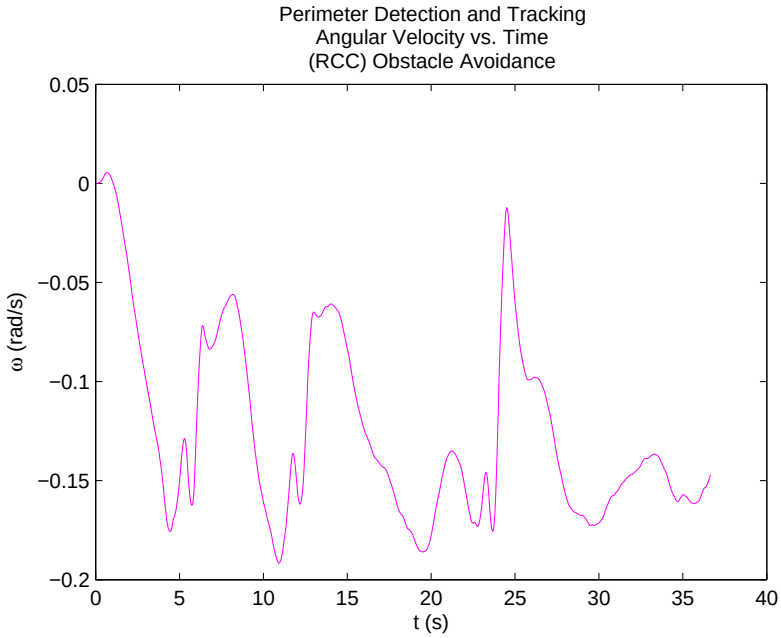

Figure 3.13: Negative angular velocity indicates robot turning right.

Both the linear and angular velocities have a moderate amount of error in Figs. 3.14 and 3.15 because of the delays associated with the robot's motors switching between forwards and reverse.

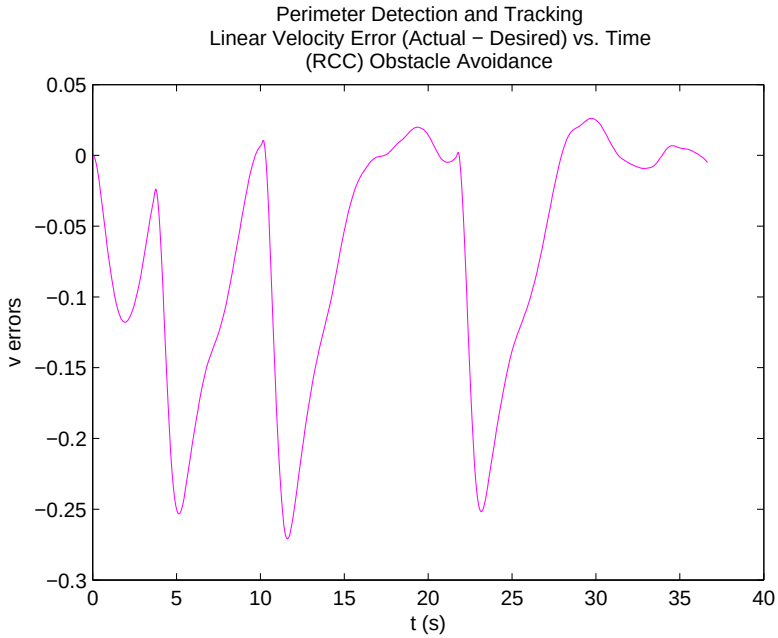

Figure 3.14: Moderate linear velocity error from switching.

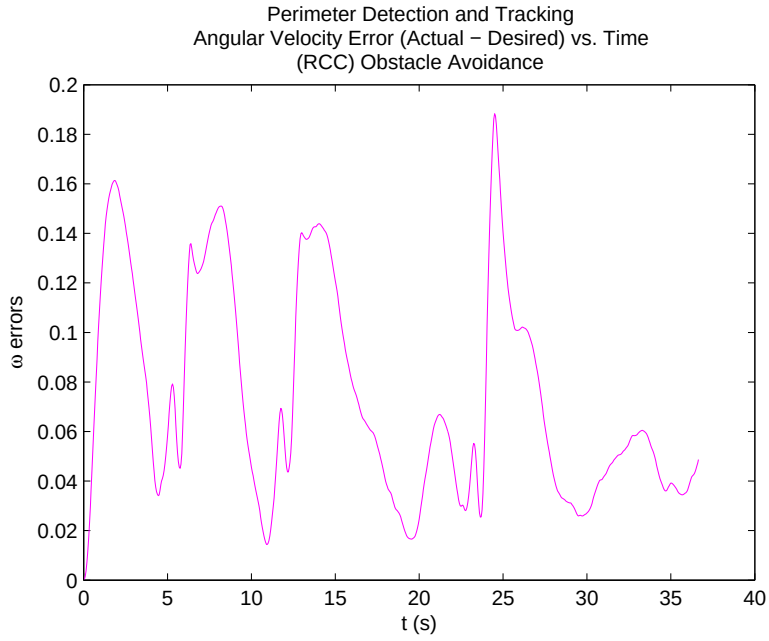

Figure 3.15: Moderate angular velocity error from switching.

3.5.2 Potential Field Controller

Potential fields have been used by a number of groups for controlling a swarm [28, 8, 7, 2]. The majority of the time when potential field methods are used, attractive

(goals) and repulsive (obstacles) potentials are created. See Fig. 3.16[3] for an example of the attractive and repulsive potentials created when using this method. In Fig.

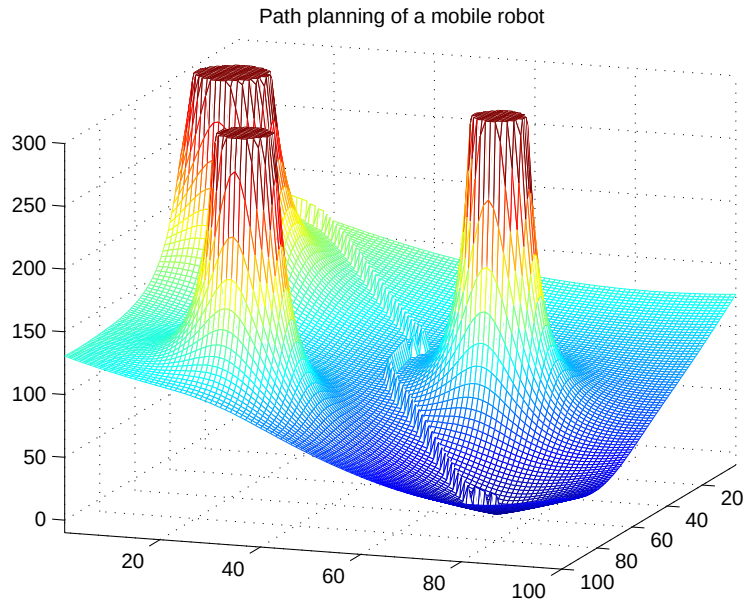

Figure 3.16: Attractive (dark blue) and repulsive (dark red) potentials showing a robot path to a goal.

3.16, dark blue indicates more attractive potentials, whereas dark red indicates more repulsive potentials. To use attractive and repulsive potentials, the locations of the goal(s) and obstacle(s) must be known. For perimeter detection and tracking, only the goal was known once a robot located the perimeter, so only an attractive potential was used.

The Potential Field Controller (PFC) uses an attractive potential allowing the robots to quickly move to the perimeter once it has been detected. The first robot to detect the perimeter *broadcasts* its location to the other robots, who upon receiving the goal location can broadcast also, forming a relay communication scheme. If a robot is within range, the PFC is used to quickly move to the perimeter. Otherwise, the robot will continue to use the RCC unless it comes within range of another robot that has already detected the perimeter, at which point the robot will switch to the PFC. See Fig. 3.17 for an example of the relay scheme used.

[3]http://arri.uta.edu/acs/jmireles/Robotics/PathPlanMobileRobots.pdf

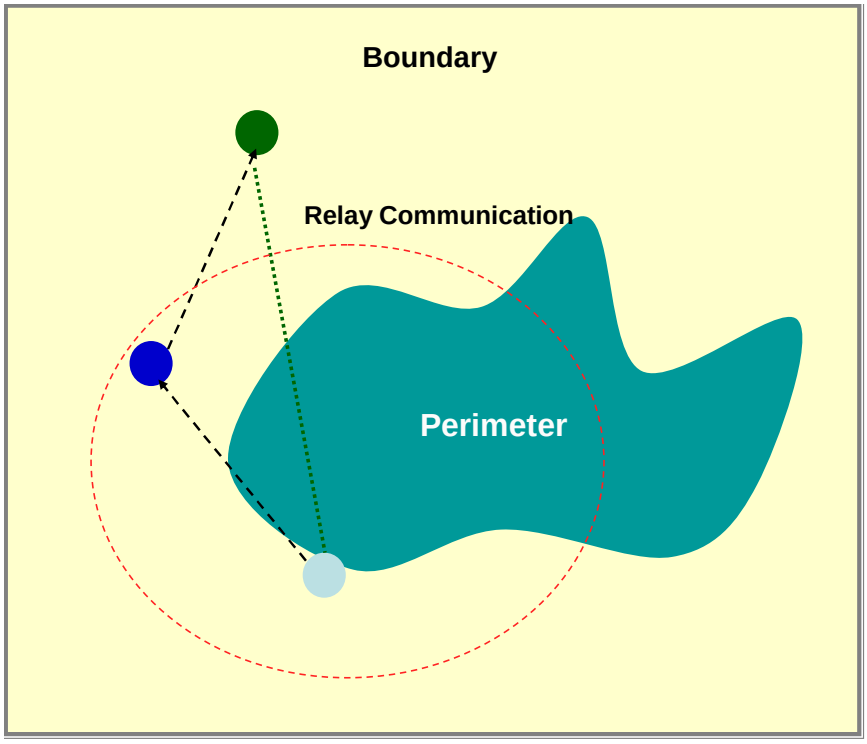

Figure 3.17: Relay communication example.

As a robot moves towards the goal and detects the perimeter before it reaches the goal, it will switch to the TC. The PFC has two states: (1) *attractive potential* and (2) *collision avoidance* (same as RCC). Refer to Fig. 3.18 for the PFC finite state machine.

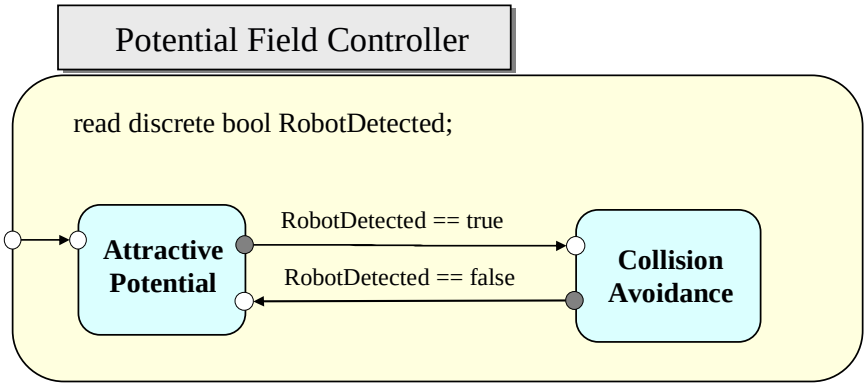

Figure 3.18: Potential field controller (PFC) finite state machine.

The attractive potential, $\mathbf{P_a}(x_i, y_i)$, is [29]:

$$\mathbf{P_a}(x_i, y_i) = \frac{1}{2}\epsilon[(x_i - x_g)^2 + (y_i - y_g)^2], \tag{3.8}$$

where (x_i, y_i) is the position of robot i, ϵ is a positive constant, and (x_g, y_g) is the position of the attractive point (goal). The attractive force, $\mathbf{F_a}(x_i, y_i)$, is derived below.

$$\begin{aligned}
\mathbf{F_a}(x_i, y_i) &= - \bigtriangledown \mathbf{P_a}(x_i, y_i) \\
\mathbf{F_a}(x_i, y_i) &= - \begin{bmatrix} \frac{\partial \mathbf{P_a}}{\partial x_i} \\ \frac{\partial \mathbf{P_a}}{\partial y_i} \end{bmatrix} \\
\mathbf{F_a}(x_i, y_i) &= \epsilon \begin{bmatrix} x_g - x_i \\ y_g - y_i \end{bmatrix} = \begin{bmatrix} F_{a,x_i} \\ F_{a,y_i} \end{bmatrix}
\end{aligned} \tag{3.9}$$

Equation (3.9) is used to get the desired orientation angle, $\theta_{i,d}$, of robot i:

$$\theta_{i,d} = \arctan 2(F_{a,y_i}, F_{a,x_i}) \tag{3.10}$$

Depending on θ_i and $\theta_{i,d}$, the robot will turn the optimal direction to quickly line up with the goal using the following proportional angular velocity controller:

$$\omega_i = \pm k \ (\theta_{i,d} - \theta_i) \tag{3.11}$$

where $k = \frac{\omega_{max}}{2\pi}$ and $\omega_{max} = 0.4 \ rad/s$. Each robot moves with constant linear speed when using the PFC.

In Fig. 3.19, robot 1 begins in the RCC at (3.5, 0). Robot 2 (not shown) was placed on the perimeter so it could broadcast the goal location to the robot 1. When robot 1 receives the goal located at (2.5, 2.5), it turns left and moves toward the goal using the PFC. Upon detection of the perimeter, robot 1 begins tracking.

The state transitions are shown in Fig. 3.20. The robot moves with nearly constant linear velocity in Fig. 3.21. In Fig. 3.22, the robot initially is turning right, but upon entering the PFC, turns left. As the robot detects the perimeter, it turns right and then left again as it is tracking the perimeter. The error in linear velocity is fairly small, as shown in Fig. 3.23. Large errors occur in the angular velocity in

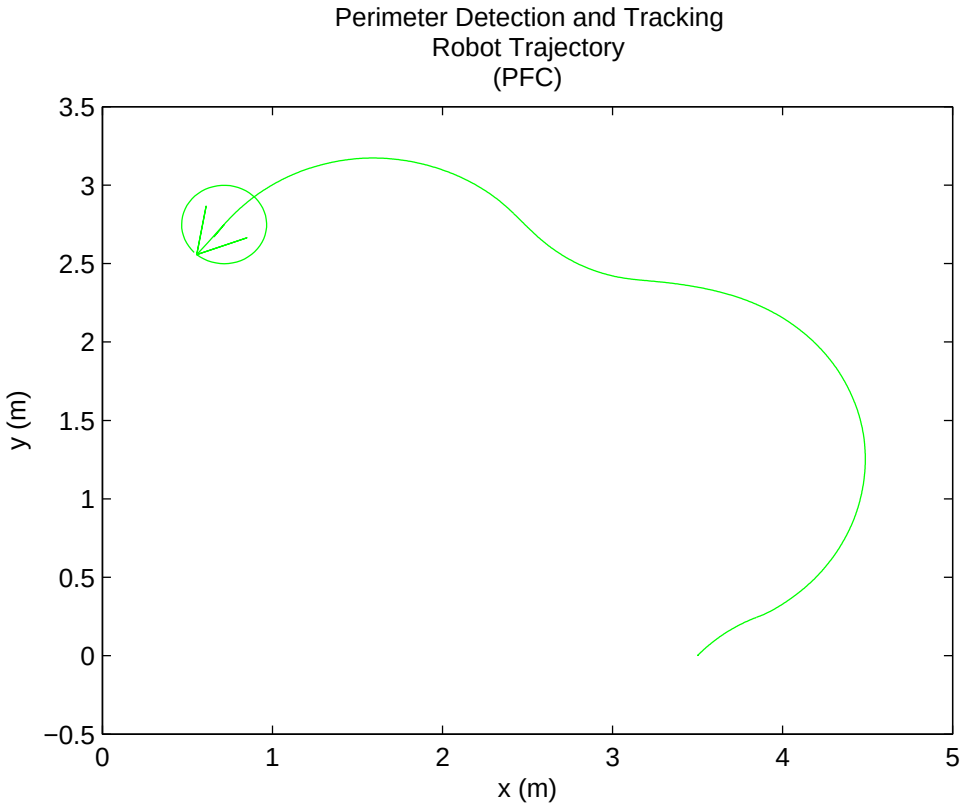

Figure 3.19: Robot shown receiving the goal location, it then moves towards and tracks the perimeter, which begins at (2.5, 2.5).

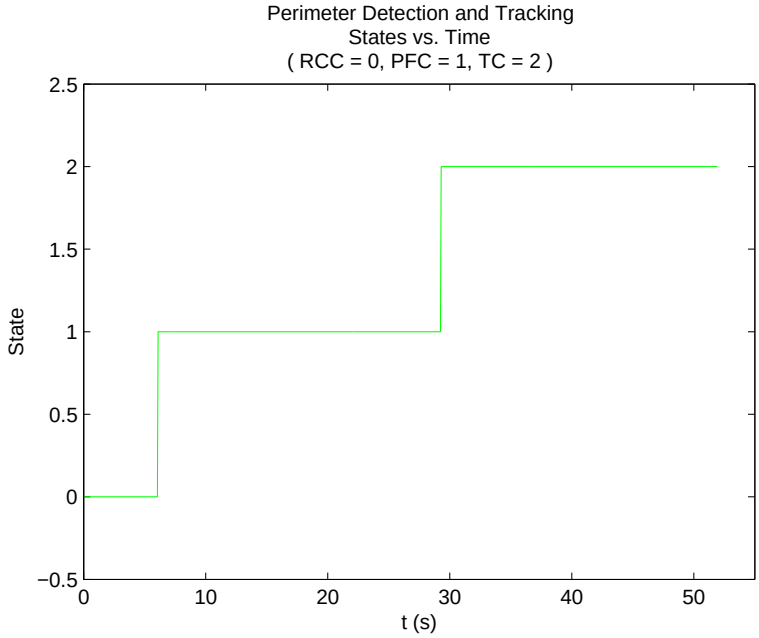

Figure 3.20: Discrete state transitions showing the robot entering the (PFC), then the (TC).

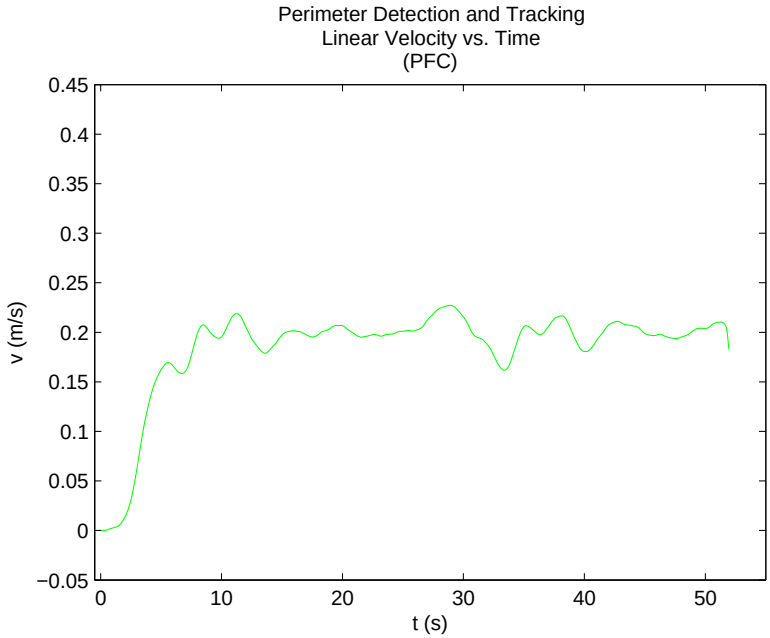

Figure 3.21: Nearly constant linear speed.

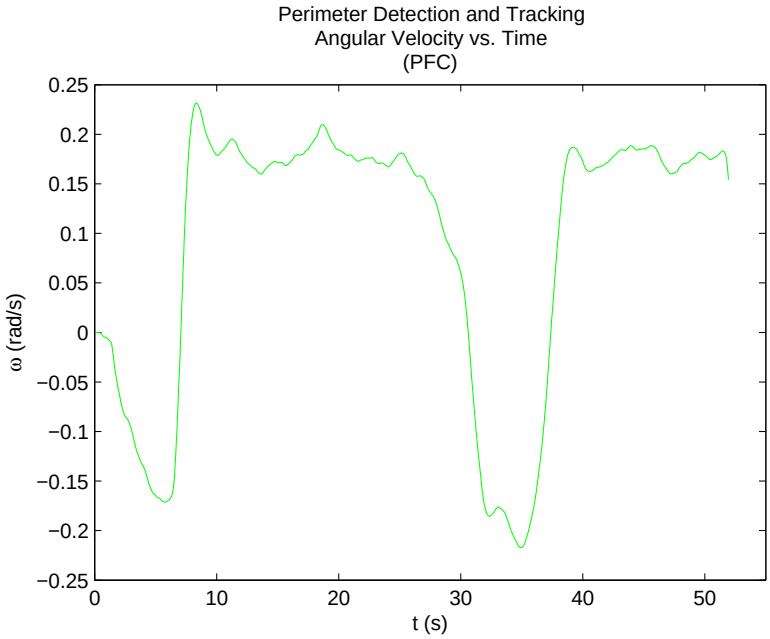

Figure 3.22: Angular velocity showing robot turning right (RCC), turning left (PFC), and then turning right and then left as it finds and tracks the perimeter (TC).

Fig. 3.24 as the robot turns back and forth.

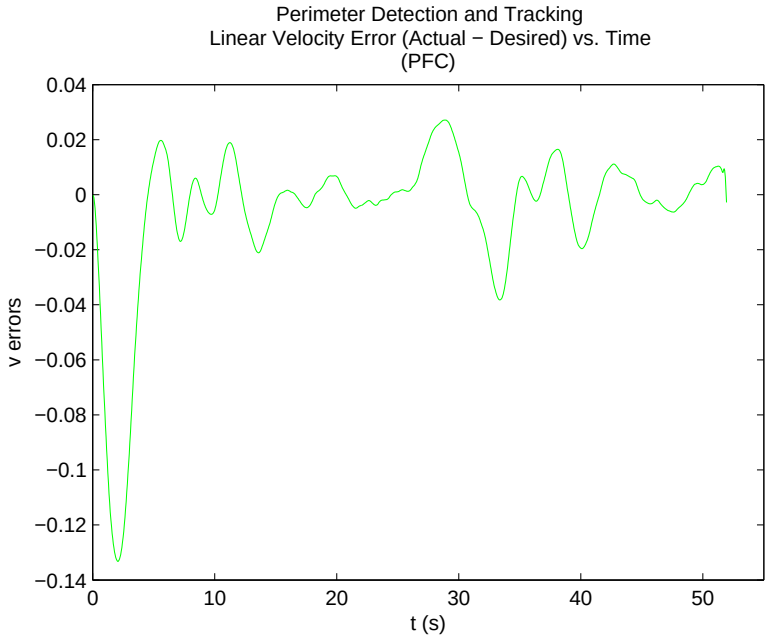

Figure 3.23: Small linear velocity error.

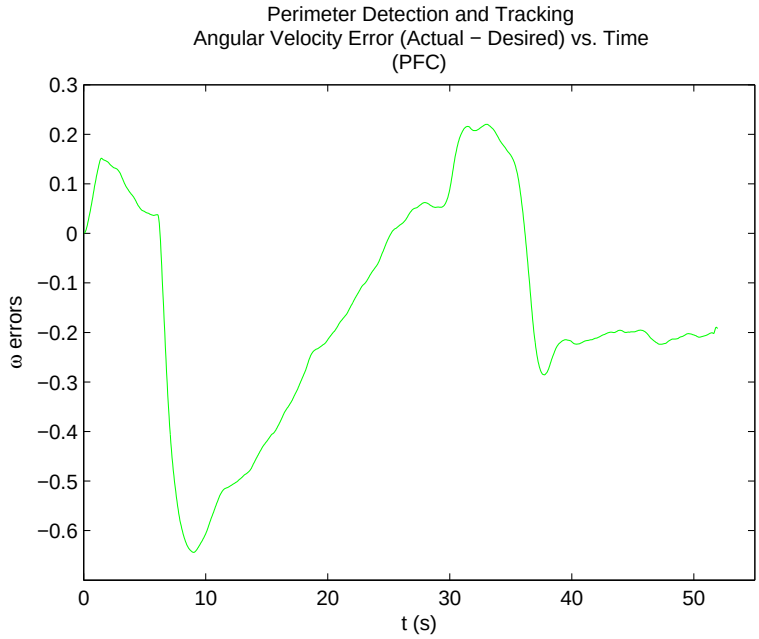

Figure 3.24: Large angular velocity error as the robot turns.

3.5.3 Tracking Controller

The Tracking Controller (TC) changes ω and v in order to track the perimeter and avoid collisions, respectively. Cyclic behavior *emerges* as multiple robots track the perimeter. The TC differs from [30] in that each robot's objective is to track the

perimeter, while avoiding collisions. There are no restrictions on robot order, v is not constant, and orientation is controlled by either a bang-bang (Matlab) or proportional (Gazebo and experiments) controller.

The robots' goal in this state is to accurately track the perimeter counterclockwise. The TC consists of two states: (1) *tracking* and (2) *collision avoidance*. Refer to Fig. 3.25 for the TC finite state machine. Tracking is accomplished by adjusting the

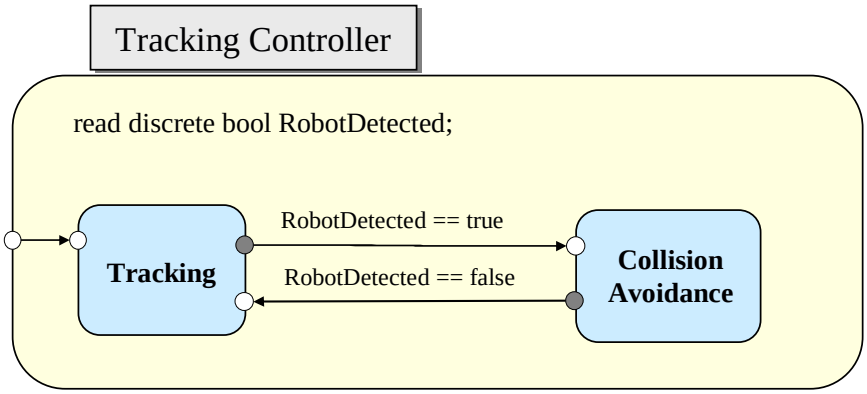

Figure 3.25: Tracking controller (TC) finite state machine.

angular velocity. On the other hand, collisions are avoided by changing the linear velocity.

The angular velocity controller in Matlab is:

$$\omega_i = \begin{cases} -\omega_t & \text{inside perimeter} \\ \omega_t & \text{outside perimeter,} \end{cases} \tag{3.12}$$

where $\omega_t = 0.1\ rad/s$. A look-ahead distance is defined to be the sensor point directly in front of the robots. If the omnidirectional sensor has detected the perimeter and the look-ahead distance is inside the perimeter, the robot will turn right. Otherwise, the robot turns left. This zigzagging behavior is often seen in moths following a pheromone trail to its source [31].

Otherwise, the angular velocity controller is:

$$\omega_i = k_p\ (\gamma_{out} - \gamma_{in}), \tag{3.13}$$

where $k_p > 0$, and γ_{out} and γ_{in} are the areas outside and inside the perimeter seen by the camera (blobfinder), respectively. This controller is described in more detail in Chapter 4.

A robot is shown tracking a static perimeter in Fig. 3.26. In Fig. 3.27, the linear

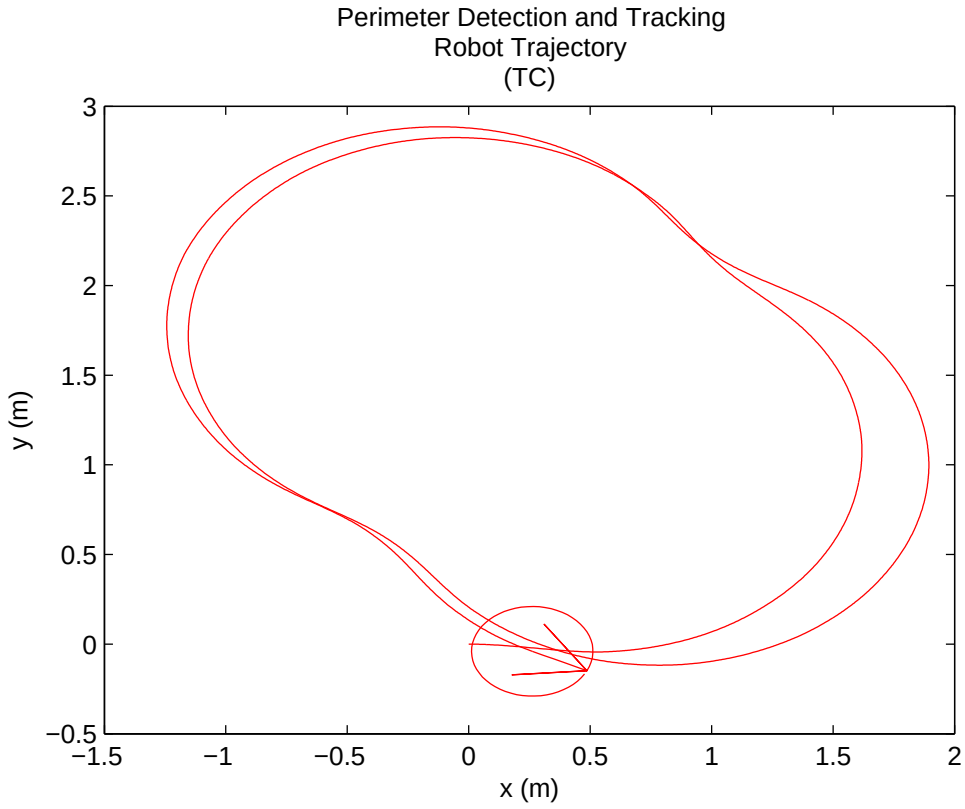

Figure 3.26: Robot tracking a static perimeter.

velocity is nearly constant. The angular velocity is sinusoidal in Fig. 3.28, meaning the robot is tracking the perimeter. In Fig. 3.29, the error in linear velocity is near zero. The moderate amount of error in the angular velocity in Fig. 3.30 shows that the robot is tracking and turning back and forth.

Solo tracking occurs only if one robot has detected the perimeter and in this state the linear velocity is constant. Otherwise, cooperative tracking occurs when two or more robots have sensed the perimeter. Equation (3.12) is still used to track the perimeter, but now the linear velocity is adjusted to allow the swarm to distribute around the perimeter. The robots communicate their separation distances to each other and each robot only reacts to its nearest neighbor on the perimeter. The swarm

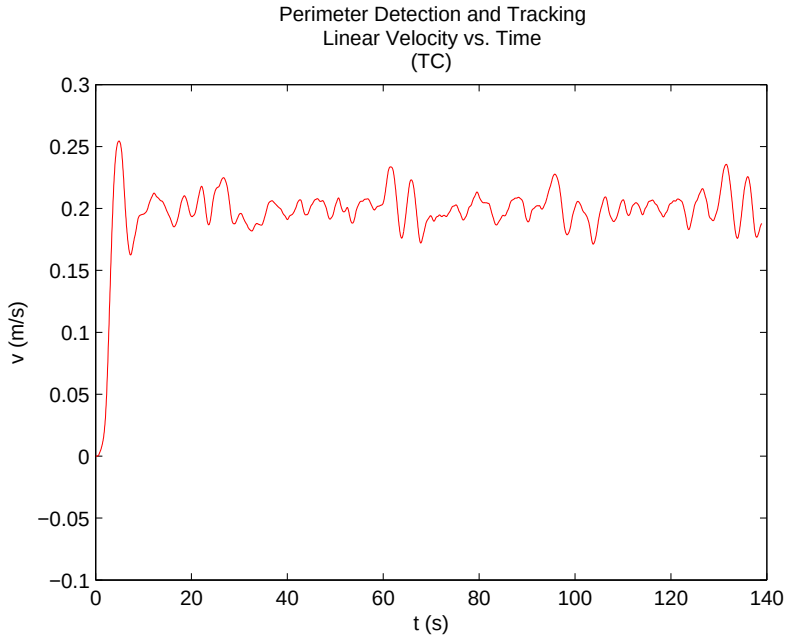

Figure 3.27: Nearly constant linear speed.

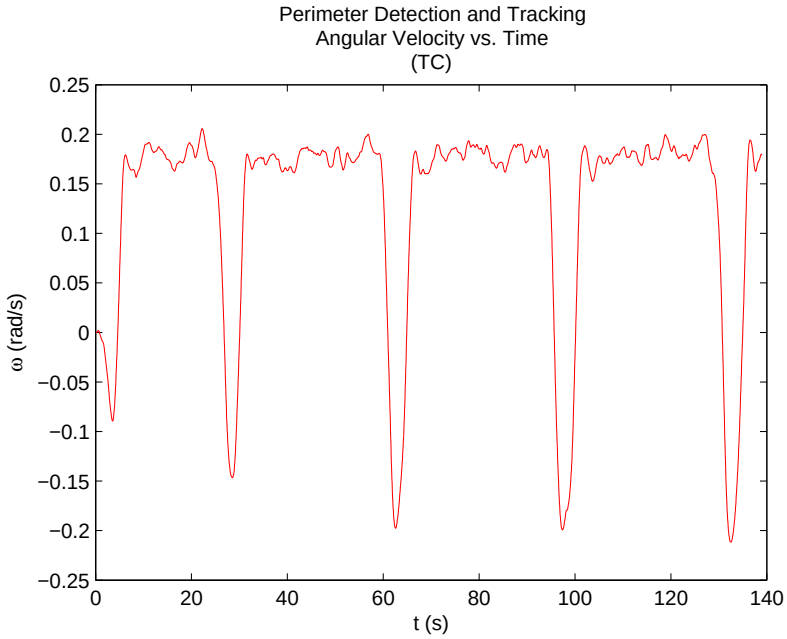

Figure 3.28: Sinusoidal angular velocity indicating the robot is tracking.

will attempt to uniformly distribute around the perimeter at a desired separation

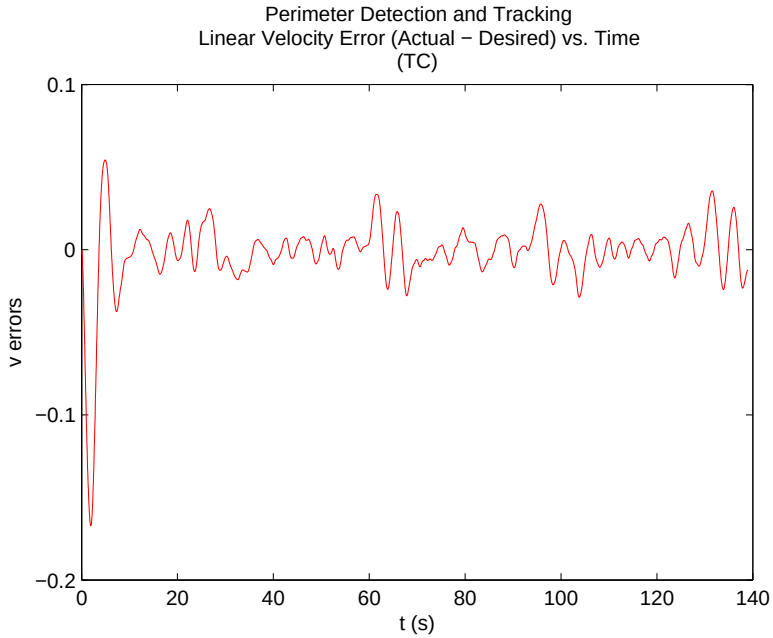

Figure 3.29: Small linear velocity error.

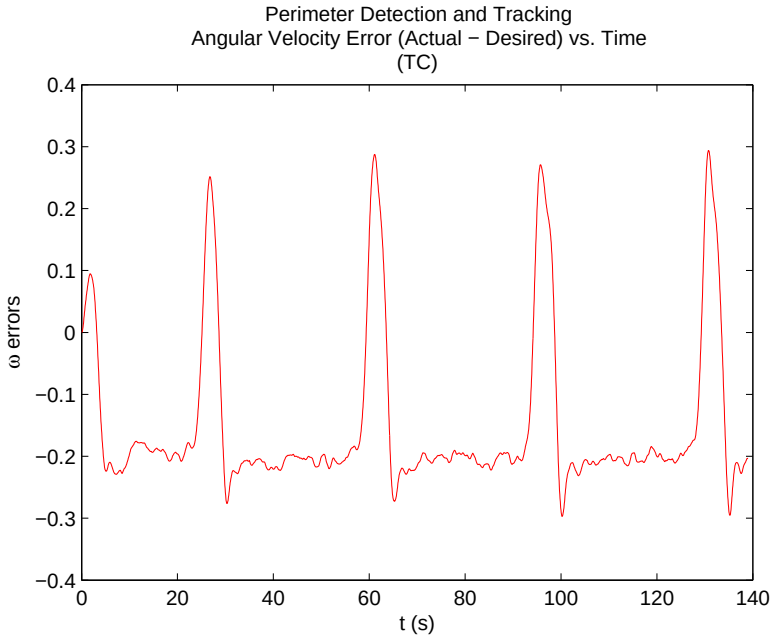

Figure 3.30: Moderate angular velocity error (also sinusoidal).

distance, d_{des}, using the following linear velocity proportional controller:

$$v_i = \begin{cases} 0 & |d_{ij} - d_{des}| < \epsilon \\ k_p|d_{des} - d_{ij}| & \text{otherwise} \end{cases} \tag{3.14}$$

where d_{ij} is the distance from robot i to robot j, $\epsilon = 0.01$, and $k_p = \frac{v_{max}}{|d_{des} - d_{ij,max}|} \simeq 0.06$.

When the swarm is uniformly distributed, it stops. This allows each robot to conserve its resources. If the perimeter continues to expand or robots are added or deleted, the swarm will reconfigure until the perimeter is uniformly surrounded. However, uniformly distributing around a dynamic perimeter may be difficult to achieve.

The choice of d_{des} is critical to the swarm's behavior. If d_{des} is too low, the swarm will not surround the perimeter. Subgroups may also be formed since each robot only reacts to its nearest neighbor. If d_{des} is too high, the swarm will surround the perimeter, but group cycling emerges and the robots will not stop.

Collision avoidance is handled inherently in the controller. Since the robots can communicate, collisions should never occur.

The cooperative tracking method described above has only been implemented in Matlab, not in Gazebo or experimentally. Using euclidean distances instead of the distance apart along the perimeter is not an effective way to distribute around the perimeter, hence, distribution will not work for many irregular-shaped perimeters. Cooperative tracking will only work effectively for the majority of perimeters if the perimeter location can be accurately estimated.

Chapter 4

Simulation Results

4.1 Introduction

Matlab[1] was initially used to simulate the hybrid system. Once the system worked in Matlab, Gazebo, a simulator developed at USC, was used to verify the simulation results from Matlab [32]. Results from both simulators are described below for static and dynamic perimeters.

4.2 Simulations from Matlab

Described in this section are the static and dynamic perimeter simulations from Matlab. Accompanied with these descriptions are figures that present the algorithm working with and without communication for static perimeters and with communication for dynamic perimeters.

4.2.1 Static Perimeter

In the following simulations, five to 11 nonholonomic[2] robots are shown tracking a static perimeter with and without communication. The omnidirectional sensor and communication ranges are 3 m and 25 m, respectively.

[1] http://www.mathworks.com

[2] It is common in the literature to use simplified linear models.

In Fig. 4.1, there is no communication available, thus, each robot must search for and locate the perimeter on its own. After 300 s, all of the robots except robot 1

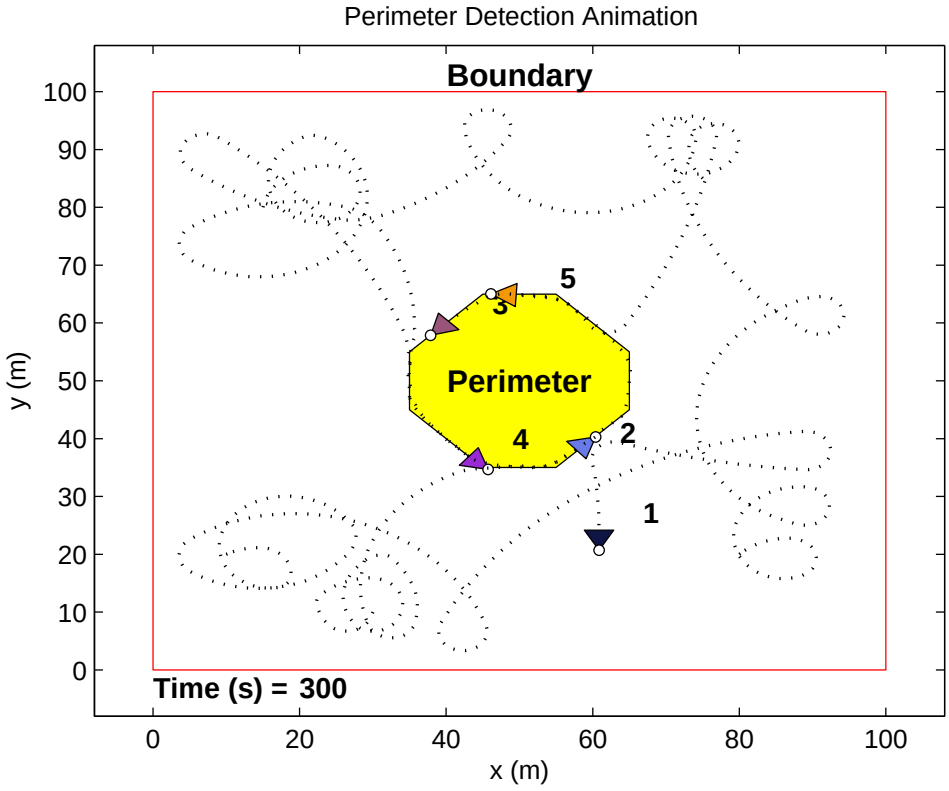

Figure 4.1: Five robots tracking a static perimeter without communication.

reach the perimeter in Fig. 4.1. Robot 1 detects the perimeter, however, it is tracking clockwise instead of counterclockwise. Therefore, robot 1 continues searching for the perimeter. If communication were available, robot 1 would reach the perimeter.

The same simulation as in Fig. 4.1 is run again with communication now available (see Fig. 4.2) to see if the communication relay improved the performance of the system. Specifically, the communication relay is established in Fig. 4.2. The robots start searching, robot 5 locates the perimeter first and begins broadcasting the goal location. As robot 5 is tracking the perimeter, robot 1 comes within range, receives the goal location, and is able to reach the perimeter. Robot 3 also receives the goal location, reaching the perimeter slightly quicker than during the previous simulation. Robots 2 and 4 find the perimeter on their own.

With 11 robots and communication available, after 450 s, all robots locate the

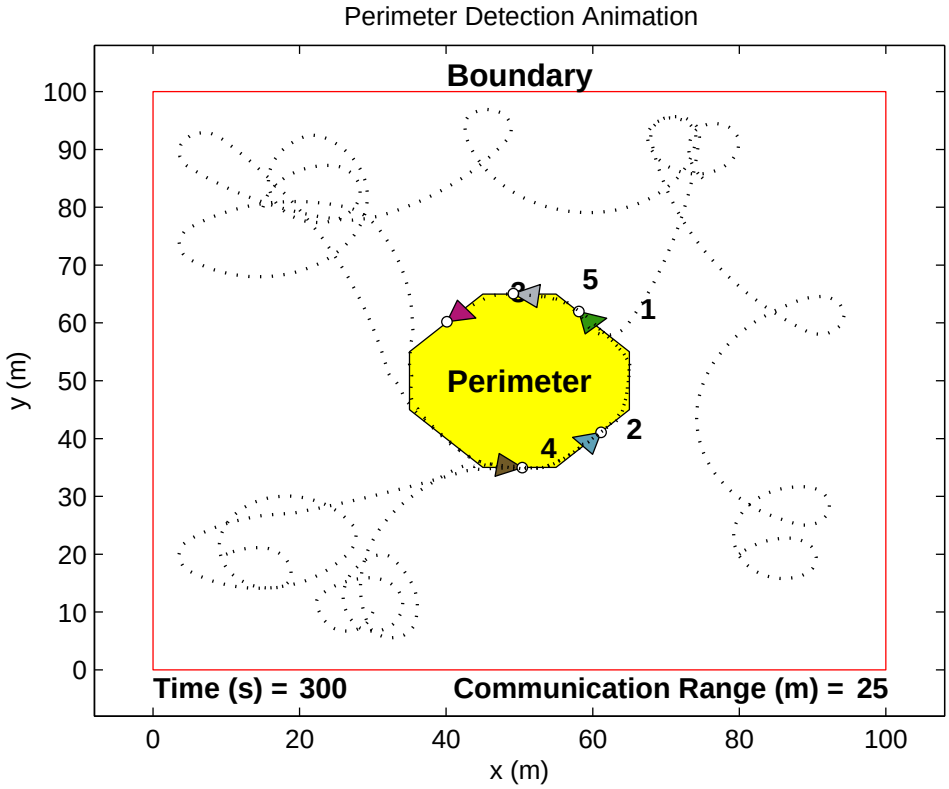

Figure 4.2: Five robots tracking a static perimeter with communication.

perimeter. See Fig. 4.3. In Fig. 4.3, robots 4 and 11 received the goal location, but had difficulty reaching the perimeter. Each one was avoiding the others that were tracking, but eventually reached the perimeter. No collisions occurred during simulation. Paths intersect, but at different instances of time.

4.2.2 Dynamic Perimeter

In the following simulation, D_{com}, D_{sen}, and D_{sep} are the communication range, the sensor range, and the desired separation distance, respectively. Shown in Fig. 4.4, five robots are tracking a dynamic perimeter that is expanding at 12.5 mm/s.

Initially, all robots are searching for the perimeter in Fig. 4.4. Robot 4 locates the perimeter first. Robot 3 is within range and receives the location. Once robot 3 reaches the perimeter, it stops until robot 4 reaches D_{des}. As robot 4 is tracking the perimeter, robot 2 comes within range, receives the location, and moves toward it. As robot 2 nears the perimeter, it avoids a collision with robot 4 and reaches the

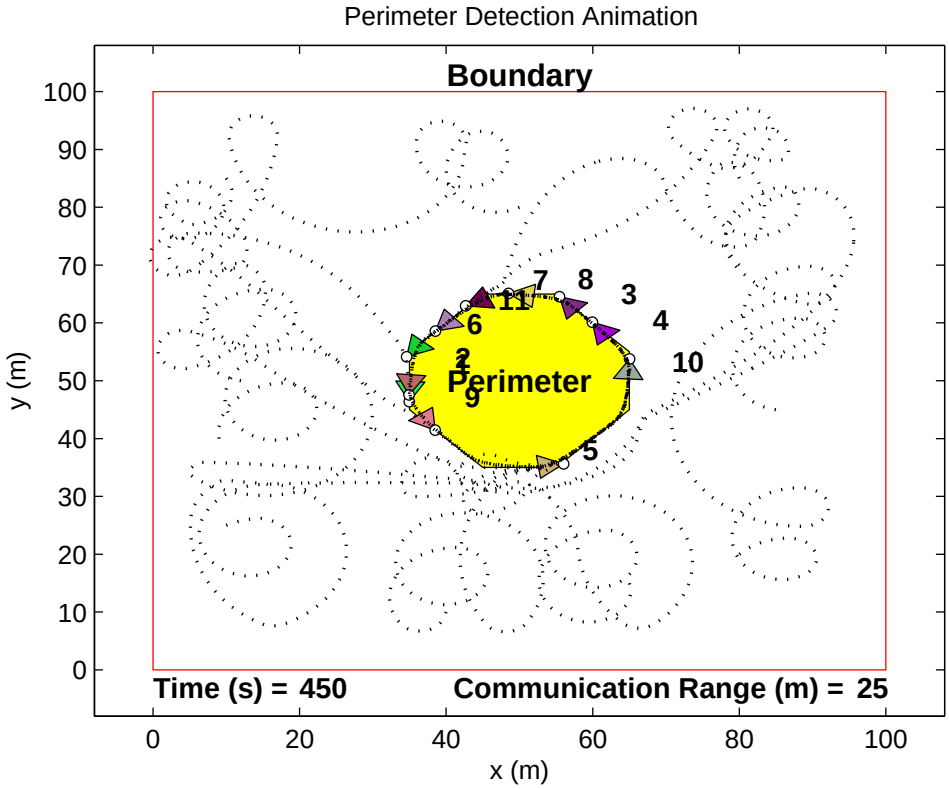

Figure 4.3: 11 robots tracking a static perimeter.

perimeter. Robots 1 and 5 locate the perimeter on their own. When robot 5 reaches the perimeter, it is tracking the wrong way, so it quickly turns around. Subgroups are formed because there are not enough robots to distribute around the perimeter. The swarm does not stop because the expanding perimeter is never uniformly surrounded.

Unfortunately, some of the system behavior, such as communication and collision avoidance, is not apparent when viewing these static plots. However, movies were made for a number of simulations to show the different behaviors. These movies are available electronically and show a number of behaviors including reconfiguration, collision avoidance, and some unstable behavior with 15 robots. The code for the controller logic is shown in Appendix A.1.

Writing all Matlab functions as MEX (Matlab executable) files is recommended. MEX files allow the user to write code in C, which Matlab can then access like a normal Matlab function. MEX files are more efficient than Matlab functions. In hindsight, writing all of the functions as MEX files would save an enormous amount

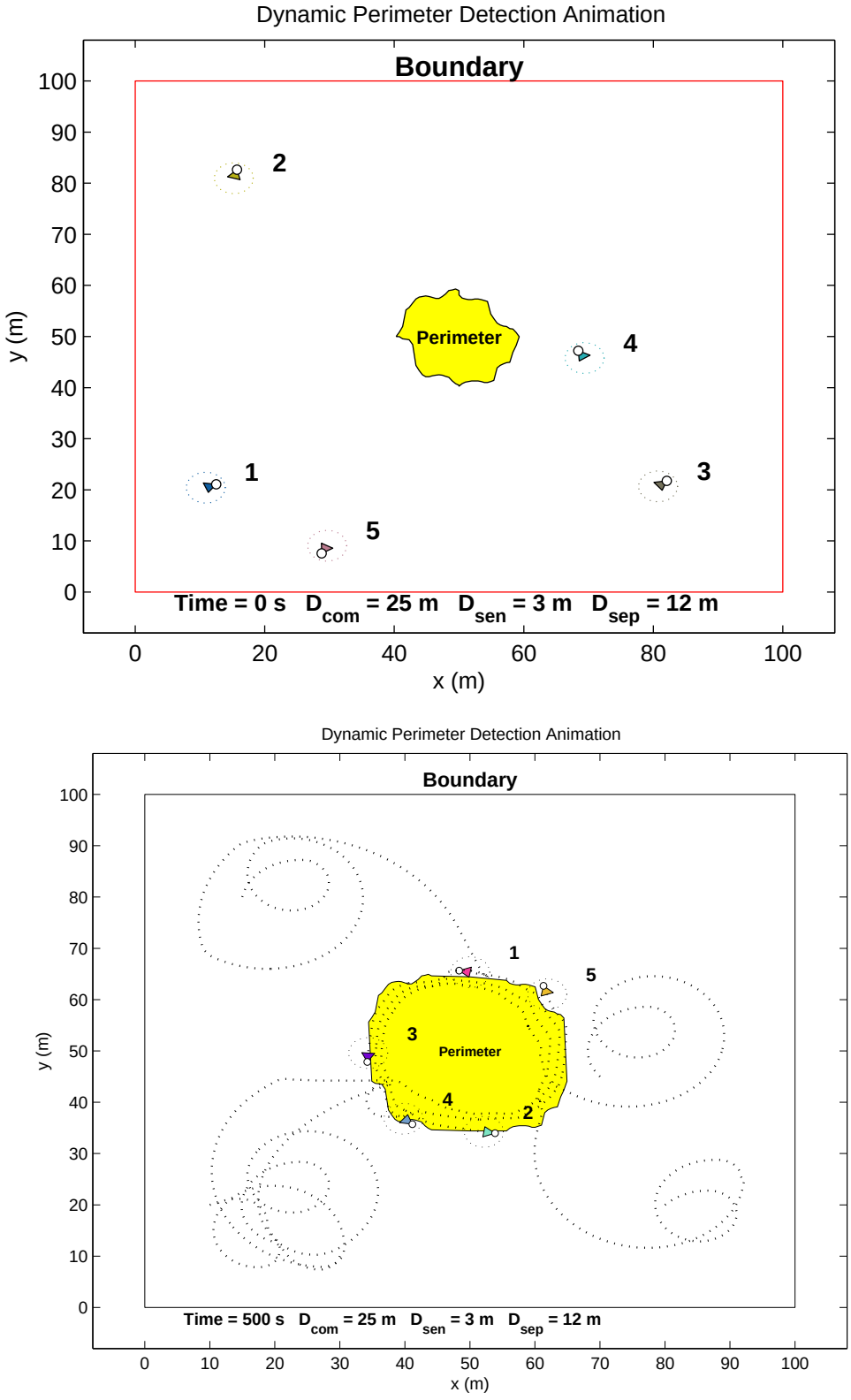

Figure 4.4: Five robots tracking a dynamic perimeter expanding at 12.5 mm/s: (a) Initial configuration and (b) Final configuration.

of time. When simulating a complex hybrid system such as this, simulation time ran up to 19 hours on a dual Intel Xeon processor depending on the number of robots. MEX files would have significantly reduced the required simulation time.

4.3 Simulations from Gazebo

Gazebo served as a bridge between Matlab and the real world since it models real world physics (sensors, robots, and the environment) fairly accurately. If the system works in Gazebo, then ideally the system should also work experimentally without any modification to the code.

Gazebo is an open-source environment simulator that allows virtual robots to be simulated by connecting the driver from Player[3] to Gazebo. Player is a server for robotic communication that is language independent and works under UNIX-based operating systems. It provides an interface to controllers and sensors/actuators over TCP/IP protocol. Each robot has a driver and is connected to a specified socket, which can read/write to all of the sensors and actuators on the robot. Users connect to the sockets through a client/robot interface and can send commands or receive information from each robot. Each robot can communicate with each other or a single common client and update their information continuously to all clients [33].

Our platform, the Tamiya TXT-1, can be modeled in Gazebo using the ClodBuster model. See Fig. 4.5[4] for several views of the ClodBuster model. Each robot is equipped with odometers, a pan-tilt-zoom camera, and a sonar array. Refer to Fig. 4.6[5] for an example of a ClodBuster model equipped with a pan-tilt-zoom camera. Position and orientation are estimated using the odometers. The camera is used for tracking the perimeter. It is pointed down and to the left on each robot to allow the robots to track the perimeter at a small offset. The sonar array is used to avoid collisions. It consists of a ring of 16 sonar sensors around the robot, and only the

[3]Player is freely available under the GNU General Public License from http://playerstage.sourceforge.net.

[4]http://playerstage.sourceforge.net/doc/Gazebo-manual-0.5-html/figures/ClodBuster-6-1.gif.

[5]Courtesy of Daniel Cruz.

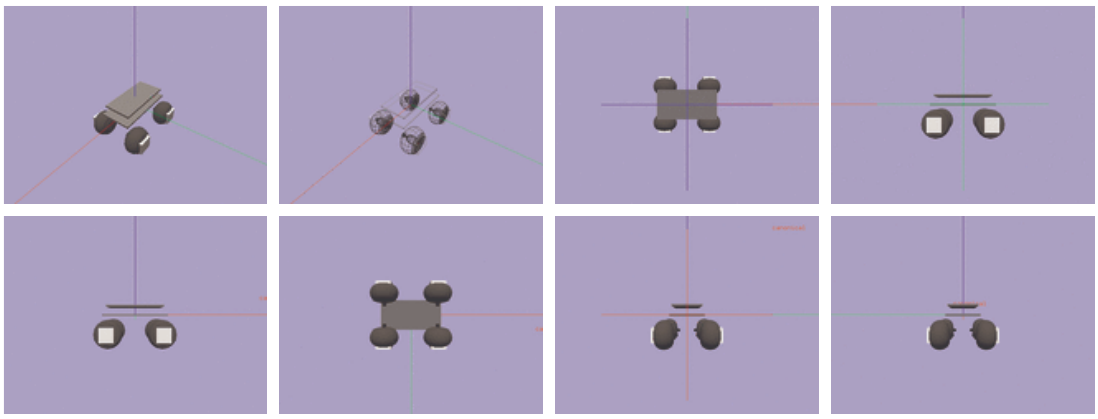

Figure 4.5: Gazebo ClodBuster model.

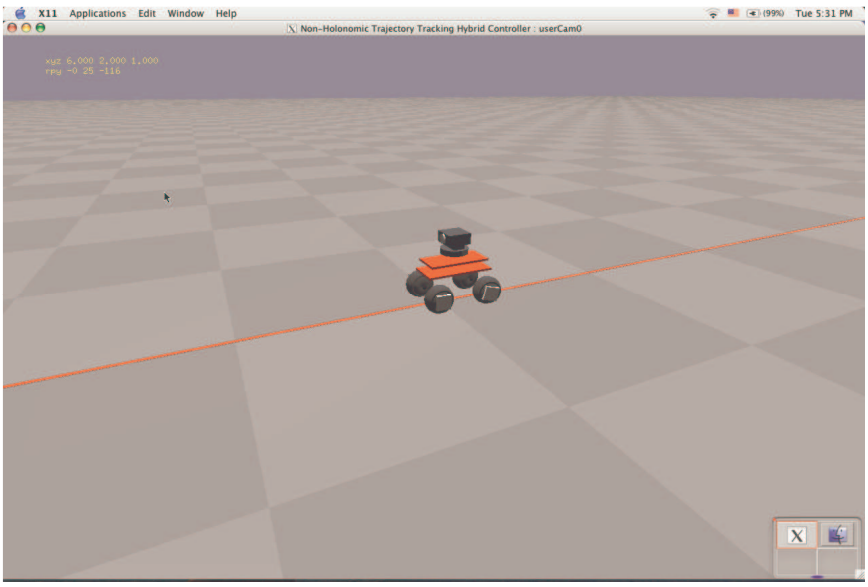

Figure 4.6: ClodBuster model equipped with pan-tilt-zoom camera.

front four are being used.

Shown in Fig. 4.7 is a simulation of a robot that searches for, locates, and tracks a perimeter while avoiding collisions. An environment was created to represent an oil spill in a grassy area in Fig. 4.7. The perimeter and boundary are represented by a black cylinder (oil) and gray walls, respectively. A proportional controller shown below (same as equation (3.13), but shown again for clarity) was developed based off the differences between the areas outside (grass) and inside the perimeter (oil) seen by the camera.

$$\omega_i = k_p \ (\gamma_{out} - \gamma_{in}), \tag{4.1}$$

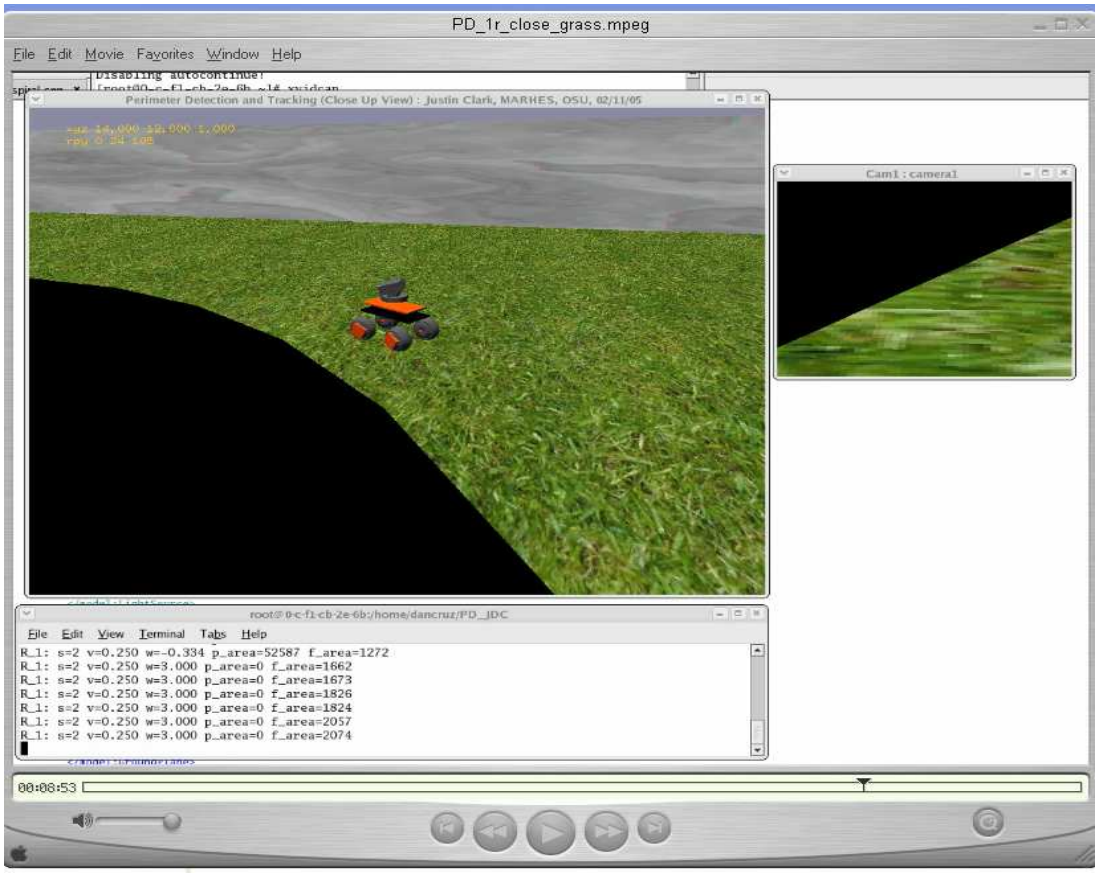

Figure 4.7: Gazebo simulation showing camera view on the right.

where $k_p > 0$, and γ_{out} and γ_{in} are the areas outside and inside the perimeter seen by the blobfinder, respectively. See Fig. 4.8 for a view of what the camera sees in Gazebo. This controller is described further in Chapter 5.

Another simulation is shown in Fig. 4.9 in which three robots search for, locate, and track a perimeter while avoiding collisions. Again, an environment was created to represent an oil spill in a grassy area in Fig. 4.9. The perimeter and boundary are represented by a black cylinder (oil) and gray walls, respectively. The wall textures and the grass were removed to speed up the simulation because during simulations with multiple robots and textures, the simulations ran extremely slow.

The main advantage of Gazebo over Matlab is its ability to debug real world problems without incurring damage to the real robots. For example, when a robot hits the boundary in Matlab, it passes through the boundary, but in Gazebo, the

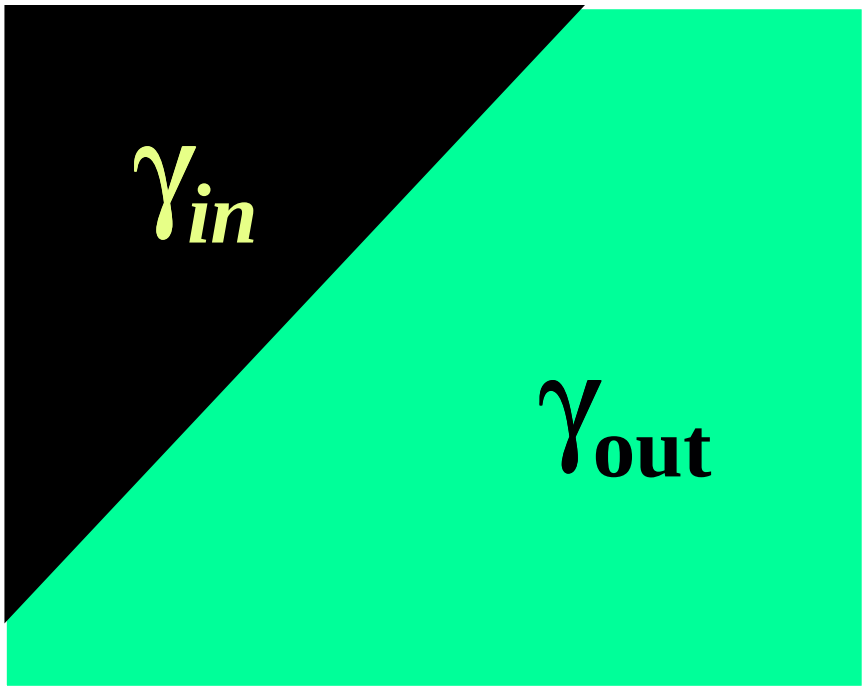

Figure 4.8: Gazebo camera view (the perimeter is black, while the grass is green).

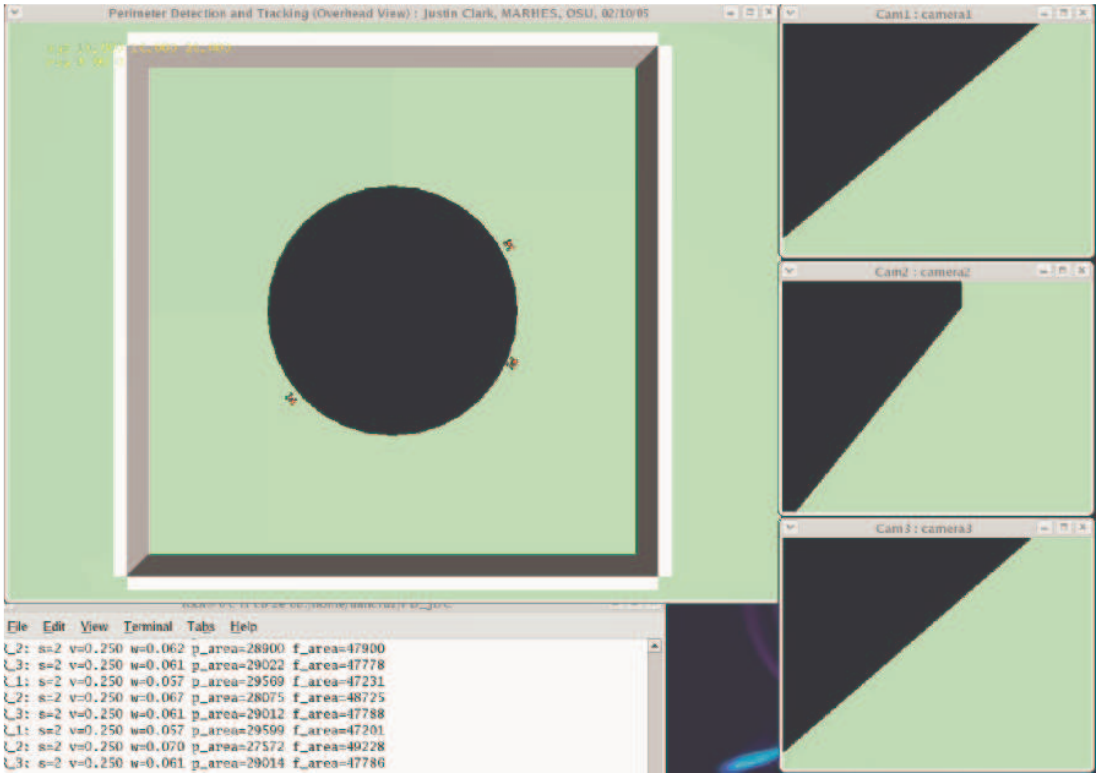

Figure 4.9: Gazebo simulation showing three robots (overhead view) with camera views on the right.

robot climbs the wall and flips over. See Fig. 4.10[6] for an example of a robot hitting a wall and flipping over. Additionally, if a robot is driven too fast in Gazebo, it pops

Figure 4.10: Gazebo simulation showing a robot hitting a wall and flipping over.

a wheelie. These examples are situations that are difficult to simulate in Matlab.

For the tracking controller, in Matlab a binary sensor (bang-bang controller) was used to track the perimeter. In Gazebo, a blobfinder (proportional controller) was used which allowed for much smoother tracking. In fact, a nearly identical proportional controller was implemented for experiments.

The code for the controller logic is shown in Appendix A.2.1. In Gazebo, all environmental features (plane, walls, robots, light source, etc.) must be defined in a world file. World files are written in Extensible Markup Language (XML). See Appendix A.2.2 for the world file used.

As of yet, dynamic perimeters cannot be modeled in Gazebo. When simulating multiple robots with textures, *i.e.*, the grass and the wall, the simulations began slowing down, whereas without textures, the simulations ran at close to real-time. Gazebo is becoming a powerful tool for verifying cooperative control systems as processing

[6]Courtesy of Daniel Cruz.

power increases and more models are defined. Gazebo was an invaluable tool for this application because of the time saved by debugging in simulation and the ability to port the C code almost directly experimentally.

Chapter 5

Experimental Results

5.1 Introduction

This chapter contains a brief description of the experimental testbed (see Fig. 5.1), encountered hardware issues, and detailed experimental results.

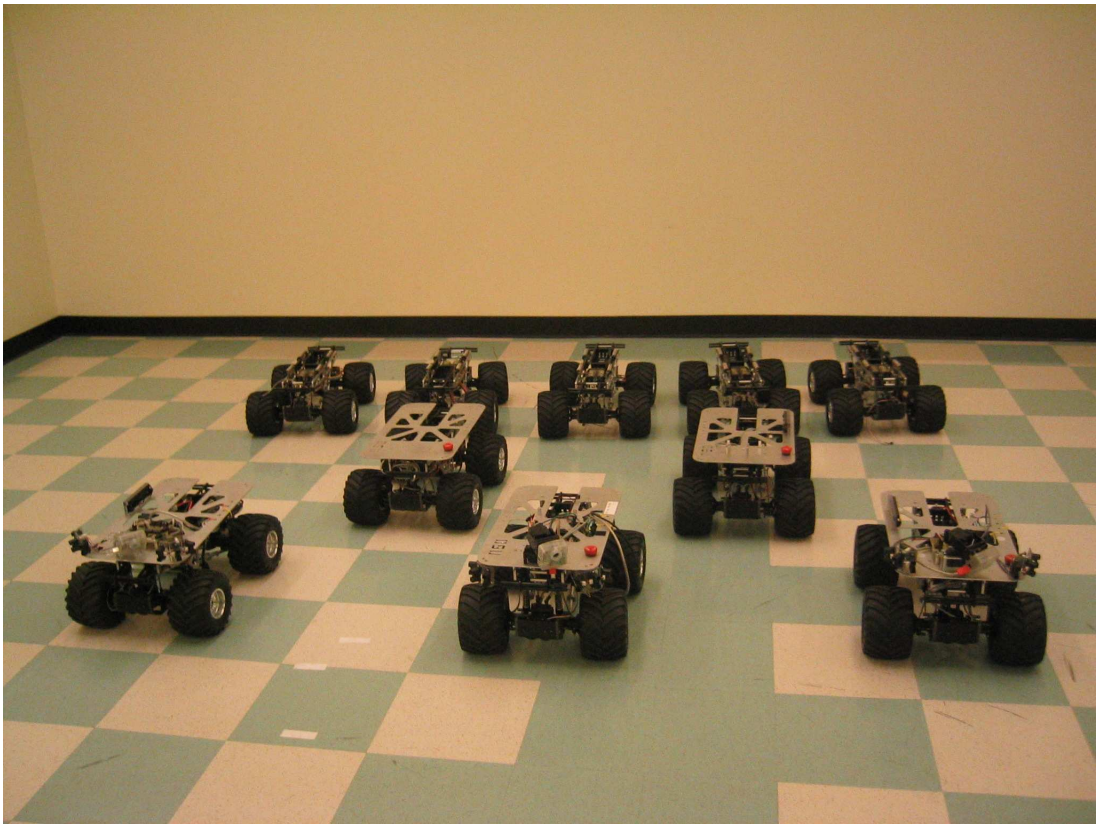

Figure 5.1: MARHES multi-vehicle experimental testbed.

5.2 MARHES Multi-Vehicle Experimental Testbed

The platform hardware and system architecture for the MARHES testbed are briefly discussed below [34]. For more detailed descriptions of the testbed, refer to the MARHES website at http://marhes.okstate.edu.

5.2.1 Platform Hardware

The platform consists of a R/C truck chassis from Tamiya Inc., the TXT-1, odometer wheel sensors, a vision system, an embedded computer (*e.g.*, PC-104 or laptop), a suite of sensors, actuators, and wireless communication capabilities. The lower-level sensing, speed control, and actuator control are all networked using the Controller Area Network (CAN) system [35]. This lower-level network is capable of controlling linear speed up to 2 m/s, angular speed, and managing the network. See Fig. 5.2 for a picture of the platform.

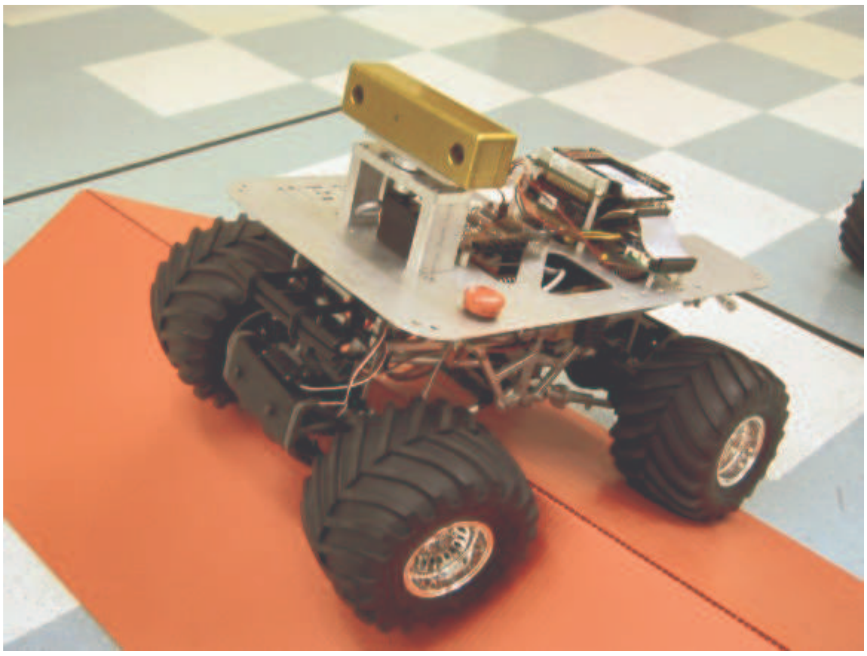

Figure 5.2: Platform equipped with a Bumblebee stereo-vision camera and a PC-104.

Having a velocity-controlled platform is important, since many coordination approaches available in the literature have been developed considering kinematic rather than dynamic mobile robot models [14]. Managing the network consists of detect-

ing modules, controlling the state of a module, and maintaining network integrity by monitoring the network for messages from all modules currently working. The higher-level PC is notified when a module is not working properly. The vehicles are versatile enough to be used indoors, or outdoors in good weather.

5.2.2 System Architecture

The system architecture consists of low-level and high-level layers. The low-level layer is composed of sensors, electronics, and a PCMCIA card to interface the CAN bus with the high-level PC. The high-level layer is made up of PC's and the server. Currently, communication is handled by opening TCP sockets.

The CAN protocol is a message-based bus, therefore bases and workstation addresses do not need to be defined, only messages. These messages are recognized by message identifiers. The messages have to be unique within the whole network and define not only the content, but also its priority. Specifically, the lower-level CAN Bus system allows for sensors and actuators to be added and removed from the system with no reconfiguration required to the higher-level structure *c.f.*, [36]. This type of flexible capability is not common in commercially available mobile robotic platforms. The current configuration of the CAN Bus system is shown in Fig. 5.3.

5.3 Sensing and Communication

In order for the controllers to work properly, several hardware issues with sensing (odometry, vision, and IR) and communication were debugged. These problems and their solutions are described in this section.

5.3.1 Sensing

Odometry

The position and orientation information is estimated from the odometers. Initially, each robot uses the RCC to search for the perimeter, which requires the current

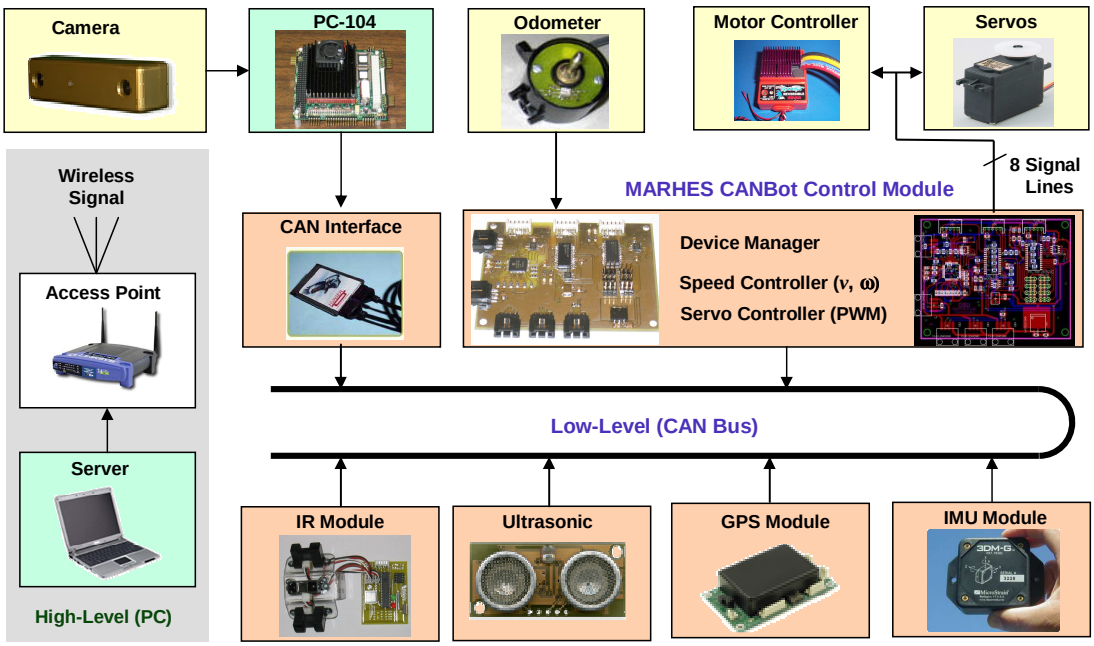

Figure 5.3: CAN bus configuration.

estimate of the orientation angle. In order for the PFC to work, a global coordinate system was set up and each robot's position was estimated so that each robot knows where it is in relation to the goal location.

If the experiments were run for long periods of time, the error from the odometers will lead to large position errors. This was not a problem indoors, as each experiment lasted less than seven minutes. A rough estimate of each robot's initial position was initially taken, which turned out to be effective enough. A detailed description of the odometry is presented in [37].

Vision

An inexpensive Unibrain Fire-i camera was used for vision. A black and white blobfinder algorithm was implemented. For the camera to see black for the table-cloths (perimeter) and white for the floor, the camera was calibrated for the testing area.

As each robot detects the perimeter, it begins using the TC with the following proportional angular velocity controller (same as equation (3.13), but shown again

here for clarity):

$$\omega_i = k_p \ (\gamma_{out} - \gamma_{in}), \tag{5.1}$$

where $k_p > 0$, and γ_{out} and γ_{in} are the areas outside and inside the perimeter seen by the blobfinder, respectively. Counterclockwise tracking is assumed that implies the robot will turn left (right) if the robot is too far outside (inside) the perimeter.

Some outdoor experiments were attempted, but the cameras were ineffective in bright sunlight because the blobfinder algorithm used saw only white. In order to run experiments outdoors, a blobfinder is needed that is immune to bright sunlight, perhaps by looking at the noise (variance) of different textures to separate them into blobs[1].

IR Sensors

Collision avoidance is accomplished through the use of IR sensors. Sometimes collisions occurred during experiments when the IR sensors missed a reading or because their range is small. Other times, collisions occurred because IR sensors were not placed on the rear of each robot, so when a robot backed up, it was "blind" to whatever was behind it.

5.3.2 Communication

Because the experiments were indoors, communication for the experiments worked differently than in Matlab. Specifically, the relay communication scheme was not needed because every robot is within range of the others at all times.

When the experiments begin, all of the robots are pinging each other once per second. The first robot to locate the perimeter sends the goal location when pinged by the others. Once a robot receives the goal location, it stops pinging. The advantage of doing this is robots could be added to the experiment and they should be able to quickly locate the perimeter.

To communicate, an access point was needed, so a base station was setup to

[1]Idea provided by Chris Flesher.

function as the access point. The algorithm is still decentralized as another robot could be deployed to act as the access point.

5.4 Experiments

Experiments were performed with the MARHES multi-vehicle testbed in the Student Union, Room 415. The perimeter was represented by blue tablecloths. See Fig. 5.4 for a view of the testing area with three robots.

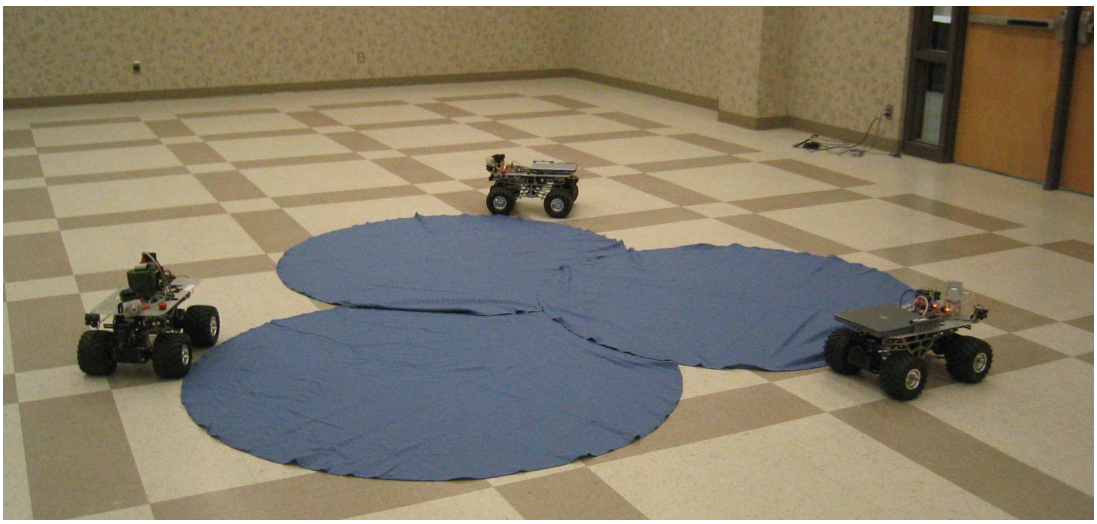

Figure 5.4: Indoor experimental setup for perimeter detection.

Experiments were run in which three robots search for, locate, and track static and dynamic perimeters while avoiding collisions. Refer to Appendix A.3 for the code used for the controller logic.

5.4.1 Static Perimeter

For the static perimeter experiment, all robots were able to successfully reach and track the perimeter. Refer to Fig. 5.5 for a trajectory plot. The perimeter in Fig. 5.5 is not exactly like the perimeter in Fig. 5.4, but the perimeter defined by the robots is fairly accurate and allows the user to infer the location of the perimeter.

R_1 locates the perimeter first and begins tracking. It broadcasts its location to

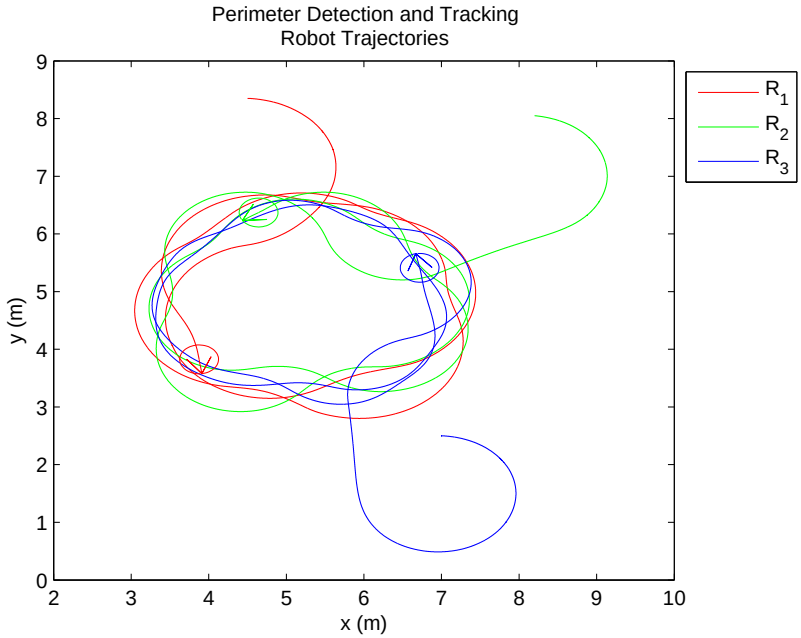

Figure 5.5: Three robots defining a perimeter.

the other robots, who upon receiving the location, enter the PFC. R_2 locates the perimeter next, followed by R_3. Fig. 5.6 shows the state transitions for each robot.

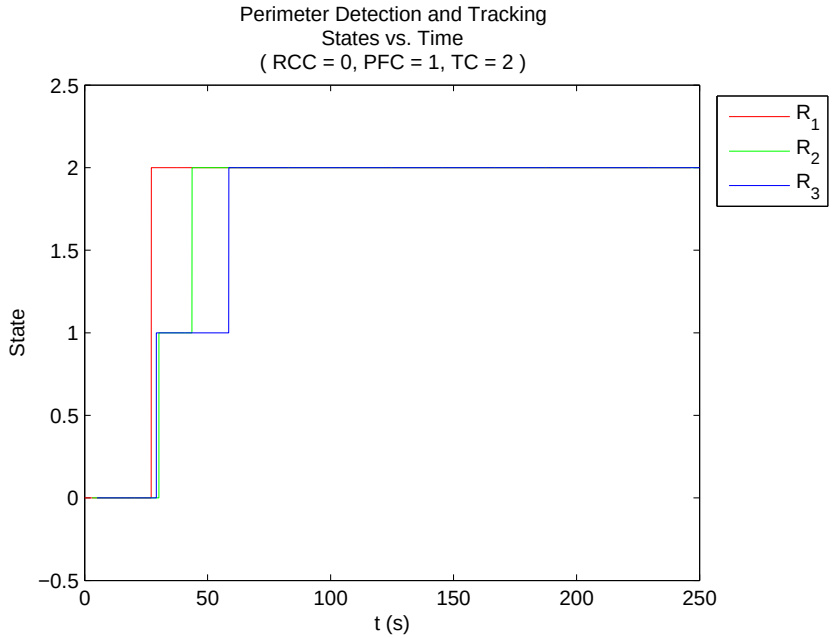

Figure 5.6: Discrete state transition plot.

The robots move with almost constant linear speed in Fig. 5.7. The large spike at approximately 55 s occurs when a robot detects an obstacle and starts to move in

reverse. The sinusoidal angular speed indicates the robots are tracking the perimeter

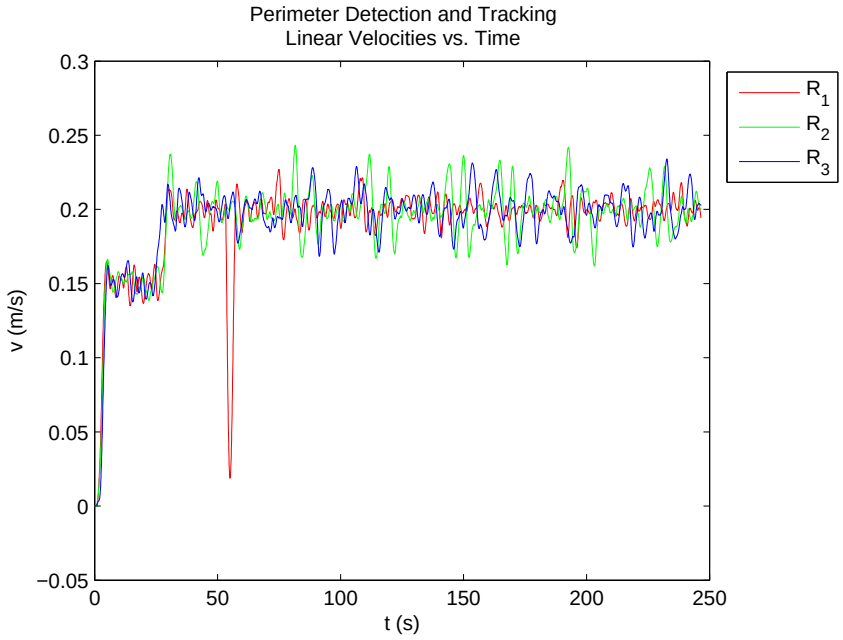

Figure 5.7: Robots tracking with nearly constant speed.

in Fig. 5.8.

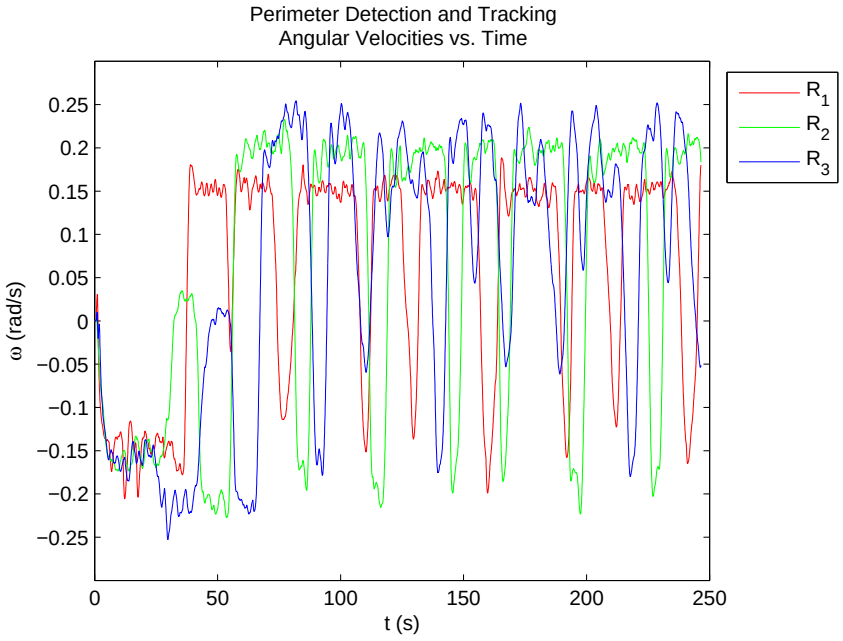

Figure 5.8: Angular velocities become sinusoidal indicating the robots are tracking the perimeter.

To verify the accuracy of the low-level control system, the errors between the actual and desired velocities were plotted. The linear velocity error is relatively small

the majority of the time, except when a robot goes in reverse to avoid an obstacle, *i.e.*, at 55 s in Fig. 5.9. There will almost always be a moderate amount of error in

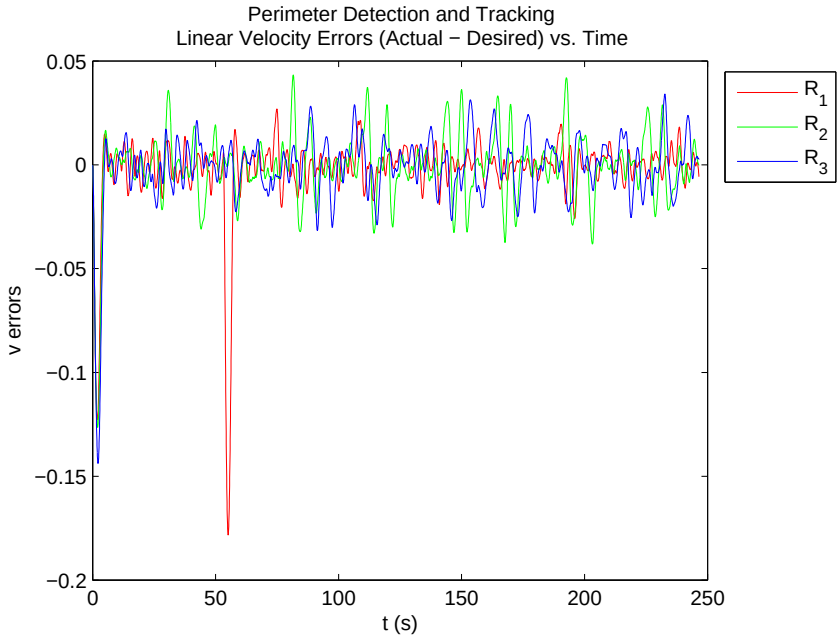

Figure 5.9: Accurate linear velocity provided by low-level control unit.

the angular velocity in Fig. 5.10 because the robots are constantly switching in an effort to track the perimeter.

5.4.2 Dynamic Perimeter

A second experiment was conducted where all of the robots were placed next to the perimeter. The robots start tracking the perimeter. After one lap or so, a section of the perimeter was removed to show that the swarm can still track this dynamic perimeter. This setup is a simplified case of a dynamic perimeter, but indoor experiments limit the ability to test with true dynamic perimeters. Refer to Fig. 5.11 for a trajectory plot. The robots were able to adjust as a section of the perimeter was removed in Fig. 5.11. Also, R_1 ran low on batteries and began to have trouble tracking the perimeter until it completely stopped.

R_2 begins tracking first, so R_1 and R_3 switch to the PFC. R_3 followed by R_1 locate the perimeter and switch to the TC. These transitions can be seen in Fig.

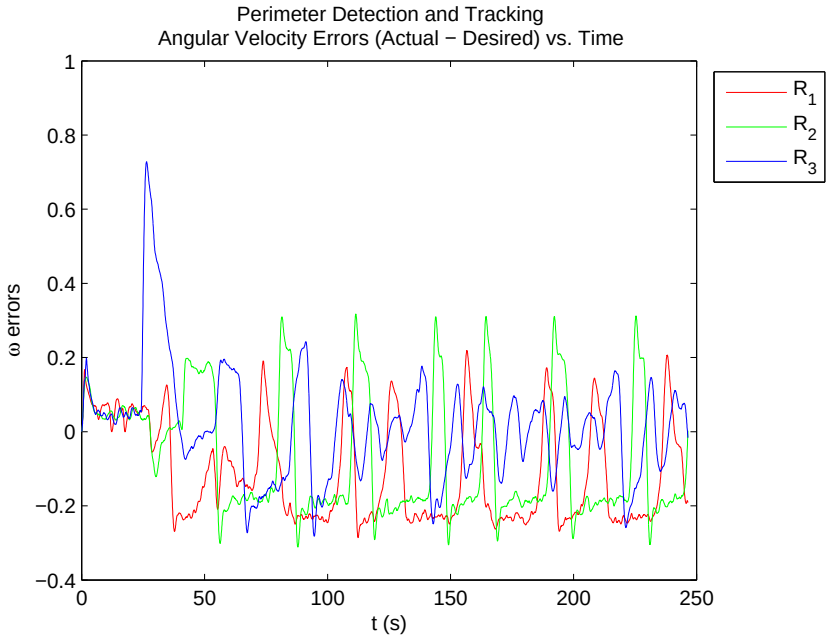

Figure 5.10: Moderate error indicating the robots are constantly turning in an effort to track the perimeter.

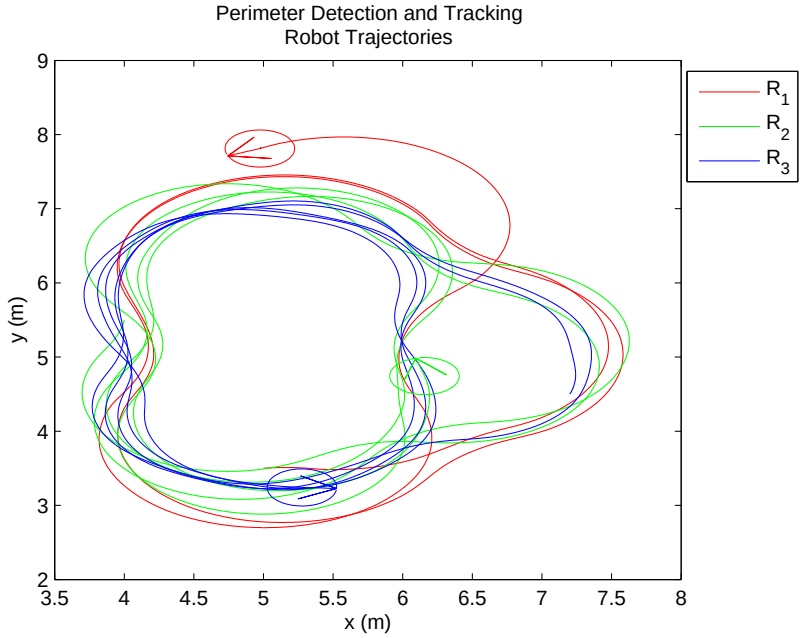

Figure 5.11: Three robots tracking a perimeter where part of the perimeter is removed. R_1 runs low on power and stops.

5.12. The robots track with nearly constant speed, shown in Fig. 5.13. R_1 begins to lose battery power at about 250 s, which causes it to slow down and stop moving for the rest of the experiment in Fig. 5.13. The large spike at approximately 280 s was when R_3 was performing obstacle avoidance. In Fig. 5.14, the angular velocities are

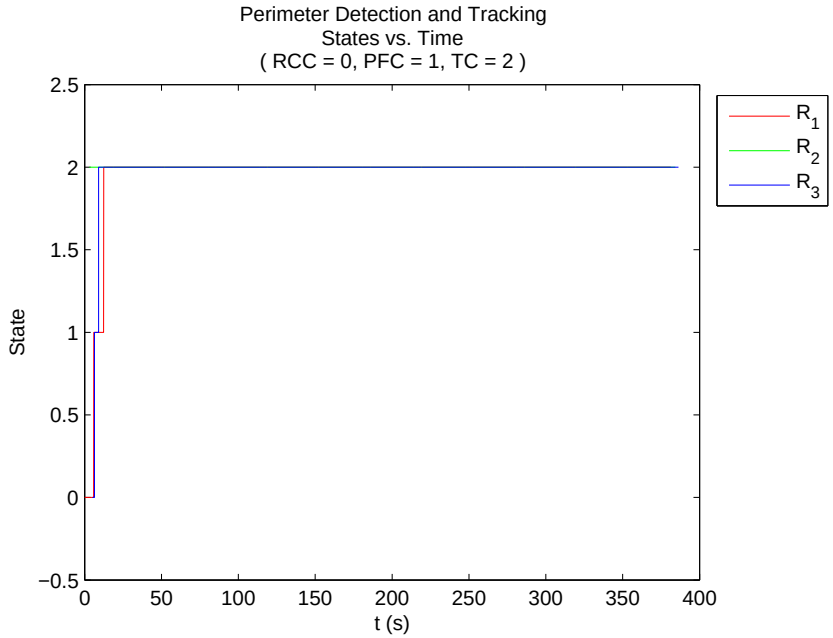

Figure 5.12: Discrete state transition plot with a dynamic perimeter.

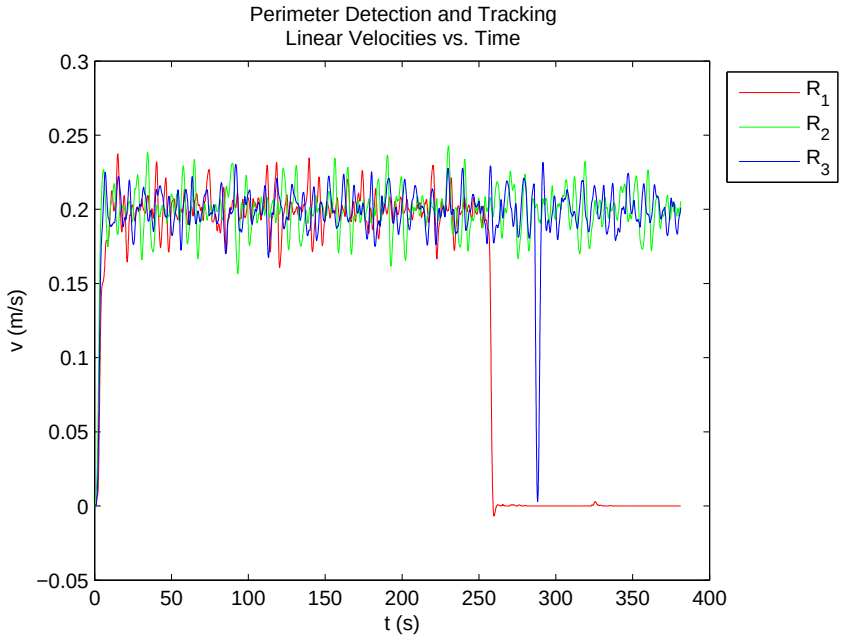

Figure 5.13: Dynamic perimeter tracking with nearly constant speed.

sinusoidal indicating the robots are tracking the perimeter. At approximately 250 s, R_1 is unable to turn, again, because of low power. The errors between the actual and desired velocities are shown in Fig. 5.15 and Fig. 5.16. In Fig. 5.15, the linear velocity errors are very small except at 250 s when R_1 runs out of power and when

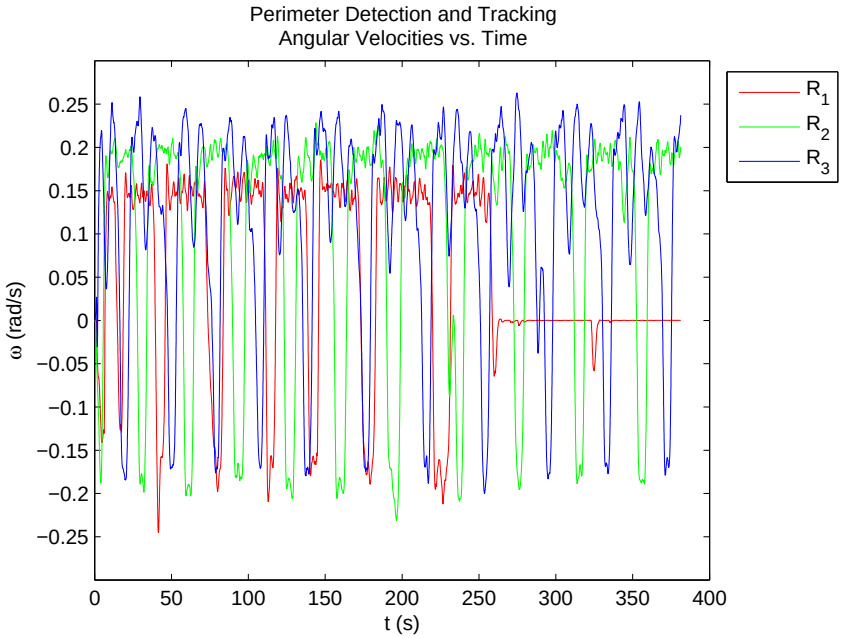

Figure 5.14: Angular velocities become sinusoidal indicating the robots are tracking the dynamic perimeter.

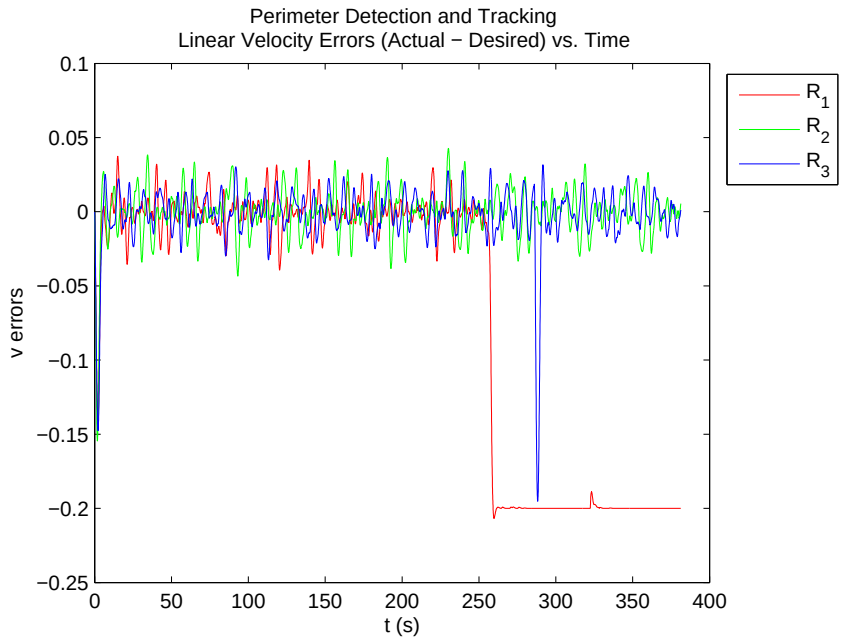

Figure 5.15: Small error except at 250 s when R_1 runs out of power and R_3 is avoiding an obstacle.

R_3 is performing obstacle avoidance. As before, there is a moderate amount of error in the angular velocities in Fig. 5.16 because of the constant switching required to track the perimeter.

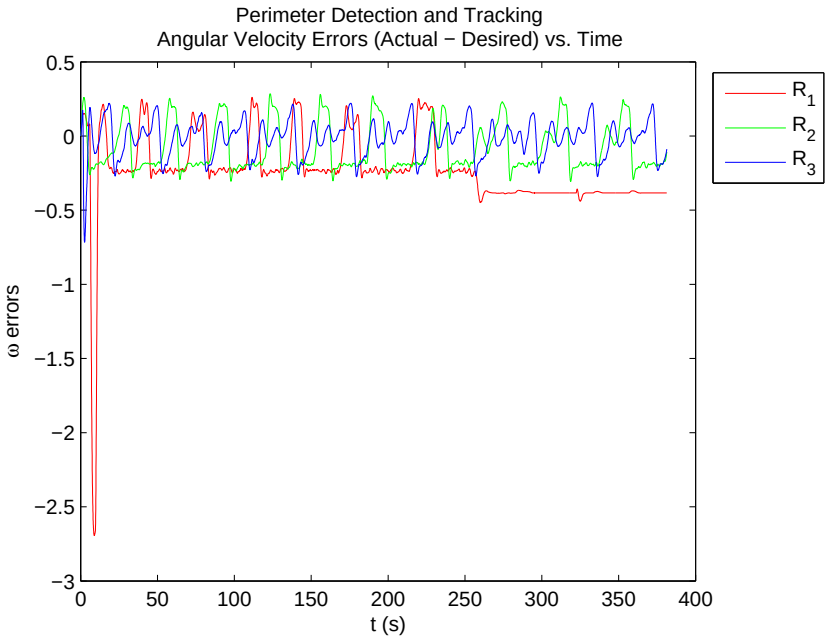

Figure 5.16: Moderate error indicating the robots are constantly turning in an effort to track the dynamic perimeter.

Chapter 6

Conclusions and Future Work

A decentralized, cooperative hybrid system was shown that allows a group of *nonholonomic* robots to search for, detect, and track a dynamic perimeter, while avoiding collisions and reconfiguring *on-the-fly* as additional robots locate the perimeter or the perimeter shape changes. The algorithm has been extensively tested in Matlab, Gazebo, and experimentally.

The two simulators each have benefits and disadvantages. Matlab was useful for individually verifying each controller and eventually, the hybrid system, and for simulating many robots. On the other hand, Player/Gazebo was invaluable for debugging real-world problems. Testing in the Gazebo environment allowed the hybrid system to be implemented experimentally in a short amount of time.

Experimental results were promising, but further testing is required. Specifically, outdoor testing is needed in more realistic scenarios. If the camera is used as the detector, it must be able to handle a variety of lighting conditions.

The most useful feature this algorithm might provide is the initial framework for allowing a dynamic perimeter to be estimated as it evolves. Dynamic perimeter estimation could be important for a variety of applications, such as monitoring a chemical spill. Each robot must be able to estimate at least a portion of the perimeter (through some form of position estimation, *e.g.*, odometry, GPS, IMU, or a combination of the three) for this to work. Once an estimate of the perimeter is known, cooperation can be used to attempt to uniformly surround the perimeter, which should give a better

estimate of the perimeter. If the perimeter is uniformly surrounded, it may be more efficient for the robots to hold their positions instead of moving. They could then move as needed as the perimeter expands/contracts.

Future work may include the following:

1. Experiments with all ten robots in more realistic scenarios, *i.e.*, outdoors, which will require sensor fusion (GPS, IMU, etc.) for localization. See Fig. 6.1 for an example of an outdoor setup for perimeter detection.

Figure 6.1: Outdoor setup for perimeter detection.

2. Methods to estimate the dynamic perimeter as it evolves.

3. Formal analysis of the system behavior, *i.e.*, cycling, stability, etc.

4. Verification and validation using recent tools from hybrid systems.

5. The performance of the team may be evaluated based on different communication schemes.

6. Methods to optimize the search space, which would reduce the required searching time.

7. Further investigation of the proportional angular velocity tracking controller.

8. Extension to tracking a fully three-dimensional field, *i.e.*, underwater plume (pollutants released from a point source).

This algorithm is envisioned to be used in the future at a much higher-level in which the unmanned ground vehicles are part of a heterogeneous team consisting of unmanned aerial and underwater vehicles tailored to the specific application, *i.e.*, planetary exploration, monitoring chemical spills, combat scenarios (see Fig. 6.2).

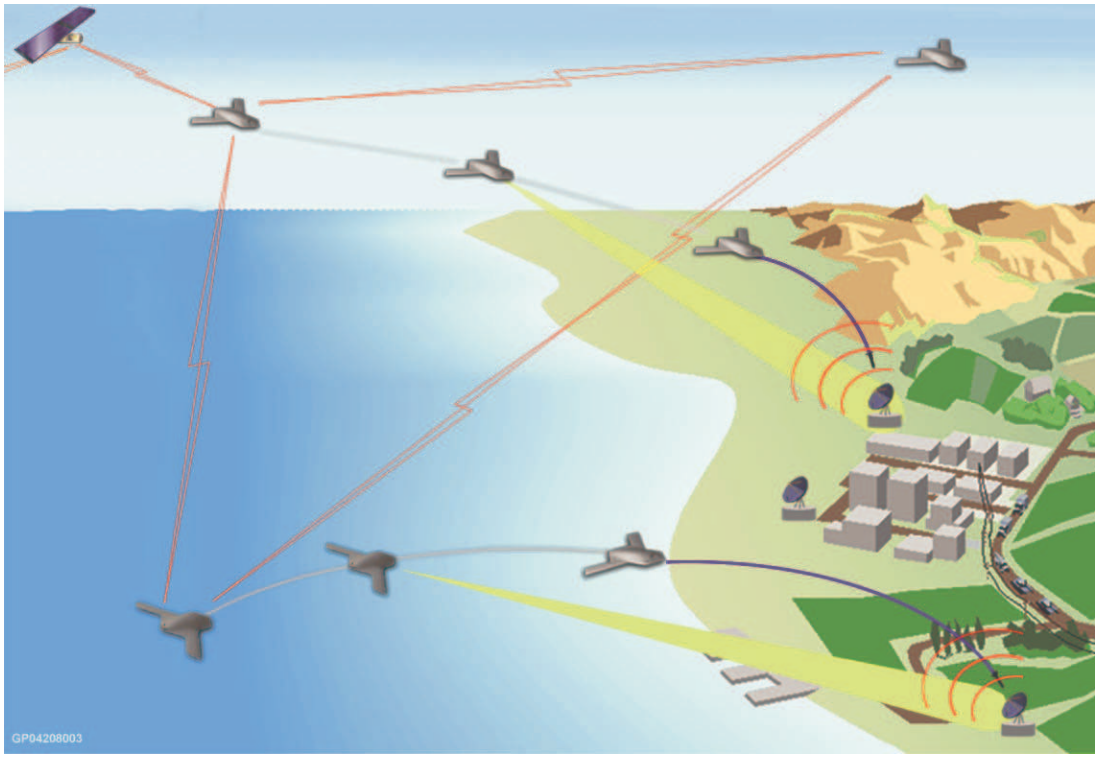

Figure 6.2: Example of a combat scenario.

Bibliography

[1] J. Clark and R. Fierro, "Cooperative hybrid control of robotic sensors for perimeter detection and tracking," in *Proc. American Control Conf.*, Portland, Oregon USA, June 8-10 2005, (To appear).

[2] D. J. Bruemmer, D. D. Dudenhoeffer, M. D. McKay, and M. O. Anderson, "A robotic swarm for spill finding and perimeter formation," in *Spectrum 2002*, Reno, Nevada USA, August 2002.

[3] J. Cortés, S. Martínez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," in *Proc. IEEE Int. Conf. Robot. Automat.*, May 2002, pp. 1327–1332.

[4] H. V. D. Parunak, S. A. Brueckner, and J. Odell, "Swarming pattern detection in sensor and robot networks," in *ANS 10th Int. Conf. on Robotics and Remote Systems for Hazardous Environments*, Gainesville, Florida USA, March 28-31 2004, pp. 444–451.

[5] J. T. Feddema, C. Lewis, and D. A. Schoenwald, "Decentralized control of cooperative robotic vehicles: Theory and application," *IEEE Trans. on Robotics and Automation*, vol. 18, no. 5, pp. 852–864, October 2002.

[6] R. Fierro, P. Song, A. Das, and V. Kumar, "Cooperative control of robot formations," in *Cooperative Control and Optimization*, R. Murphey and P. Pardolos, Eds. Kluwer Academic Press, 2002, vol. 66, ch. 5, pp. 73–93.

[7] J. Tan and N. Xi, "Peer-to-peer model for the area coverage and cooperative control of mobile sensor networks," in *Proc. of the SPIE*, vol. 5403, September 2004, pp. 439–450.

[8] P. Ögren, E. Fiorelli, and N. E. Leonard, "Cooperative control of mobile sensor networks: Adaptive gradient climbing in a distributed environment," *IEEE Trans. on Automatic Control*, vol. 49, no. 8, pp. 1292–1302, August 2004.

[9] Y. U. Cao, A. S. Fukanaga, and A. Kahng, "Cooperative mobile robotics: Antecedents and directions," in *Autonomous Robots*, vol. 4, no. 1. Kluwer Academic Publishers, March 1997, pp. 7–27.

[10] J. T. Feddema, R. D. Robinett, and R. H. Byrne, "An optimization approach to distributed controls of multiple robot vehicles," in *Workshop on Control and Cooperation of Intelligent Miniature Robots, IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Las Vegas, Nevada USA, October 31 2003.

[11] J. A. Marshall, M. E. Broucke, and B. A. Francis, "Formations of vehicles in cyclic pursuit," *IEEE Trans. on Automatic Control*, vol. 49, no. 11, pp. 1963–1974, November 2004.

[12] D. E. Chang, S. C. Shadden, J. E. Marsden, and R. Olfati-Saber, "Collision avoidance for multiple agent systems," in *Proc. IEEE Conf. on Decision and Control*, Maui, Hawaii USA, December 2003, pp. 539–543.

[13] D. E. Chang and J. E. Marsden, "Gyroscopic forces and collision avoidance," in *Proc. of Conf. in Honor of A. J. Krener's 60th Birthday*, 2002.

[14] A. K. Das, R. Fierro, V. Kumar, J. P. Ostrowski, J. Spletzer, and C. J. Taylor, "A vision-based formation control framework," *IEEE Trans. on Robotics and Automation*, vol. 18, no. 5, pp. 813–825, October 2002.

[15] R. Fierro, A. Das, V. Kumar, and J. P. Ostrowski, "Hybrid control of formations of robots," in *Proc. IEEE Int. Conf. Robot. Automat.*, Seoul, Korea, May 2001, pp. 157–162.

[16] P. Ögren and N. E. Leonard, "Obstacle avoidance in formation," in *Proc. IEEE Int. Conf. Robot. Automat.*, Taipei, Taiwan, September 2003, pp. 2492–2497.

[17] A. Jadbabaie, J. Yu, and J. Hauser, "Stabilizing receding horizon control of nonlinear systems: A control lyapunov function approach," in *Proc. American Control Conf.*, vol. 3, San Diego, CA USA, June 2-4 1999, pp. 1535–1539.

[18] R. Bachmayer and N. E. Leonard, "Vehicle networks for gradient descent in a sampled environment," in *Proc. IEEE Conf. on Decision and Control*, Las Vegas, Nevada USA, December 10-13 2002, pp. 112–117.

[19] A. T. Hayes, A. Martinoli, and R. M. Goodman, "Distributed odor source localization," *IEEE Sensors*, vol. 2, no. 3, pp. 260–271, 2002.

[20] D. W. Gage, "Randomized search strategies with imperfect sensors," in *Proceedings of SPIE Mobile Robots VIII*, vol. 2058, Boston, Massachusetts USA, September 1993, pp. 270–279.

[21] D. Marthaler and A. L. Bertozzi, "Collective motion algorithms for determining environmental boundaries," in *Autonomous Robots*, 2002, special issue on Swarming (Submitted).

[22] C. H. Hsieh, Z. Jin, D. Marthaler, B. Q. Nguyen, D. J. Tung, A. L. Bertozzi, and R. M. Murray, "Experimental validation of an algorithm for cooperative boundary tracking," in *Proc. American Control Conf.*, Portland, Oregon USA, June 8-10 2005, (To appear).

[23] L. Chaimowicz, "Dynamic coordination of cooperative robots: A hybrid systems approach," Ph.D. dissertation, Computer Science Department, Federal University of Minas Gerais, Belo Horizonte, Brazil, June 2002.

[24] T. Henzinger, "The theory of hybrid automata," in *Proc. of the 11th Annual Symposium on Logic in Computer Science*, 1996, pp. 278–292.

[25] R. Alur, T. Dang, J. Esposito, R. Fierro, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky, "Hierarchical hybrid modeling of embedded systems," in *Embedded Software*, ser. LNCS 2211, T. Henzinger and C. Kirsch, Eds. Springer, 2001, pp. 14–31.

[26] R. Fierro, A. Das, J. Spletzer, J. Esposito, V. Kumar, J. P. Ostrowski, G. Pappas, C. J. Taylor, Y. Hur, R. Alur, I. Lee, G. Grudic, and J. Southall, "A framework and architecture for multi-robot coordination," *Int. J. Robot. Research*, vol. 21, no. 10-11, pp. 977–995, October-November 2002.

[27] E. Edwan, "Design of a modular autonomous robot vehicle," Master's thesis, Oklahoma State University, Stillwater, Oklahoma USA, August 2003.

[28] J. S. Baras, X. Tan, and P. Hovareshti, "Decentralized control of autonomous vehicles," in *Proc. IEEE Conf. on Decision and Control*, Maui, Hawaii USA, 2003, pp. 1532–1537.

[29] B. Bayazit, "Potential field methods," Washington University in St. Louis, St. Louis, Missouri USA, Tech. Rep., 2003, cs 522A course notes.

[30] J. A. Marshall, M. E. Broucke, and B. A. Francis, "Unicycles in cyclic pursuit," in *Proc. American Control Conf.*, Boston, Massachusetts USA, June 30-July 2 2004, pp. 5344–5349.

[31] S. J. Phillips and P. W. Comus, *A Natural History of the Sonoran Desert*. Arizona-Sonora Desert Museum: University of California Press, December 1 1999.

[32] R. T. Vaughn, B. P. Gerkey, and A. Howard, "On device abstractions for portable, reusable robot code," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Las Vegas, Nevada USA, October 2003, pp. 2121–2427.

[33] B. Gerkey, R. T. Vaughn, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proc. of the 11th Int. Conf. on Advanced Robotics*, Coimbra, Portugal, June 2003, pp. 317–323.

[34] R. Fierro, J. Clark, D. Hougen, and S. Commuri, "A multi-robot testbed for biologically-inspired cooperative control," in *Multi-Robot Systems. From Swarms to Intelligent Automata*, L. E. Parker, F. E. Schneider, and A. C. Schultz, Eds., vol. III. Springer, 2005, pp. 171–182.

[35] K. Etschberger, *Controller Area Network: Basics, Protocols, Chips and Applications.* Weingarten, Germany: IXXAT Press, 2001.

[36] D. Gomez-Ibanez, E. Stump, B. Grocholsky, V. Kumar, and C. J. Taylor, "The robotics bus: A local communications bus for robots," in *Proc. of SPIE*, vol. 5609, 2004.

[37] D. Cruz, "Robot low-level controller," Oklahoma State University, Stillwater, OK USA, Tech. Rep., December 2004, for MAE/ECEN 5483: Digital Data Acquisition and Control, Term Project 2.

Appendix A

Perimeter Detection and Tracking Example Code

A.1 Matlab

The following portion of code shows the structure/logic for implementing the controllers. However, for clarity purposes, the code for individual functions is not shown.

```
for i=1:N,
    [x_lad(i), y_lad(i)]=Sensor(x_r(i), y_r(i), theta(i), D_OS);
        % look-ahead distance
    IN_lad(i)=inpolygon(x_lad(i), y_lad(i), Z_PER(:,1), Z_PER(:,2));
        % look-ahead check
    if IN_os(:,i) == 0 % omni-directional sensor outside
        r_state(i)=0; % robot state
        if GOAL == false % perimeter not found yet => (RCC)
            [v(i), w(i)]=RCC(x_r(i), y_r(i), theta(i), x_os(:,i),...
                y_os(:,i), t, V_MAX, Z_B); % (RCC)
        else % perimeter detected by some robot => (PFC)
            [v(i), w(i)]=RCC(x_r(i), y_r(i), theta(i), x_os(:,i),...
                y_os(:,i), t, V_MAX, Z_B); % (RCC)
            for j=1:size(R_REC,2), % check all detecting robots
                L(j,i)=sqrt((x_r(i)-x_r(R_REC(1,j)))^2+(y_r(i)...
                    -y_r(R_REC(1,j)))^2); % calculate distance from
                        % detecting robots to robot i
                if L(j,i) <= D_COM % inside communication range => (PFC)
                    r_state(i)=1; % robot state
                    [v(i), w(i)]=PFC(x_r(i), y_r(i), theta(i), Z_GOAL,...
                        w_max, t, V_MAX); % (PFC)
                    if size(R_REC,2) == N % all robots receivers
                        break
                    elseif R_REC(1,:) ~= i % if robot not a receiver yet
                        R_REC(R_COUNT)=i; % store receiving robot number
```

```
                    R_COUNT=R_COUNT+1; % increment robot counter
                end
                    break
            end
        end
    end
else % omni-directional sensor inside => (TC)
    r_state(i)=2; % robot state
    if ((size(TC_REC,2) == 0) | (size(TC_REC,2) == 1)) % 0-1 robots
        if ((size(TC_REC,2) == 0) | (TC_REC(1,:) ~= i)) % store if
            % new robot
            TC_REC(TC_COUNT)=i; % store robot
            TC_COUNT=TC_COUNT+1; % increment (TC) counter
        end
        IN_lad(i)=inpolygon(x_lad(i), y_lad(i), Z_PER(:,1),...
            Z_PER(:,2)); % look-ahead check
        [v(i), w(i)]=TC(v_track, IN_lad(i), w_track); % (TC)
            % bang-bang
    else % 2+ robots
        if TC_REC(1,:) ~= i % store if new robot
            TC_REC(TC_COUNT)=i; % store robot
            TC_COUNT=TC_COUNT+1; % increment (TC) counter
        end
        rn=0; % initialize robot number
        L_sm=L_max; % initialize norm of closest robot
        [x_v(i), y_v(i)]=Vertex_Front(0.6, x_r(i), y_r(i),...
            theta(i)); % calculate front point of robot
        for j=1:size(TC_REC,2),
            L_TC(j,i)=sqrt((x_r(i)-x_r(TC_REC(1,j)))^2+(y_r(i)-...
                y_r(TC_REC(1,j)))^2); % calculate norms to
                    % other robots
            if ((L_TC(j,i) ~= 0) & (L_TC(j,i) < L_sm))
                % 0 < norm < stored norm
                L_sm=L_TC(j,i); % store norm of closest robot
                rn=j; % store number of closest robot
            end
            L_TCf(j,i)=sqrt((x_v(i)-x_r(TC_REC(1,j)))^2+(y_v(i)-...
                y_r(TC_REC(1,j)))^2); % calculate norms from front
                    % to other robots
        end
        [v(i)]=TC_Spacing(D_TC, L_sm, L_TCf(:,i), rn, L_max, N,...
            V_MAX); % (TC) proportional
        [w(i)]=TC_Angular(IN_lad(i), w_track); % set angular velocity
    end % End number of robots if
end % End sensor check if
end % End main for loop
```

A.2 Gazebo

This section includes the code for the main function used in Gazebo and an example of the XML code used for the world file.

A.2.1 Main

```
// Run Commands
// compile client, start (1) server, (2) player, (3) playerv (optional)
    // , and (4) client
// make, gazebo P_detection.world, player -g default player.cfg,
    // playerv (optional)
// ./spiral using script => make, ./run
// End Run Commands

// Header Files
    #include <playerclient.h> // Player
    #include <math.h> // math
    #include <stdio.h> // standard input/output
    #include <stdlib.h> // standard library
    #include <unistd.h> // universal standard
    #include "Initial_Conditions.h" // Initial Conditions
    #include "RCC.h" // Random Coverage Controller (RCC)
    #include "PFC.h" // Potential Field Controller (PFC)
    #include "TC.h" // Tracking Controller (TC)
// End Header Files

// Constant Definitions
    const float B_MAX = 20.0; // maximum boundary limit
    const bool GRASS = true; // logical for grass
    const unsigned short int n = 1; // number of robots
    const unsigned short int N = n+1; // number of robots + 1
    const unsigned short int DELAY = 5; // pan-tilt delay
    const float PI = 3.14159265; // pi
// End Constant Definitions

// Main
int main(int argc, char **argv)
{
    // Initialize Variables
    float x_ic[N], y_ic[N], theta_ic[N]; // initial conditions
    float x_i[N], y_i[N], oa[N], v[N], w[N];
    float s_ml[N], s_mr[N], s_ol[N], s_or[N], s_avg[N]; // sonars
    int i, j, s[N]; // robot counters, states
    bool p_detect[N]; // (RCC) => perimeter detected logical
    bool Goal=false; // (PFC) => perimeter logical
    float x_g, y_g; // (PFC) => perimeter location
    int p_area[N], f_area[N]; // (TC)  => areas
```

```
// End Initialize Variables

PlayerClient robot("localhost"); // connect to Player

// Initialize Pointers to Sensors
BlobfinderProxy* camera[N]; // blobfinder
PositionProxy* position[N]; // odometers
PtzProxy* ptz[N]; // pan-tilt-zoom camera
SonarProxy* sonar[N]; // sonar array
// End Initialize Pointers to Sensors

// Initializations
for (i=1; i<N; i++)
{
    camera[i] = new BlobfinderProxy(&robot,i,'r');
        // blobfinder
    position[i] = new PositionProxy(&robot,i,'a');
        // odometers
    ptz[i] = new PtzProxy(&robot,i,'a');
        // pan-tilt-zoom camera
    sonar[i] = new SonarProxy(&robot,i,'a');
        // sonar array

    ptz[i] -> SetCam(0.5, 0.3, 0); // set pan, tilt, and
        // zoom (degrees)
    p_detect[i] = false; // logical for detecting perimeter
    Initial_Conditions(i, x_ic[i], y_ic[i], theta_ic[i]);
        // initial conditions from world file
}
sleep(DELAY); // delay to allow cameras to point down and
    // to the left
// End Initializations

while(1)
{
    robot.Read(); // read all sensors

    for (i=1; i<N; i++)
    {
        // Average Sonar Readings
        s_ol[i] = sonar[i] -> ranges[3]; // outer left
        s_ml[i] = sonar[i] -> ranges[4]; // middle left
        s_mr[i] = sonar[i] -> ranges[5]; // middle right
        s_or[i] = sonar[i] -> ranges[6]; // outer right
        s_avg[i] = (s_ol[i]+s_ml[i]+s_mr[i]+s_or[i])/4.0;
            // average readings
        // End Average Sonar Readings

        // Store x_i, y_i, theta_i
        x_i[i] = position[i] -> xpos+(B_MAX-y_ic[i]); // x position
        y_i[i] = position[i] -> ypos+x_ic[i]; // y position
```

```
oa[i] = position[i] -> theta+(theta_ic[i]*(PI/180));
    // theta
for (j=0; j<10; j++) // orientation angles must be
    // positive for functions to work properly
{
    if (oa[i] < 0) // theta negative
        oa[i] = 2*PI+oa[i]; // make theta >= 0
    if (oa[i] > 2*PI) // theta > 2*pi
        oa[i] = oa[i]-2*PI; // make theta <= 2*pi
}
// End Store x_i, y_i, theta_i

// Check for Perimeter
    // Blobfinder Channels
    // 1 => green
    // 3 => black
    // 6 => yellow
    // End Blobfinder Channels

if (GRASS == false) // for solid with grid.ppm
{
    if ((camera[i] -> blob_count > 0) && (camera[i]...
         -> blobs[0].id == 3)) // perimeter found
    {
        p_area[i] = camera[i] -> blobs[0].area;
            // store perimeter area of blob 0 on
                // channel 3 (black)
        if (p_area[i] == 76788) // 76,788 =>
            // camera seeing black on robot
            p_area[i] = 0;
    }
    else // perimeter not found
        p_area[i] = 0; // initialize perimeter area

    if ((camera[i] -> blob_count > 0) && ...
        (camera[i] -> blobs[0].id == 1)) // floor
            // found
        f_area[i] = camera[i] -> blobs[0].area;
            // store floor area of blob 0 on
                // channel 1 (pale green)
    else // floor not found
        f_area[i] = 0; // initialize floor area
}
else // when using grass.jpg
{
    if ((camera[i] -> blob_count > 0) && ...
        (camera[i] -> blobs[0].id == 3))
            // perimeter found
    {
        p_area[i] = camera[i] -> blobs[0].area;
            // store perimeter area of blob 0 on
```

```
                // channel 3 (black)
        if ((p_area[i] < 50000) || ...
            (p_area[i] == 76788)) // 76,788 =>
                // camera seeing black on robot
            p_area[i] = 0; // initialize
    }
    else // perimeter not found
        p_area[i] = 0; // initialize perimeter area

    if ((camera[i] -> blob_count > 0) && ...
        (camera[i] -> blobs[0].id == 6)) // floor
            // found
        f_area[i] = camera[i] -> blobs[0].area;
            // store floor area of blob 0 on
                // channel 6 (yellow)
    else // floor not found
        f_area[i] = 0; // initialize floor area
}

if (p_area[i] > f_area[i]) // perimeter detected
{
    p_detect[i] = true; // set perimeter detected to
        // true => (TC)
    if (Goal == false) // perimeter not found already
    {
        x_g = position[i] -> xpos+(B_MAX-y_ic[i]);
            // store x position of perimeter location
        y_g = position[i] -> ypos+x_ic[i];
            // store y position of perimeter location
    }
    Goal = true; // perimeter found => set goal for (PFC)
}
// End Check for Perimeter

// Controllers
if (p_detect[i] == false) // perimeter not detected
    // => (RCC) or (PFC)
{
    if (Goal == false) // perimeter not located => (RCC)
    {
        RCC(oa[i], s_avg[i], v[i], w[i]); // (RCC)
        s[i] = 0; // set state
    }
    else // perimeter located => (PFC)
    {
        PFC(i, x_g, y_g, x_i[i], y_i[i], oa[i]...
            , PI, s_avg[i], v[i], w[i]); // (PFC)
        s[i] = 1; // set state
    }
}
else // perimeter detected => (TC)
```

```
            {
                TC(GRASS, p_area[i], f_area[i], s_avg[i]...
                    , v[i], w[i]); // (TC)
                s[i] = 2; // set state
            } // end if (controllers)
            position[i] -> SetSpeed(v[i], w[i]); // set motor speeds
            // End Controllers

            // Print Data
            printf("R_%d: s=%d v=%1.3f w=%1.3f p_area=%d...
                f_area=%d \n", i, s[i], v[i], w[i], p_area[i], f_area[i]);
            // End Print Data
        } // End for loop
        usleep(20000); // 20 ms delay to improve odometer readings
    } // End infinite while loop

    // Shut Down
    for (i=1; i<N; i++)
    {
        position[i] -> SetSpeed(0,0); // turn off robot
    }
    robot.Disconnect(); // disconnect from Player
    // End Shut Down
}
// End Main
```

A.2.2 World File

The following code is the XML code for the world file used for this application.

```
<?xml version="1.0"?>

<gz:world
 xmlns:gz='http://playerstage.sourceforge.net/gazebo/xmlschema/#gz'
 xmlns:model='http://playerstage.sourceforge.net/gazebo/xmlschema/#model'
 xmlns:sensor='http://playerstage.sourceforge.net/gazebo/xmlschema/#sensor'
 xmlns:window='http://playerstage.sourceforge.net/gazebo/xmlschema/#window'
 xmlns:param='http://playerstage.sourceforge.net/gazebo/xmlschema/#params'
 xmlns:ui='http://playerstage.sourceforge.net/gazebo/xmlschema/#params'>

    <param:Global>
        <gravity>0.0 0.0 -9.8</gravity>
    </param:Global>

    <model:ObserverCam>
        <id>userCam0</id>
        <xyz>10 10 18</xyz> <!-- Overhead View -->
        <rpy>0 10 0</rpy>
        <updatePeriod>0.050</updatePeriod>
            <window>
```

```
        <title>SharedWindow1</title>
        <size>800 600</size>
        </window>
</model:ObserverCam>

<model:LightSource>
    <id>light1</id>
        <xyz>0 0 100.0</xyz>
</model:LightSource>

<model:GroundPlane>
    <id>ground1</id> <!-- color = rgb -->
    <color>0 1 0</color> <!-- green = grass -->
    <texture2D>grass.jpg</texture2D>
        <!-- Boundary -->
        <model:SimpleSolid>
            <id>x_axis</id>
            <xyz>10 0 0.125</xyz>
            <shape>box</shape>
            <size>20 0.25 3</size>
            <color>0.7 0.7 0.7</color>
        </model:SimpleSolid>
        <model:SimpleSolid>
            <id>y_axis</id>
            <xyz>0 10 0.125</xyz>
            <shape>box</shape>
            <size>0.25 20 3</size>
            <color>0.7 0.7 0.7</color>
        </model:SimpleSolid>
        <model:SimpleSolid>
            <id>y_axis</id>
            <xyz>10 20 0.125</xyz>
            <shape>box</shape>
            <size>20 0.25 3</size>
            <color>0.7 0.7 0.7</color>
        </model:SimpleSolid>
        <model:SimpleSolid>
            <id>y_axis</id>
            <xyz>20 10 0.125</xyz>
            <shape>box</shape>
            <size>0.25 20 3</size>
            <color>0.7 0.7 0.7</color>
        </model:SimpleSolid>
        <!-- End Boundary -->

        <!-- Perimeter -->
        <model:SimpleSolid>
            <id>Perimeter</id>
            <xyz>10 10 0</xyz>
            <shape>cylinder</shape>
            <size>10 0.020</size> <!-- radius, height -->
```

```
                <color>0 0 0</color> <!-- black = oil -->
            </model:SimpleSolid>
            <!-- End Perimeter -->
    </model:GroundPlane>

    <!-- Robots -->
    <model:ClodBuster>
            <id>robot1</id>
            <xyz>15 15 0</xyz>
            <rpy>0 0 270</rpy>
            <model:Pioneer2Sonars>
                <id>sonar1</id>
                <xyz>2 0 0</xyz>
            </model:Pioneer2Sonars>
        <model:SonyVID30>
                <id>camera1</id>
                <xyz>0 0 0.1</xyz>
                <rpy>0 0 0</rpy>
                <updatePeriod>0.300</updatePeriod>
                <window>
                    <title>Cam1</title>
                    <size>320 200</size>
                </window>
        </model:SonyVID30>
    </model:ClodBuster>
    <!-- End Robots -->
</gz:world>
```

A.3 Experimental

The following C code is the main program used for the experiments. No functions are shown for clarity.

```
// Header Files
    #include <stdio.h>
    #include <pthread.h>
    #include "vision.h"
    #include "blobfinder.h"
    #include "txtcontrol.h"
    #include "pdetect.h"
    #include "comms.h"
// End Header Files

// Constants
    //#define SDL
    const float PI = 3.141535;
    const float SAMPLING_PERIOD = 20e-3;
// End Constants
```

```
// Threads
    pthread_mutex_t outgoing_lock;
    pthread_t tid_controller, tid_video, tid_comms, tid_commands;
// End Threads

// Variables
    //Global
    int go=1;
    FILE    *LogFile;

    // Control & Positioning
    float v=0,w=0;
    float x_pos, y_pos, theta;
    float r_v, r_w;
    // End Control & Positioning

    // Potential Field Controller
    float x_goal,y_goal,x_target,y_target;
    int PFC=0;
    // End PFC

    // Video
    int sw1 = 1;
    int black, white, PDetect=0; // (TC)
    Vision *v_obj = new Vision("FileName");
    // End Video

    int IRR[3], IRL[3]; // obstacle avoidance
// End Variables

// Threads & Functions
    void *video(void *);
    void *controller(void *);
    void *comms(void*);
    void *commands(void *);
    void ResetOdometry(void);
    void CalcOdometry(void);
// End Threads & Functions


// Main
int main(int, char *[])
{
    pthread_mutex_init(&outgoing_lock,NULL);
    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_attr_setscope(&attr,PTHREAD_SCOPE_SYSTEM);

    pthread_create(&tid_controller,&attr,controller,NULL);
    sleep(5); // 5 second safety delay
    pthread_create(&tid_video,&attr,video,NULL);
```

```
    pthread_create(&tid_comms,&attr,comms,NULL);
    pthread_create(&tid_commands,&attr,commands,NULL);

    pthread_join(tid_controller,NULL);
    pthread_join(tid_video,NULL);
    pthread_join(tid_comms,NULL);
    pthread_join(tid_commands,NULL);
    return 0;
}
// End Main

// Video Acquisition Thread
void *video(void *) {
    int s=0; // States

    LogFile = fopen("data.out","w");

    if(!v_obj)
    {
        printf("Video Thread: cannot create a vision object!...
            \n\nquitting ...\n");
        exit(1);
    }
    v_obj->Setup();
    v_obj->MakeCameraManual();

    while(sw1 && go)
    {
        sw1 = v_obj->Process_Events();
        if(sw1 > 0)
        {
            v_obj->CaptureImage();
            GetBlobs(v_obj->pImage, v_obj->blobThresh, black, white);
        }
    #ifdef SDL
        v_obj->Show_Image();
        printf("Video Thread: BlobFinder Threshold  =...
            %d\n", v_obj->blobThresh);
    #endif

        GetOdometry(r_v, r_w, x_pos, y_pos, theta);
        GetIRs(&IRR[0],&IRL[0]);

        if (!PDetect) // check for perimeter
        {
            PDetect = (black>white)?1:0; // set perimeter logical
        }

        if (!PDetect) // (RCC) or (PFC)
        {
            RandomCoverageControl(theta, v, w); // call (RCC) function
```

```
            s = 0; // set state
            if (IRR[2]>80 || IRR[1]>80 || IRL[2]>80 || IRL[1]>80)
                // obstacle detected => back up!
            {
                v = -0.3; // back up
                sleep(3);
            }
            if (PFC)
            {
                PotentialFieldControl(x_pos, y_pos, theta, x_goal,...
                     y_goal, v, w); // call (PFC) function
                s = 1; // set state
            }
        }
        else // (TC)
        {
            if (s!=2)
            {
                x_target = x_pos;
                y_target = y_pos;
            }
            TrackingControl(black, white, v, w); // call (TC) function
            s = 2; // set state
            if (IRR[2]>80 || IRR[1]>80 || IRL[2]>80 || IRL[1]>80)
                // obstacle detected
            {
                v = -0.1; // back up slowly
                w = -w;
                sleep(2);
            }
        }
        PutCommand(v,w);
    }
    v_obj->print();
    v_obj->Shutdown();
    fclose(LogFile);
    return 0;
}
// End Video Acquisition Thread


// Control Thread
void *controller(void *)
{
    ResetOdometry();
    UpCANBUS();
    while(go)
    {
        ReadCANBUS();
    }
    PutCommand(0,0);
```

```
    DownCANBUS();
    return 0;
}
// End Control Thread


// Communications Thread
void *comms(void *) {
    char robot1[5]="pete";
    char robot2[5]="fett";
    char robot3[5]="vaio";
    static int Broadcast=0;

    while(go)
    {
        if (Broadcast){CommsBcast(x_target,y_target);}
        else
        {
            if (PDetect) // if the perimeter is detected
            { // start broadcasting
                CommsStartServer(); // perimeter position
                Broadcast = 1;
            }
            else
            { // if perimeter not detected
                sleep(1); // listen for others
                if (!PFC)
                {
                    PFC = CommsRequest(x_goal,y_goal,robot1);
                    if (PFC){return 0;}
                    PFC = CommsRequest(x_goal,y_goal,robot2);
                    if (PFC){return 0;}
                    PFC = CommsRequest(x_goal,y_goal,robot3);
                }
            }
        }
    }
    return 0;
}
// End Comms Thread

// Commands Thread
void *commands(void *)
{
    char tmp[2];

    while(go)
    {
        scanf("%s",tmp);
        if (tmp[0]=='q')
        {
```

```
            go=0;
            pthread_exit(&tid_controller);
            pthread_exit(&tid_video);
            pthread_exit(&tid_comms);
            pthread_exit(&tid_commands);
        }
    }
    return 0;
}
// End Commands Thread
```

VITA

Justin Delaney Clark

Candidate for the Degree of

Master of Science

Thesis: COOPERATIVE HYBRID CONTROL OF ROBOTIC SENSORS FOR PERIMETER DETECTION AND TRACKING

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Coffeyville, Kansas, on August 3, 1978, the son of Delaney and Loleta Clark.

Education: Received Bachelor of Science degree in Electrical Engineering and a minor in Mathematics from Oklahoma State University, Stillwater, Oklahoma in December 2002. Completed the requirements for the Master of Science degree with a major in Electrical Engineering at Oklahoma State University in May 2005.

Experience: Employed by Oklahoma State University, School of Electrical and Computer Engineering as an undergraduate and graduate teaching assistant, June 2002 to May 2005.

Professional Memberships: Eta Kappa Nu, IEEE Control Systems Society, IEEE Robotics and Automation Society

Name: Justin Clark Date of Degree: May, 2005

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: COOPERATIVE HYBRID CONTROL OF ROBOTIC SENSORS FOR PERIMETER DETECTION AND TRACKING

Pages in Study: 89 Candidate for the Degree of Master of Science

Major Field: Electrical Engineering

Scope and Method of Study: The purpose of this work is to provide experimental results of a real-world multi-vehicle coordination application, perimeter detection and tracking, not to prove optimality, convergence, stability, etc. The tools were provided to experimentally verify the hybrid system (a dynamical system composed of discrete and continuous states). The algorithm has been extensively tested in simulation and experiments.

Findings and Conclusions: A decentralized, cooperative hybrid system was designed and implemented that allows a group of *nonholonomic* robots to successfully search for, detect, and track a dynamic perimeter with limited communication, while avoiding collisions and reconfiguring *on-the-fly*. Furthermore, an assessment of advantages and disadvantages was drawn concerning the simulators (Matlab and Gazebo) used from this testing. Finally, experimental results were promising, but further testing is needed. Future areas of research might include more realistic outdoor tests, methods to estimate the dynamic perimeter as it evolves, and a formal analysis of the hybrid system, *i.e.*, cycling, stability, etc.

Advisor's Approval: _____

Dr. Rafael Fierro