UNIVERSITY OF CALIFORNIA
RIVERSIDE

Experimental Validation of the Distributed Shortest-Distance Rendezvous Algorithm on a Low-Cost Robot Platform

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Electrical Engineering

by

Kush Jay Parikh

June 2013

Thesis Committee:
- Dr. Wei Ren, Chairperson
- Dr. Ertem Tuncel
- Dr. Eamonn Keogh

The Thesis of Kush Jay Parikh is approved:

Committee Chairperson

University of California, Riverside

To my parents

Table of Contents

Page

List of Figures

Chapter 1: Introduction

Multi-agent systems is a very prominent subject of research that is heavily applied in modern fields of study such as transportation, communication, robotics, military technology, energy, space, and many other. Achieving certain complex tasks can be done easily and efficiently with the use of multiple agents. For example, in a domestic domain, multiple vacuum cleaner robots can efficiently clean an entire house by communicating through some form of consensus. This task would be done much faster than a single robot. Consensus is a very strong method to achieve objectives through agreement amongst the agents. An example for an objective would be to rendezvous between all agents at a certain location – an aspect this project follows.

This paper presents the problem of finding the optimal rendezvous point that minimizes the sum of squared distances to it from two or more preset convex regions (circles or ellipses) using a set of mobile robots. Each robot only has information of its own region, its own position, its neighbor(s), and their position(s). We employ the distributed shortest-distance algorithm presented in [1] that minimizes one's distance to its region while moving towards the direction of its neighbor(s). This algorithm is implemented in an infrastructure built using Advanced Robot Interface for Application (ARIA), which provides various tools to communicate and network in C++. This framework is used to program a set of commercially available robots called iRobot Create. Each robot is equipped with a Bluetooth module to send encoder data and receive control commands to and from a central computer. A downward-looking web camera is affixed to the ceiling to emulate proprioceptive and exteroceptive sensor data for each robot. The computer, using the camera, runs ARToolkit to detect the pose of unique markers mounted on each robot. The control data is calculated for each robot using self and neighbor position data and is sent back

to each robot from the central computer. We further improve the algorithm by introducing potential functions in the control law to enable collision avoidance among robots.

1.1 Literature Review

There have been many areas of research involving different algorithms of consensus. Furthermore, there have been many experimental validations proving these theorems to correctly work on multi-robot platforms. Once a theory has been implemented into a real-world physical system, then only can its true effectiveness can be seen and measured. [3] demonstrates the use of distributed containment control with two cases in mind: one with leaders with the same velocity and one with leaders with different velocities. In the results shown, the trajectories of the multi-agent system are contained within a certain region. In [4], experimental validation is performed on several consensus algorithms including rendezvous and axial alignment. The experiments performed prove the theoretical results valid and demonstrate the effectiveness and robustness of the consensus algorithms on a real-world system. The research done in [5] portrays consensus-tracking algorithms on directed network topologies. These topologies can be dynamic throughout the experiment; meaning leaders of the group can change. The algorithm has been tested through experiments confirming its validation. Along with consensus algorithms, there are other experiments that have been demonstrated using physical experiments. To demonstrate dynamic coordination between robots, [6] uses experiments involving passing two balls between two robots. This shows how robust the method of self-organized timing selection can be. The paper proposed a hierarchical architecture for rhythmic coordination and movements to perform juggling-like tasks. Another direction that is taken in distributed control is shown in [7]. The technique, SWARMORPH, is a distributed morphology generation mechanism demonstrated on

autonomous self-assembling mobile robots. Therefore, as noticed, there have been several experiments performed in several directions demonstrating the effectiveness of distributed control.

1.2 Motivation

Imagine a scenario in military warfare with four different platoons from four separate bases away from each other. For convenience, the names of the bases and their platoons are A, B, C, and D. The platoons need to find a rendezvous point that is not far from their respective bases where they can trade intelligence, plan strategies, and trade supplies. Suppose multiple bombs are detonated around this war site and several communication lines are broken between the bases and platoons. So now, A can only talk to B, B can only talk to C, and C and only talk to D as shown in figure 1.1. The meeting point will now be decided through a scheme where the platoons will have to coordinate with their communication-able "neighbors" to find their position data. Now, platoon A will have to move towards platoon B while maintaining an acceptable distance from its base. This point will result in the center of all the bases (this will depend on the shape and size of the bases). The platoons can cooperatively find this location where going back to their home base will not be too much of an issue.

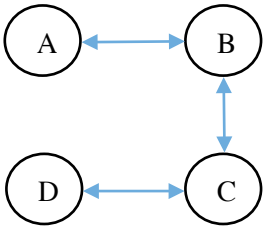

Figure 1.1: Neighbor Topology

1.3 Problem Statement

How do we implement the control laws for the robots to reach an optimal point in the scenario described above? Furthermore, how do we do this with the added constraints that each robot only has knowledge of the position and size of its base and the position of its neighbors?

How can we implement some form of collision avoidance while maintaining the shortest-distance scheme?

1.4 Proposed Solution

A way to reach the optimal point described in the motivation is to find one's minimum distance to its region and scaling it against the direction vector towards its neighbors as a function of time. Given a predefined directed communication topology and predefined closed convex regions, the problem will be use consensus to reach its destination.

To avoid robot collision, the algorithm from [1] is combined with the potential functions in [2]. This prevents the robots from getting too close to each other and maintains an appropriate distance all the way until the final location.

1.5 Outline for Chapters

Chapter 2 covers the theoretical knowledge gained from [1] and its development into this experiment. It goes through all the terms in the control expression and how they are derived.

Chapter 3 shows the incorporation of the collision avoidance scheme to the control law and its development in the experiment.

Chapter 4 covers the hardware and software used by the experiments. It includes everything from the actual parts to the libraries used. It also lists what parts might bring in errors to the system. Additionally, the software architecture of the system is also explained in depth.

Chapter 5 covers the process of implementation, MATLAB simulations, physical and simulated tests, and results achieved.

Chapter 6 goes over several possible improvements to the experiment, potential future work and applications, and a conclusion.

Chapter 2: Shortest Distance Controller

2.1 Model

Before going into the algorithm itself, it is important to note that the model of this system is a single-integrator dynamic. The state space for this system will be

$$\dot{x}(t) = u(t)$$

$$y(t) = 1$$

u is the control input given from the algorithm which is converted to linear and angular velocities. These velocities are then sent over to the robots with special commands. *y* is the final output, but since this model is ideal, there is no sensor data adjustment. In the physical experiment, there will be a *y* associated with the sensor data.

2.2 Principal Algorithm

The principal shortest distance algorithm, developed in [1], is

$$u_i(t) = \alpha \sum_{j \in N_i(t)} sgn(x_j(t) - x_i(t)) + P_{X_i}(x_i(t)) - x_i(t), \tag{2.1}$$

where $x(t)$ denotes the position vector of a robot at time *t*, $N_i(t)$ denotes the neighbor set of agent *i* at time *t*, $P_{X_i}(x_i(t))$ is the projection of the i'th robot's position to its convex set, *sgn* is a signum function, and α is a scaling factor ($\alpha > 0$). The resulting u_i will give the best position in the time step.

The *u* and θ are used to derive values for the linear and angular velocities, which are fed to the iRobot Create wrapper (Marchant). The formulae are as follows,

$$v = u_x \cos(\theta) + u_y \sin(\theta)$$

$$\omega = -u_x \frac{\sin(\theta)}{L} + u_y \frac{\cos(\theta)}{L},$$

where θ is the angle the robot is pointing towards and L is the radius of the robot.

2.2.1 Signum Function

The signum function takes in the distance between a *j'th* neighboring agent and the *i'th* agent. If the distance is negative, the signum function will return a -1. If the distance is positive, it will return a 1. Lastly, if the distance is 0, the function will return a 0. All of the signum functions for each neighbor are then summed up. This process is done for both, the x and y coordinates on the Cartesian plane. The signum function is shown below in figure 2.1.

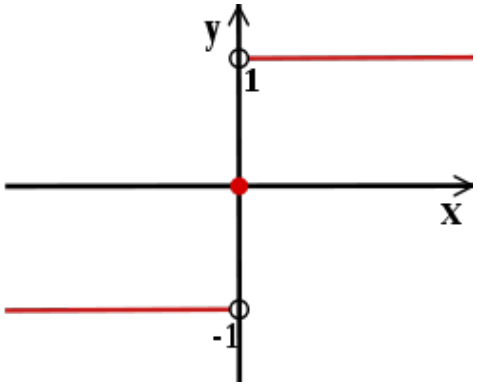

Figure 2.1: Signum Function

2.2.2 Projection

The projection term, $P_{X_i}(x_i)$ denotes the projected point of *i'th* agent's position upon its closed convex region X_i. The projected point is the point on the region that has the minimum distance to its agent. If the agent is inside its region, the projected point will be its own position. This phenomena is shown in figures 2.2 and 2.3.

The projected point is found using constrained optimization. We are trying to solve the following equation:

$$P_X(x) = \arg\min_{\bar{x}\in X} \|x - \bar{x}\|, \tag{2.2}$$

where $\|\bar{x}\|$ is the standard Euclidean norm. The methods that able to perform constrained optimization are plenty. Possible methods include gradient projection, newton raphson, newton-gauss, lagrangian multipliers, and more. For this project's purpose, lagrangian multipliers are used to solve for a closed form equation for certain convex regions. The closed form equation is calculated analytically and provides the least computationally intensive algorithm to implement in programming. There is, however, a tradeoff to using lagrangian multipliers. A region with a specific shape will have a specific closed form equation to calculate the minimum x and y coordinates.

Circles

In the first implementation of the shortest distance algorithm, circles are used as the regions. The derivation for the projection is as follows.

$$\text{Constraint: } (x-p)^2 + (y-q)^2 = r^2$$

The cost function is determined from the argument in (2.2),

$$\left\| \begin{matrix} n \\ m \end{matrix} - \begin{matrix} x \\ y \end{matrix} \right\| = \sqrt{(n-x)^2 + (m-y)^2}$$

Combining the cost function and constraint yields to a lagrangian function,

$$\therefore \Lambda(x, y, \lambda) = \sqrt{(n-x)^2 + (m-y)^2} + \lambda(x^2 - 2px + p^2 + y^2 - 2qy + q^2 - r^2),$$

where n and m are the agent's coordinates, p and q are the center coordinates of the region, r is the radius of the region, x and y are the coordinates of the projected point on the region, and λ is the lagrangian multiplier.

The partial derivatives of the lagrangian function with respect to the x, y, and λ are taken:

$$\frac{\delta\Lambda}{\delta x} = \frac{x-n}{\sqrt{n^2-2nx+x^2+m^2-2my+y^2}} + \lambda(2x-2p) = 0 \tag{2.3}$$

$$\frac{\delta\Lambda}{\delta y} = \frac{y-m}{\sqrt{n^2-2nx+x^2+m^2-2my+y^2}} + \lambda(2y-2q) = 0 \tag{2.4}$$

$$\frac{\delta\Lambda}{\delta\lambda} = x^2-2px+p^2+y^2-2qy+q^2-r^2 = 0 \tag{2.5}$$

Solving for λ in (2.3),

$$\lambda = \frac{n-x}{2(x-p)\sqrt{n^2-2nx+x^2+m^2-2my+y^2}}$$

Solving for λ in (2.4),

$$\lambda = \frac{m-y}{2(y-q)\sqrt{n^2-2nx+x^2+m^2-2my+y^2}}$$

By setting the previous two equations equal to each other, an expression for y can be formed.

$$\frac{n-x}{x-p} = \frac{m-y}{y-q}$$

$$\therefore\ y = \frac{x(m-p)-pm+qn}{n-p} \tag{2.6}$$

Substituting (2.6) into (2.5),

$$x^2-2px+p^2+\left(\frac{x(m-p)-pm+qn}{n-p}\right)^2-2q\left(\frac{x(m-p)-pm+qn}{n-p}\right)+q^2-r^2=0$$

This equation yields to a polynomial expression,

$$x^2[(n-p)^2+m^2-2mq+q^2]$$
$$+x[-2p(n-p)^2-2pm^2+2mqn+2qmp-2q^2n-2qm(n-p)$$
$$+2q^2(n-p)]$$
$$+[(p^2+q^2-r)(n-p)^2+(pm-qn)^2+(2pmq-2q^2n)(n-p)]=0$$

Using the quadratic formula to find the roots of *x* from the polynomial equation. With this information, the *y* for each root can be calculated from (2.6). By checking the calculated values in the constraint formula, the correct projected coordinates can be determined.

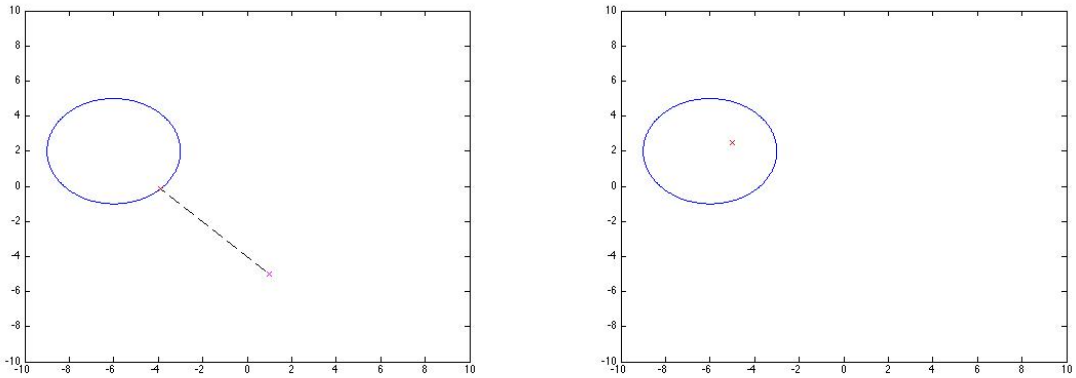

Figure 2.2: Circular Projections (a) Outside region (b) Inside region

Ellipses

The second implementation uses ellipses at regions. According to the constraint, if values *a* and *b* are equal, the regions can be set at circles.

$$\text{Constraint: } \frac{(x-p)^2}{a^2} - \frac{(y-q)^2}{b^2} = 1$$

The cost function remains the same resulting in the following lagrangian function,

$$\therefore \Lambda(x, y, \lambda) = \sqrt{(n-x)^2 + (m-y)^2} + \lambda\left(\frac{(x-p)^2}{a^2} + \frac{(y-q)^2}{b^2} - 1\right),$$

where *a* and *b* are the horizontal and vertical stretch of the region. The rest of the variables are the same.

The partial derivatives of the lagrangian function with respect to *x*, *y*, and λ are taken:

$$\frac{\delta\Lambda}{\delta x} = \frac{x-n}{\sqrt{n^2 - 2nx + x^2 + m^2 - 2my + y^2}} + \lambda\left(\frac{2x}{a^2} - \frac{2p}{a^2}\right) = 0 \qquad (2.7)$$

$$\frac{\delta \Lambda}{\delta y} = \frac{y - m}{\sqrt{n^2 - 2nx + x^2 + m^2 - 2my + y^2}} + \lambda\left(\frac{2y}{b^2} - \frac{2q}{b^2}\right) = 0 \tag{2.8}$$

$$\frac{\delta \Lambda}{\delta \lambda} = \frac{x^2}{a^2} - \frac{2px}{a^2} + \frac{p^2}{a^2} + \frac{y^2}{b^2} - \frac{2qy}{b^2} + \frac{q^2}{b^2} - 1 = 0 \tag{2.9}$$

Solving for λ in (2.7),

$$\lambda = \frac{a^2(n - x)}{2(x - p)\sqrt{n^2 - 2nx + x^2 + m^2 - 2my + y^2}}$$

Solving *for* λ in (2.8),

$$\lambda = \frac{b^2(m - y)}{2(y - q)\sqrt{n^2 - 2nx + x^2 + m^2 - 2my + y^2}}$$

Setting the two previous equation equal to each other will give an expression for y.

$$\frac{a^2(n - x)}{x - p} = \frac{b^2(m - y)}{y - q}$$

$$\therefore y = \frac{b^2mx - b^2pm + a^2qn - a^2xq}{a^2n - a^2x + b^2x - b^2p} \tag{2.10}$$

Substituting (2.10) into (2.9),

$$\frac{x^2}{a^2} - \frac{2px}{a^2} + \frac{p^2}{a^2} + \frac{\left(\frac{b^2mx - b^2pm + a^2qn - a^2xq}{a^2n - a^2x + b^2x - b^2p}\right)^2}{b^2} - \frac{2q\left(\frac{b^2mx - b^2pm + a^2qn - a^2xq}{a^2n - a^2x + b^2x - b^2p}\right)}{b^2}$$

$$+ \frac{q^2}{b^2} - 1 = 0$$

This equation yields to a polynomial expression,

$$x^4b^2l^2 + x^3b^2(2lk - 2pl^2)$$

$$+ x^3[b^2(k^2 - 4plk + p^2l^2) + a^2h^2 - 2qa^2hl + l^2(a^2q^2 - a^2b^2)]$$

$$+ x[b^2(2p^2lk - 2pk^2) + 2a^2hi - 2qa^2(hk + il) + 2lk(a^2q^2 - a^2b^2)]$$

$$+ [b^2p^2k^2 + a^2i^2 - 2qa^2ik + k^2(a^2q^2 - a^2b^2)] = 0,$$

where

$$l = b^2 - a^2$$

$$h = b^2 m - a^2 q$$

$$k = a^2 n - b^2 p$$

$$i = a^2 qn - b^2 pm$$

Using a quartic solver, the roots can be extracted, one of which will be the potential *x* value. With the roots of *x*, the respective *y* values can be calculated from (2.10). By checking the values calculated in the constraint, the correct x and *y* values can be determined. These coordinates will be the projected point on the ellipse.

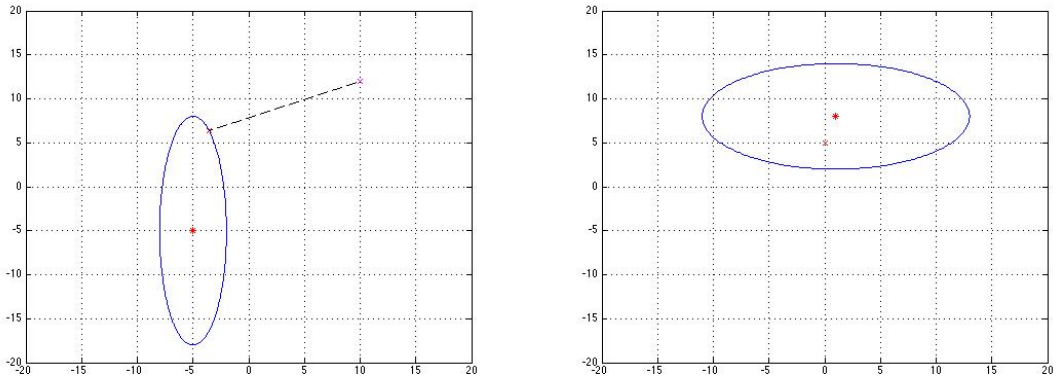

Figure 2.3: Elliptical Projections (a) Outside region (b) Inside region

2.3 Slight Change

When implementing the principal algorithm into the physical experiments, the control values are calculated to be large. These values are sent to the robots as very large velocity commands. When the experimented is started, the robots either move out of camera vision, cannot be sensed by the CV algorithm, or collide into each other. The cause of this is enormous

projection values due to the distance unit of millimeters. To prevent this disruption, a scaling parameter β is added to the principal equation resulting in,

$$u_i(t) = \alpha \sum_{j \in N_i(t)} sgn(x_j(t) - x_i(t)) + \beta(P_{X_i}(x_i(t)) - x_i(t)), \tag{2.11}$$

where $0 < \beta < 1$. The proper value of β is tuned through multiple trials for given experiment environment.

Chapter 3: Shortest-Distance Controller with Collision Avoidance

3.1 Reason

When running physical experiments, the ability to place arbitrary regions are difficult. For example, in figure 3.1, the circles represent the regions and the triangles represent their respective agents. Now if this diagram is a simulation of a physical experiment, the triangles will not be able to overlap each other. The paths of these two agents, however, cross in order to reach the optimal point. This will result in a collision rendering the results of this experiment useless. In order for every experiment to be successful, a collision avoidance scheme must be implemented with the shortest-distance algorithm.

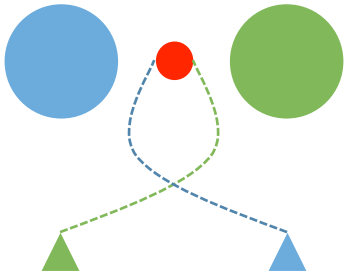

Figure 3.1: Paths of agents demonstrating collision in an experiment run

3.2 Zone Offset Method

To avoid collision at or before the rendezvous point in limited cases, the zone offset method is used. This method gives an offset in position to each agent with respect to a certain formation. The modified equation is as follows,

$$u_i(t) = \alpha \sum_{j \in N_i(t)} sgn(x_j(t) - (x_i\,(t) + h)) + \beta(P_{X_i}(x_i(t) + h) - (x_i(t) + h)), \quad (3.1)$$

where h represents the offset value. h is different for every agent to develop a certain formation. For example, figure 3.2 shows the difference between the paths taken by agents from equation (2.11) versus equation (3.1). Figure 3.2 (a) exhibits how paths will first join and collectively approach the rendezvous point. By implementing a zone offset, as shown in figure 3.2 (b), the paths not only avoid merging, but they maintain a formation throughout the entire run, even after arriving at the point. The virtual center of all agents will be on the final point, thus fulfilling the objective.

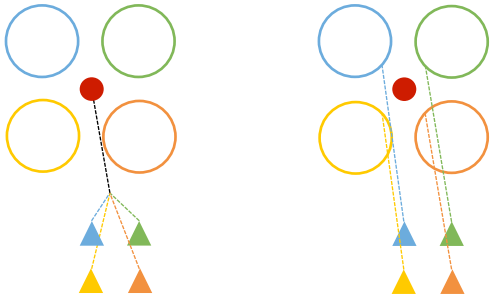

Figure 3.2: (a) Shortest-distance algorithm. (b) Shortest-distance algorithm with zone offset

The zone offset, however, is a very inefficient way to solve the collision avoidance problem since the issue of overlapping paths if regions are changed still remains.

3.3 Possible Solutions

Bumper Collision Re-routing

This scheme uses the bumpers provided on the robots. If the bumper on the robot(s) is activated, the robot will reverse drive for a given distance and stop moving. The stop duration will be the number of the robot in the experiment (eg. robot 1 will stop for 1 second). This way, all robots will reach the rendezvous point even if collision(s) occur.

This method fails in this particular physical system due to the break in the continuous algorithm. When the bumpers are activated, the robots cannot receive control commands or send encoder information. They are left with the previous control values thus giving incorrect information in the next time step. Also, when the robots are required to drive in the reverse direction instead of obeying the control law, the controller as well as the neighbors are misinformed about the agent's location. Additionally, the infrastructure is built in such a way that the robots become disabled to new control commands after bumpers are activated. Lastly, if the method results in success, it does not guarantee a perfect or close resulting final average position to the optimal point.

Chivalry

For this solution to work, one major assumption must be made – all agents are able to communicate with each other. This method is similar to the bumper collision re-routing procedure. When two robots are about to collide, the robot with a higher robot number will stop and allow the other to continue. In cases of multiple agents colliding, the most "chivalrous" agent will stop for the longest period.

The problems associated with this technique are similar to the previous one. The stop period overrides the control law, which might break the continuous operation of the algorithm. The assumption of an undirected network topology also takes out a very large aspect of this experiment. Lastly, as mentioned for the previous method, the resulting position average of all agents might not be close enough to the optimal point.

Potential Functions

This technique offers a solution to enforce collision avoidance by using special functions for neighbors and non-neighbors. A network topology between all agents is set before the experiment is initiated. An agent will have complete information about its neighbors and will use

a potential function to keep a certain distance away from them but also stay close enough to achieve rendezvous in a process known as swarming. For non-neighbors, a different potential function will be used. If a non-neighbor enters an agent's field of vision, communication is established. While in this field, the agent tries to keep a certain distance away from the non-neighbor. If the non-neighbor leaves the field of vision, communication is lost.

This method assumes the technology to make and break communication between at any point during the experiment. It also assumes agents to have sensors to identify neighbors from non-neighbors in the field of vision. In this experiment, due to the lack of such technology, position data is given to the potential function for non-neighbors at all times, but is only used if a non-neighbor enters an agent's field of vision. Due to the highest possibility of implementation, this solution is chosen.

3.4 Potential Functions

The derivative of the potential function for a non-neighbor agent is as follows,

$$\frac{\delta V_{ij}^{1}}{\delta x_i} = \begin{cases} 0, & \|x_i - x_j\| > R \\ \dfrac{2\pi(x_i - x_j)\sin[2\pi(\|x_i - x_j\| - d_{ij})]}{\|x_i - x_j\|}, & d_{ij} < \|x_i - x_j\| \le R \\ \dfrac{20(x_i - x_j)}{\|x_i - x_j\|}\dfrac{\|x_i - x_j\| - d_{ij}}{\|x_i - x_j\|}, & \|x_i - x_j\| \le d_{ij} \end{cases},$$

where d_{ij} represents the radius of the robot and R represents the maximum radius of vision. The potential function itself is shown in figure 3.3. The function decreases to the point of d_{ij} and has a slight increase before saturating to a fixed value. The flat section of the graph represents the non-neighbor being outside of the field of vision. As the non-neighbor approaches to the low

point in the function, the agent must start making distance between the two. This will avoid the two colliding.

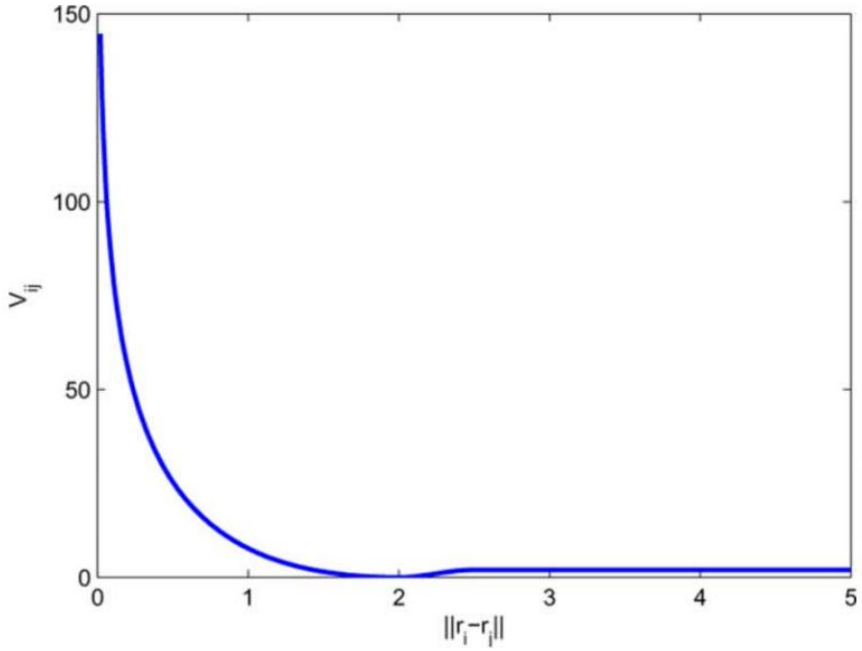

Figure 3.3: Potential function designed for non-neighbors ($d_{ij} = 2, R = 2.5$)

The derivative of the potential function for a neighbor agent is as follows,

$$\frac{\delta V_{ij}^2}{\delta x_i} = \begin{cases} \dfrac{x_i - x_j}{\|x_i - x_j\|} \dfrac{\|x_i - x_j\| - d_{ij}}{\left(\|x_i - x_j\| - R\right)^2} & d_{ij} < \|x_i - x_j\| \\ \dfrac{20(x_i - x_j)}{\|x_i - x_j\|} \dfrac{\|x_i - x_j\| - d_{ij}}{\|x_i - x_j\|}, & \|x_i - x_j\| \leq d_{ij} \end{cases}$$

The potential function plot for this equation is shown in figure 3.4. For the neighboring agents, swarming is used to keep the agents close enough together while maintaining a safe distance to avoid collision. When the distance $\|x_i - x_j\|$ approaches d_{ij}, the function V_{ij}^2 decreases in value. In order to counter this action, the controller must use the derivatives negatively. In the figure, d_{ij} is 2 and R is 2.5. The R in the function for neighbors simply represents how far one is willing to allow the neighbors to go before swarming.

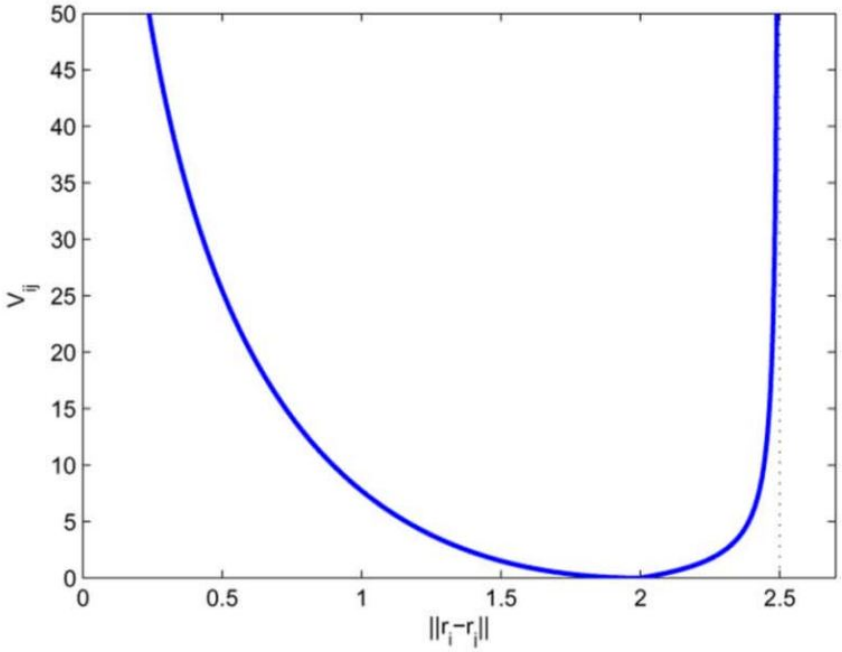

Figure 3.4: Potential function designed for neighbors ($d_{ij} = 2, R = 2.5$)

By combining the algorithm in (2.11) and the potential functions from [2], we can attain an algorithm to achieve proper rendezvous without physical collision. The revised algorithm is as follows,

$$u_i(t) = -\alpha\, sgn\left(\sum_{j \in N_i(t)} \frac{\delta V_{ij}}{\delta x_i}\right) + \beta(P_{X_i}(x_i(t)) - x_i(t)) \tag{3.2}$$

The signum function in this equation returns the vector of magnitude in the quadrant in which the average of the potential function lies. As mentioned earlier, the term including the derivate of the potential functions is used negatively in order to keep away from other agents while swarming.

Chapter 4: Hardware and Software

4.1 iRobot Create

The Create is mobile ground robot created by iRobot for research purposes. It is based on the commercially available Roomba robotic vaccuum cleaner. The reason for using this robot is for ease of programming, experimentation, and availability.

4.1.1 Model

The model used is known as differential drive kinematics. The robots take wheel velocity commands, which are calculated using linear and angular velocities from the controller. Since ARIA does not interface with iRobot Creates, a wrapper, developed by Scott Marchant, have appropriate values sent to the robots.

4.1.2 Specifications

iRobot Create

- Velocity Range: -500 to 500 mm/s
- Encoder Range: -32768 to 32767
- Commands refresh rate: 15ms

4.1.3 Commands

Drive: [145][Right velocity high byte][Right velocity low byte][Left velocity high byte][Left velocity low byte]

Distance: [142][2]

4.1.4 Errors

- Inefficient encoders calculate incorrect distance measurements

- When bumpers are activated, the robots are unable to take in new commands, thus disrupting the experiment

4.2 Bluetooth

The Bluetooth modules are mounted on each robot into a port. These modules are specially designed for the iRobot Create. The Bluetooth dongle is connected to the central computer and communicates with all the robots simultaneously during the experiment.

4.2.1 Specifications

Element Direct Bluetooth Adapter Module (BAM)

- Range: 500ft
- Frequency: 2.4GHz
- Ability to combat external RF interference

4.2.2 Errors

- Delay in transmitting and receiving information every 15ms (due to the Create refresh rate) plus the Bluetooth delay. This can provide inaccurate information and transmit false commands.

- Multiple agents can introduce further delay in communication.

4.3 Overhead Camera

The overhead camera is a wired webcam connected to the central computer. This camera works as an onboard sensor for the robots. The camera uses computer vision algorithms to detect printed markers placed on each robot to identify the pose and robot number on a 2D plane.

4.3.1 Model

The computer vision algorithm from ArToolkit uses the basic pinhole model, where the z-axis is perpendicular to the camera and the observed plane. The resulting image has the x and y axis.

4.3.2 Specifications

Logitech® Webcam C600

- 2.0-megapixel sensor
- 1600x1200 resolution

4.3.3 Errors

- Improper lighting can make the camera completely useless
- Image will be skewed due to improper fixture of the camera
- Camera has a limited field of view. If robots (with patterns) leave this field, their pose information will only come from the encoders
- FPS lag can give improper position coordinates
- Image blur will restrict the CV algorithm from detecting patterns

4.4 Computer

This is the central authority that bring the entire experiment together (robots and camera). The robots simply act as dummies to the computer's commands, which are given after computing position data and various other variables.

4.4.1 Specifications

CPU: Intel® Core™ i7 Processor 3.33GHz

RAM: 12Gb

4.5 Software

The central computer runs on Ubuntu 10.10, Maverick Meerkat. All the software used are provided and licensed.

4.5.1 Programs

Netbeans – Used for camera calibrations

MobileSim – A simulator using the model of the physical robot. This program is linked with ARIA, therefore the simulation is conducted using the C++ code.

MATLAB – Used to test and simulate algorithms before implementing them in C++.

MATLAB Simulink – Toolkit in MATLAB usedin development of block models of the controllers and easier scripting of the algorithms.

Sublime Text 2 – Text editor used for programming.

QT Creator – Used in developing Graphical User Interface and pattern position detector code using the CV tools.

4.5.2 Libraries and Tools

Advanced Robot Interface for Applications (ARIA) – Libraries used in development of the entire project. Provides functions and tools to create architecture, network, and controllers for the experiments.

ArToolkit – Computer Vision tools used in detecting patterns through camera data and extracting pose on an x-y plane.

Eigen – Library used to define Matrix types and perform linear algebraic operations with ease.

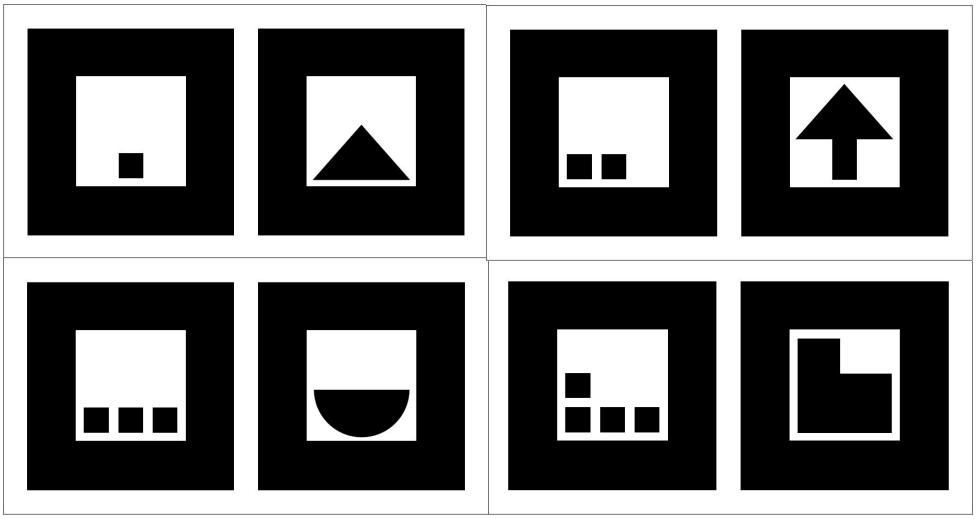

Figure 4.1: Patterns used for detecting robot pose by ArToolkit

4.6 Architecture

The main project's architecture, developed by Scott Marchant, consists of three levels: Driver, Client, and Server. Networking through these three layers are done using TCP. The

library used for this process is known as ArNetworking (a subset of ARIA). Please refer to figures 4.2 and 4.3 for a better visual understanding.

The architecture is set up in such a way where that if each robot had an on-board computers, there would be no need for a central computer. The control law for each robot would be calculated on the on-board computer using data from the prioceptive sensors.

4.6.1 The Driver

The driver is the commander on what formation the robots shall take and what path they should follow. The driver only sends information and never receives any. In this project, since there is no typical formation or a path to follow, the driver portion of the architecture is omitted. The algorithm simply results in a rendezvous.

4.6.2 The Client

This portion is where all of the control calculations take place. The client receives information from the driver (not necessary in this experiment) and from the servers. A client exists for every robot and is able to communicate with servers set through the neighbor topology. In figure 4.2 (a), client 1 can receive information from servers 1, 2, and 3, but not from server 4. Client 1 receives position data from 1, 2, and 3 from the encoders and camera, and computes control commands from the algorithm and sends them to server 1.

4.6.3 The Server

The server is the commander for the robots. The server receives information from the encoders and camera about its robot's position and sends this data to its own client and its neighbors' clients. There exists one server for each robot.

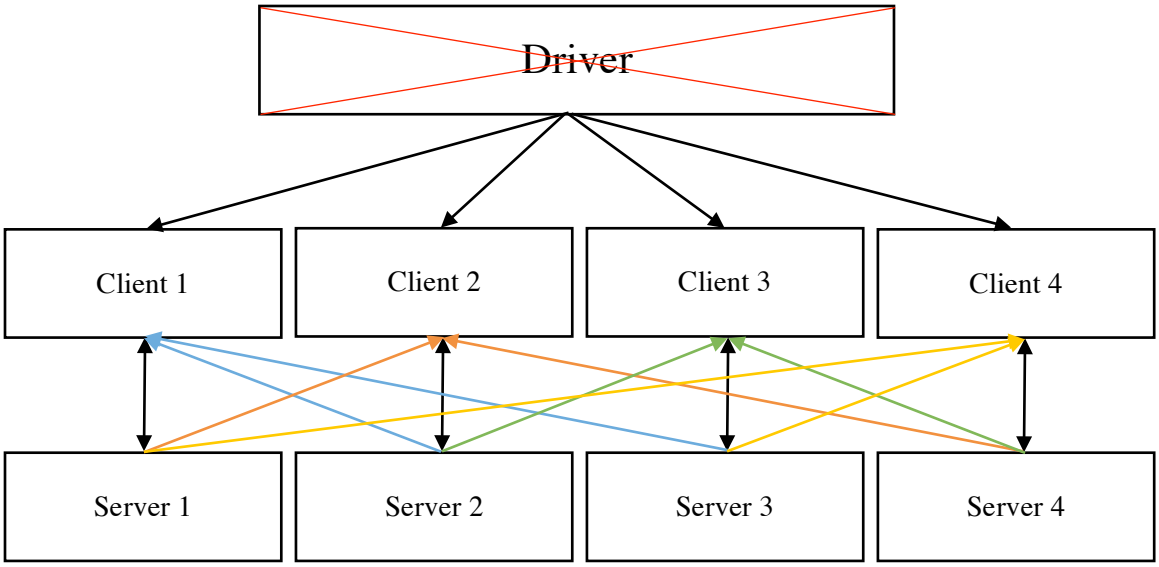

Figure 4.2 (a): The architecture based on the given neighbor topology (driver is omitted in this experiment)

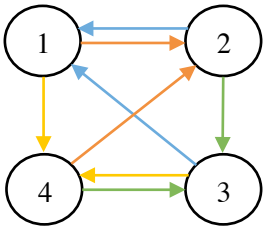

Figure 4.2 (b): The neighbor topology (direction of arrow represents the path of information transfer)

4.6.4 The GUI

The graphical user interface (ConsensusGUI), developed by Scott Marchant, is used for presetting various elements. It is used to set up the neighbor topology, decide the number and type of robots used, what controller to use, and what formation and path is needed. The GUI also helps boot up all the driver, clients, and servers, and set up the communication between them.

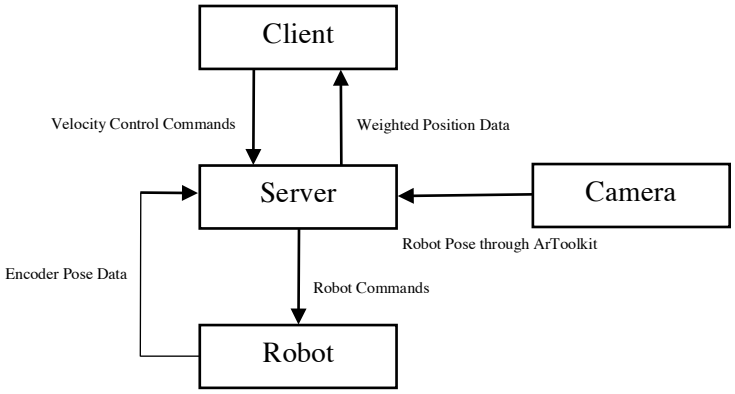

Figure 4.3: Simple block diagram showing information flow through the experiment

Chapter 5: Tests and Results

5.1 Implementation Process

There is a formal process before implementing any type of change into the C++ algorithm. The reason for this is that it might be very difficult to catch an error that can potentially disrupt the experiment in the future.

To avoid this, an algorithm (or change) is first tested using MATLAB and Simulink. The simulations give a good visual on what the controller does and what values are being calculated. Any errors and irregularities can be easily spotted and fixed. After that, the controller or change is developed in C++ as a class (to fit into the pre-built architecture). Before testing the physical experiments, simulations are conducted in MobileSim. These simulations use the C++ code to run and use the physical models of the robots. Even though MobileSim might not use the same scaling factor values in the algorithms, it still shows whether the controller works or fails. Before entering physical experiments, the camera is calibrated to avoid any erroneous pose readings. Once all preparation is complete, the physical experiments with the iRobot Create robots are conducted.

5.2 Shortest-Distance Controller

The algorithm in (2.1) is first simulated using MATLAB and Simulink. The projection component needed to be tested thoroughly (shown in figures 2.2 and 2.3) before testing the entire controller. Once the controller was tested and delivered successful results, the scaling parameter α needed to be tuned. It was then discovered that α needed to be tuned according to the size of the x-y plane. If the scale is bigger, the projection values are larger, which overpowers the signum

function term. Figure 5.1 demonstrates a successful MATLAB simulation of the algorithm in (2.1). The phenomena of agents merging and collectively moving towards the optimal point, shown in figure 3.2 and explained in section 3.2, is demonstrated in this simulation.

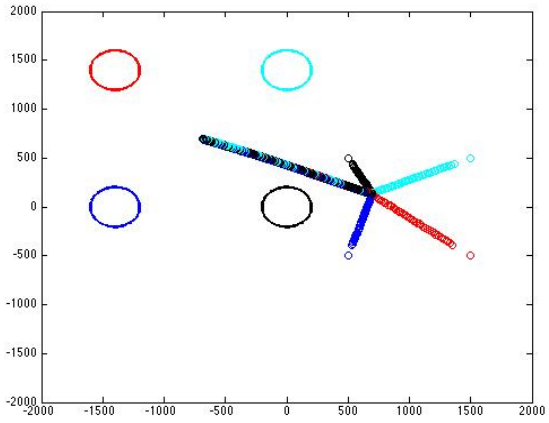

Figure 5.1: MATLAB Simulation using the basic shortest-distance consensus controller ($\alpha = 2000$)

To implement this in C++, the pre-built architecture must be thoroughly understood and all the components must be correctly working. When simulating on MobileSim, it was realized that another scaling constant was required. This resulted into the algorithm in (2.11). After finally physically experimenting the controller, the parameters α and β are tuned. For the experiment environment this project is done in, $\alpha = 20$ and $\beta = 0.1$. The following figure shows a snapshot of the starting and final positions of the robots using the controller with ellipse regions. The final position of the robots might not necessarily average to the optimal point due to camera skew and erroneous information but the result is still very close.

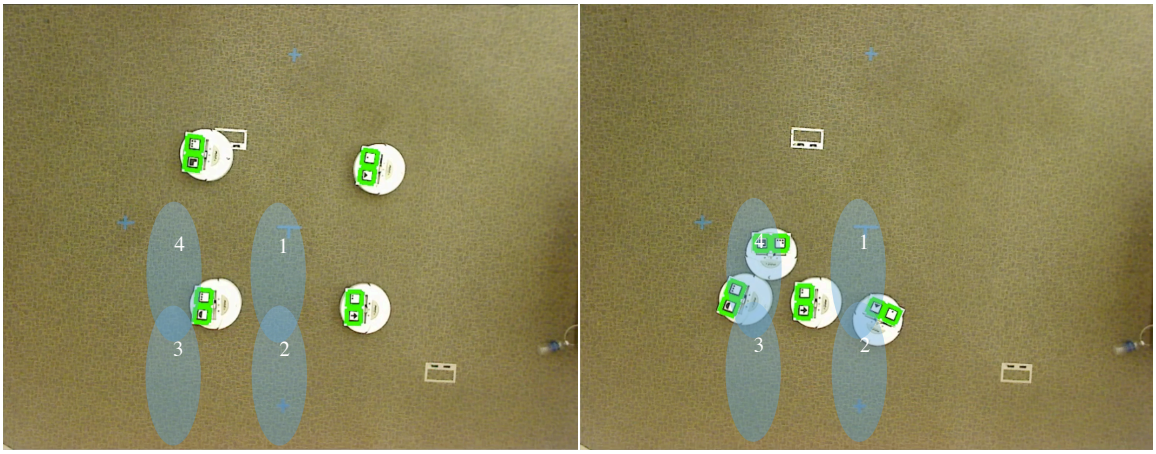

Figure 5.2: Physical robot experiment demonstrating shortest-distance control (a) Start robot positions (b) Final robot positions

5.3 Shortest-Distance Controller with Zone Offset

Using the algorithm in (3.1), multiple MATLAB simulations tuned the offset parameter h to be 200. The result is shown in the figure below.

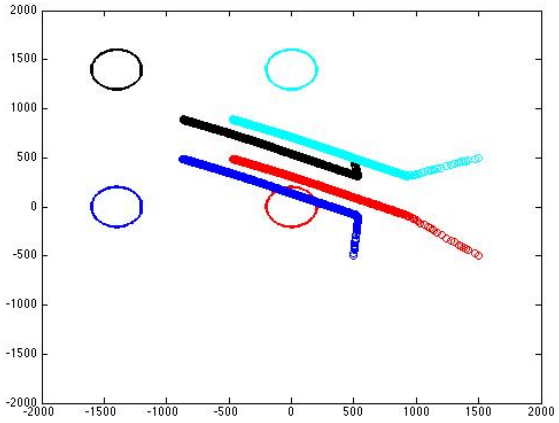

Figure 5.3: MATLAB Simulation using the shortest-distance consensus controller with a zone offset
($\alpha = 2000, \beta = 1, h = 30$)

The implementation done in C++ is shown in the start and final figures below. After multiple physical experiments, the best value of the offset parameter h is 20.

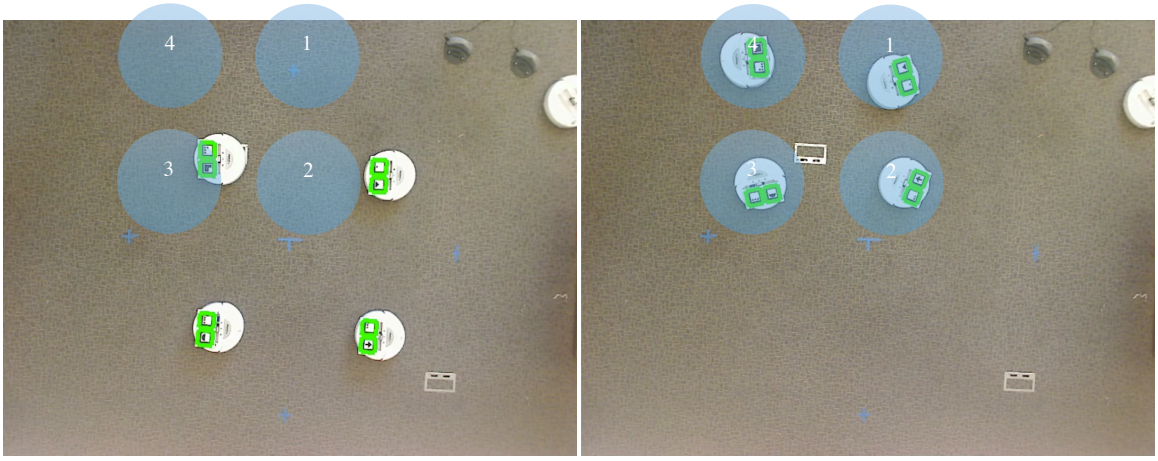

Figure 5.4: Physical robot experiment demonstrating shortest-distance control with zone offset (a) Start robot positions (b) Final robot positions

As mentioned in a previous chapter, the zone offset tends to fail in circumstance that involves agents crossing paths. This problem is recognized in the following figures. If the regions are noticed in figure 5.5 (a), the blue and cyan agents switch sides to be closer to their respective regions. The same is demonstrated in (b) except all the agents intersect due to opposite agent formation and region locations.

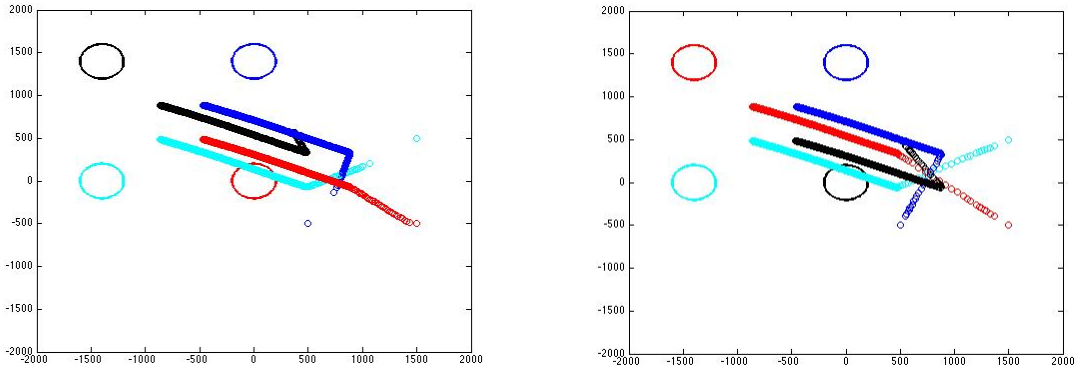

Figure 5.5: MATLAB Simulations demonstrating failure of shortest -distance control with a zone offset

5.4 Shortest-Distance Controller with Collision Avoidance

The final controller stated in (3.2) required several simulations to get proper values for all the parameters. The resulting values for the parameters in the MATLAB simulations are: $\alpha = 300, \beta = 0.1, R = 500, d_{ij} = 250$.

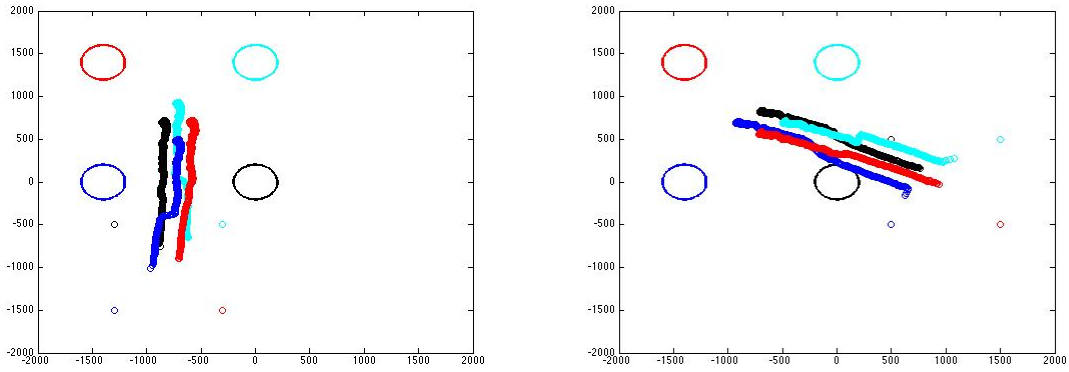

Figure 5.6: MATLAB Simulations demonstrating the shortest-distance control with collision avoidance from different starting locations

As shown in figure 5.5, the agents have a "reason" to intercept paths due to the location of their regions but are not able to because of the potential functions. Throughout the experiment, no two or more agents ever overlap each other.

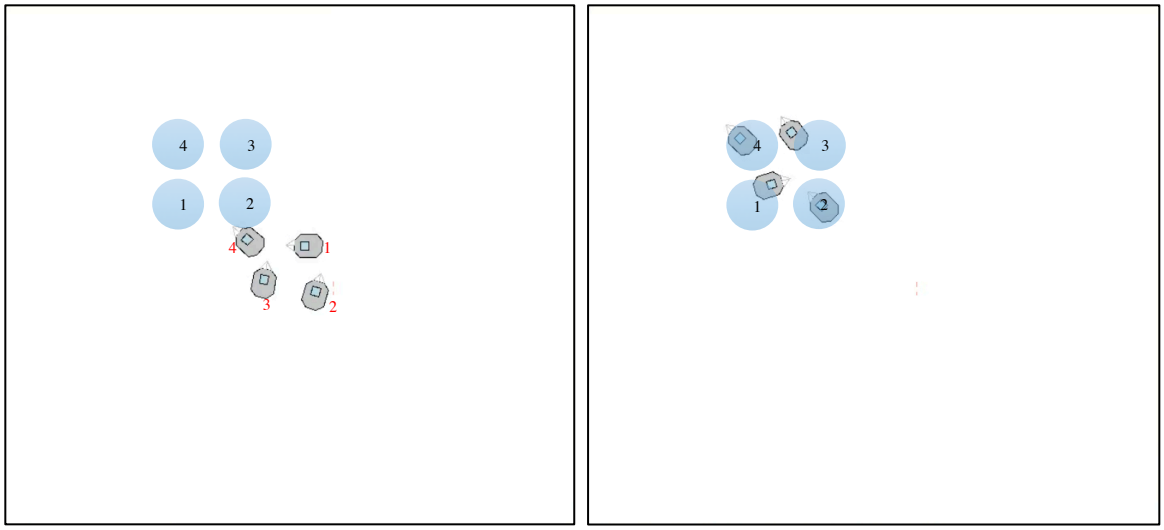

Figure 5.7: MobileSim Simulation (a) Start (b) Final

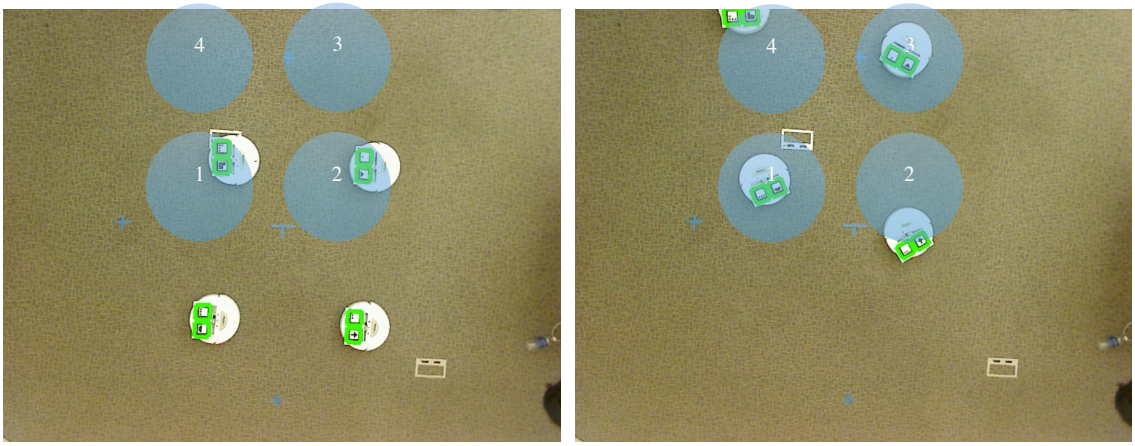

Figure 5.8: Physical robot experiment demonstrating the shortest-distance control with collision avoidance (a) Start robot positions (b) Final robot positions

Figures 5.7 and 5.8 use the following values for the parameters: $\alpha = 100, \beta = 0.1, R = 2000, d_{ij} = 1000$. With such values, the collision avoidance scheme works with any region orientation.

Chapter 6: Conclusion and Future Work

6.1 Conclusion

This paper demonstrates the shortest-distance consensus controller derived in [1] in an experimental fashion. It shows how the controller can be implemented, what changes must be made, and the process of experimentation. It solves the problem of achieving rendezvous in a multi-agent system under directed and undirected network topologies in a physical robot system. The location for rendezvous is determined by the average center of all regions. This point is found by every agent weighting its projection onto its region to the direction of its neighbors. Through physical experiments it is realized that a collision avoidance scheme is necessary for the controller to work in any given case. By combining the algorithm in [1] and potential functions derived in [2], it is possible to develop a scheme to avoid robot collision while maintaining a swarm all the way to and the final rendezvous point. Even through various extraneous errors caused by the low-cost system, the experiments are successful in achieving the desired goal.

6.2 Possible Improvements

There are many possible improvements that can be added to this project. For one, the technology used can be upgraded for a better result in experimentation. By using robots with strong encoders and radar sensors, it is possible to eliminate the use of the camera. Another option would be to use a better camera with a stronger fixture to achieve improved pattern detection thus leading to a more precise pose. In addition to distance measurements, the communication between the central computer and the robots can be improved by using Xbee®

wireless RF modules. This technology will ensure faster communication, which will result in more efficient commanding of the robots.

Alongside technology improvements, the software has areas that might cause problems as well. The architecture, though robust, uses tools not directly supported for the iRobot Create robots. A better option to ARIA would be to use Robot Operating System (ROS). The libraries included can be of great help and are used widely in modern research and experimentation.

6.3 Future Work

There are several directions that this research might progress into. An addition to this project itself can be to implement dynamically changing regions through the continuous operation of the experiment. The collision avoidance scheme can be improved and implemented in autonomous transportation systems, robotic research, and many more.

Applications can range from domestic and educational use to military and space implementations. As demonstrated in the motivation of this paper, there is use for such military technology (using autonomous robots). In space, satellites can easily rendezvous without relying on a high magnitude of communication.

References

[1] P. Lin, W. Ren, "Distributed Shortest Distance Consensus Problem in Multi-agent Systems", in Proceedings of IEEE Conference on Decision and Control, 2012, pp. 4696–4701.

[2] Y. Cao, W. Ren, "Distributed Coordinated Tracking With Reduced Interaction via a Variable Structure Approach", IEEE Transactions on Automatic Control, vol. 57, no. 1, pp. 33–48, 2012.

[3] Y. Cao, D. Stuart, W. Ren, "Distributed Containment Control for Multiple Autonomous Vehicles With Double-Integrator Dynamics: Algorithms and Experiments", IEEE Transactions on Control Systems Technology, vol. 19, no. 4, pp. 929-938, 2011.

[4] W. Ren, H. Chao, W. Bourgeous, N. Sorenson, Y. Chen, "Experimental Validation of Consensus Algorithms for Multivehicle Cooperative Control", IEEE Transactions on Control Systems Technology, vol. 16, no. 4, pp. 745-752, 2008.

[5] W. Ren, "Consensus Tracking Under Directed Interaction Topologies: Algorithms and Experiments", IEEE Transactions on Control Systems Technology, vol. 18, no. 1, 2010.

[6] H. Hirai, F. Miyazaki, "Dynamic Coordination Between Robots: Self-Organized Timing Selection in a Juggling-Like Ball-Passing Task", IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, vol. 36, no. 4, pp. 738-754, 2006.

[7] R. O'Grady, A. L. Christensen, M. Dorigo, "SWARMORPH: Multirobot Morphogensis Using Directional Self-Assembly", IEEE Transactions on Robotics, vol. 25, no. 3, pp. 738-743, 2009.

Special Thanks to:

- Scott Marchant, Nathan Sorenson, Larry Ballard, and Steven Swenson for developing the architecture
- Vaibhav Ghadiok for guidance