Fall 12-2014

Experimental Validation of a New Distributed Cooperative Control Algorithm for Multi-agent Systems with Switching Communication Topologies and Time-Delays

Junzhen Shao
Embry-Riddle Aeronautical University

Embry-Riddle Aeronautical University

Experimental Validation of a New Distributed Cooperative Control Algorithm for Multi-agent Systems with Switching Communication Topologies and Time-Delays

by

Junzhen Shao

A thesis submitted in partial fulfillment for the
degree of Master of Science in Electrical and Computer Engineering

in the

Department of Electrical, Computer, Software, and Systems Engineering, College of Engineering

Thesis Committee: Dr. Tianyu Yang(Advisor), Dr. Ilteris Demirkiran, Dr. Jing Wang

December 2014

Experimental Validation of a New Distributed Cooperative Control Algorithm for Multi-agents Systems with Switching Communication Topologies and Time-Delays

By

Junzhen Shao

This thesis was prepared under the direction of the candidate's thesis committee chairman, Dr. Tianyu Yang, Department of Electrical, Computer and Software Engineering, and has been approved by the members of his thesis committee. It was submitted to the Department of Electrical, Computer and Software Engineering and was accepted in partial fulfillment of the requirements for the degree of Master of Science in Computer and Electrical Engineering.

THESIS COMMITTEE:

[illegible]
Tianyu Yang
Committee Chairman

[illegible]
Jing Wang
Committee Member

[illegible]
Ilteris Demirkiran
Committee Member

[illegible]
Tim Wilson
Department Chair, Department of Electrical, Computer and Software Engineering

[illegible]
Dean, College of Engineering

[illegible] 12-10-14
Associate Vice President for Academics

Abstract

In this thesis, we present practical experimental results to demonstrate a control law for consensus of multiagent systems with switching topologies and time delays. The nonlinear control law utilizes discontinuous cooperative control gains and uses contraction mapping to achieve consensus of multiagent systems. The testing platform we used consists of a number of mobile robots and software simulations both in Matlab and Microsoft Studio C++. We present the effectiveness of the control law design by Aria mobile robots with applications in distributed cooperative formation control. Computer simulations and hardware experiments presented include point consensus control and formation control, both with changing topologies and time-delays. In addition to 2D simulations and experiments, we also developed the 3D model for more practical applications, such as Unmanned Aerial Vehicles (UAVs) and Autonomous Underwater vehicles (AUVs). The research presented was supported by Air Force Research Laboratory / Information Directorate (AFRL/RI)'s Machine Intelligence for Mission Focused Autonomy program. The research presented in this thesis was included in multiple presentations to AFRL program managers, who provided very favorable feedback to our research. Recently, some of our research results was published in the proceedings of 2014 IEEE International Conference on Electro/Information Technology [1], and the paper received Best Paper Award at the conference. In addition, a poster presentation describing our research was made to ERAU College of Engineering's Industry Advisory Board. The process of implementing the research results in AUVs has also been progressing significantly. Upon invitation, some hardware tests were performed as part of NASA NEEMO 19 (Extreme Environment Mission Operations) experiments, and were subsequently reported twice in the headlines of the Science and Education section in the Daytona Beach News Journal. Technical papers describing proposed cooperative control in AUVs were submitted to 2015 ACM and IEEE conferences…

Acknowledgements

This thesis is not only a certification to graduate, it contains most of my study and research life in ERAU. First and foremost I am deeply indebted to Dr. Tianyu Yang for his continuous support both in life and research study. Dr. Tianyu Yang has supported me not only by providing a graduate research assistantship, but also academically and mentally with his patience, motivation and immense knowledge. Thanks to him for the guidance of research and thesis writing for almost one and half years. I could not have a better advisor both in my life and my study.

Also, I would like to express my deepest gratitude to Dr. Jing Wang, for his excellent guidance in research and experiments. His knowledge and ideas always led me in a most efficient way. I always experienced professional mentoring from him for solving practical issues.

Besides, I would like to thank Dr. Ilteris Demirkiran with the great help of building my thesis work professionally, and Dr. Hong Liu with selfless support to my study and life.

Most importantly, I would like to thank my parents for their support and encouragement. Last but not the least, I appreciate all of my friends for a wonderful experience in two years of graduate school…

Contents

List of Figures

Chapter 1

Introduction

Cooperative control[5][10] aims at achieving consensus or agreement dynamics in a multiagent system. It is an area of research lying at the intersection of systems dynamics and graph theory. A prominent application area of cooperative control is autonomous systems, especially for military and government demands. The development of single agent systems is increasingly mature in recent years. For example, unmanned aerial vehicle (UAV) and autonomous underwater vehicle (AUV) play an important role in aerial photography, data collecting, specification monitoring in severe environments or classified operation tasks. In addition cooperative control of multiagent systems can enhance the system performance for these applications due to its' high efficiency and improved stability and reliability. The design of cooperative control is closely related to system dynamics. For linear systems, the dynamics can be simplified to the first-order integrator model or the double integrators model[1][2]. For nonlinear systems, which are more relevant to real world applications, cooperative control is complicated by complex system dynamics, system capability, signal transmitting and time-delays. Large gaps exist between theoretical system design and practical applications[1][3].

The multigent system is a computerized system of multiple interacting intelligent agents within an environment[5][13], and agents work together to accomplish certain tasks. Each agent in the system has the capability of self-operating and communicating/sensing with other agents. There are two key topics in the research of multiagent systems: the design of cooperative control laws, and the controllability of networks. The network communication topology plays a key role in accomplishing consensus tasks. From this perspective, several different communication strategies have been proposed[2][7]. A typical undirected communication network is one in witch all agents have global position information of the group, but this reliable topology presents some problems as the amount of agents increases, since one agent need to obtain position information from all other agents, which causes heavy payload for communication channels. Another popular method is the leader-follower model, in which one agent plays as the leader, and other agents communicate with the leader when performing the tasks. This model has much less demanding communication requirement. Nevertheless, the entire system breaks down once the leader agent is disabled. In this research, we tried to observe a reliable communication topology with little restrictions and satisfactory performance. The directed communication topology with the least restrictions will be presented in this thesis together with a recently proposed new control law with discontinuous control gains for this topology.

In this thesis, we experimentally validate the effectiveness of the nonlinear cooperative control proposed in [1], which is demonstrated through discontinuous cooperative control for consensus of multiagent systems with switching topologies and time-delays using mobile robots[1][2][12][15]. By designing nonlinear piecewise control gains, the rendezvous or formation of multiangent systems with switching topologies and time-delays are achieved both in software simulations and hardware experiments.

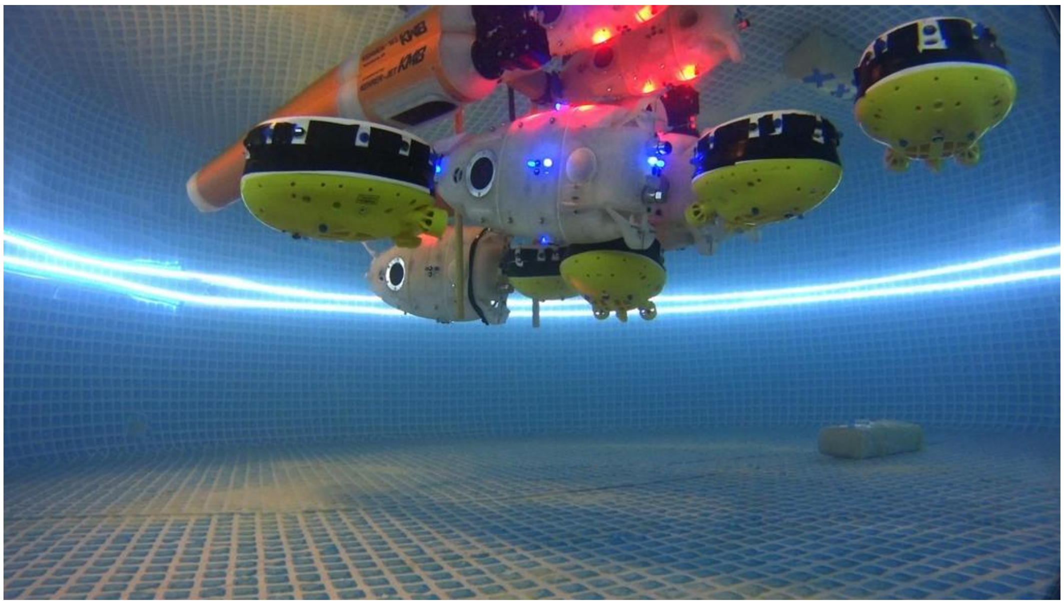
FIGURE 1.1: A fleet of AUVs

FIGURE 1.2: A fleet of UAVs

Chapter 2

Relevant Literature and Research Work

2.1 System Dynamics

Recently the development of cooperative control dynamical system has been improved in linear dynamical systems [6], in terms of both design stability and consensus efficiency. Also, significant progress has been made in nonlinear system control design [7].

2.1.1 Linear Dynamical Systems

In a continuous-time autonomous or networked linear dynamical system, we usually have the system model of (1), which specifies time derivatives of the systems variables that are real and continuous in time.

$$\dot{x}(t) = A * x(t) \tag{1}$$

In this equation $x(t) \in R^n$, n is the number of agents in the system, and A is a constant or a dynamic matrix that is not time dependent. The system model and diagram are shown in figure 2.1 and 2.2.

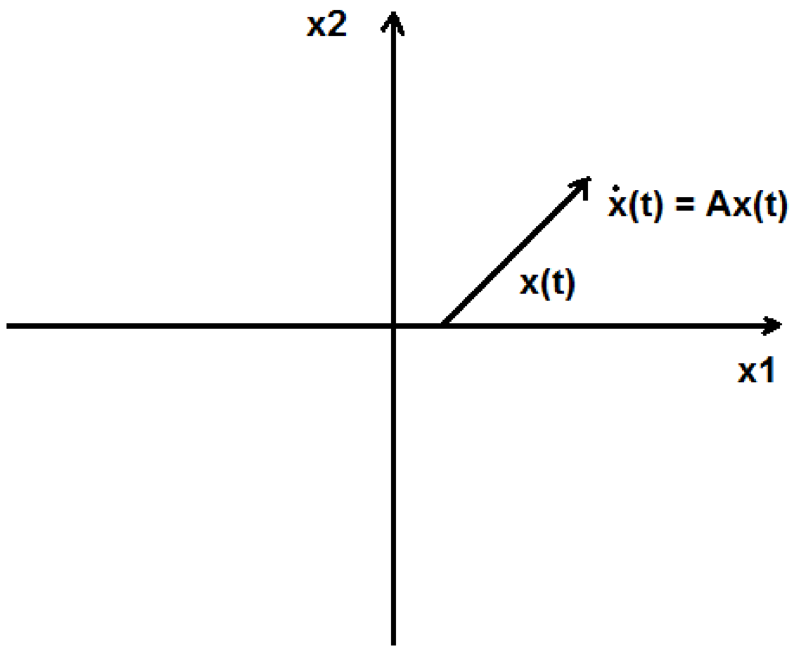

FIGURE 2.1: System Model

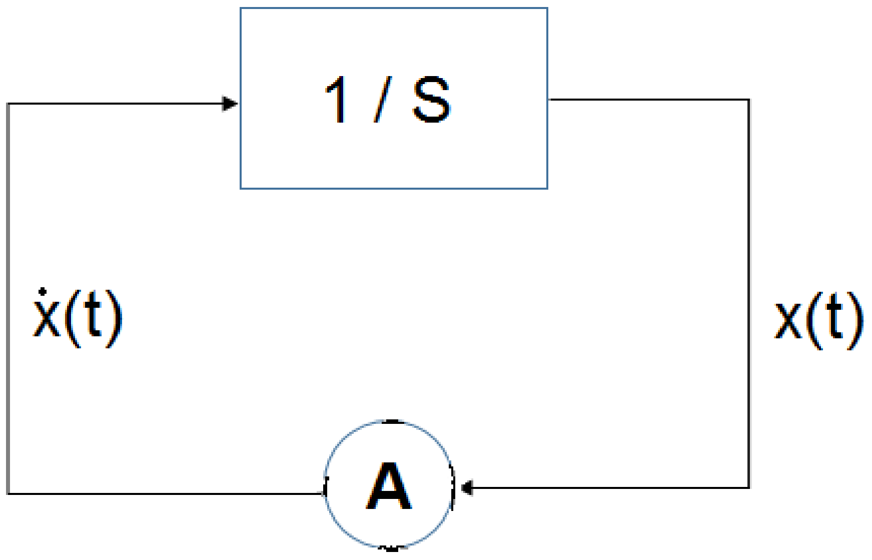

FIGURE 2.2: System Diagram

2.1.2 Nonlinear Dynamical Systems

The linear system we mentioned above does have its significance in theoretical study, and many linearized system models can be used to solve certain parts of nonlinear systems. On the other hand, a nonlinear system in the real world is often much more complicated, especially when some random system deviations exist, such as interferences or time-delay [10][8].

For general nonlinear systems, their design has seen fast development recent years with the help of Lyapunov function and passivity design [12][13]. Lyapunov function in autonomous system can be expressed as (2),

$$x^* = 0 \tag{2}$$

where x^* is an equilibrium of the autonomous system $\dot{x} = f(x)$, and Lypaunov function V can be defined with time derivative as in (3).

$$\dot{V}(x) = \frac{d}{dt} V(x(t)) = \frac{\partial v}{\partial x} \cdot \frac{dx}{dt} = \nabla V \cdot \dot{x} = \nabla V \cdot f(\lambda) \tag{3}$$

2.2 Communication Topologies

Communication topologies describe the communication of information among a fleet of agents. It can be expressed in matrix as (4),

$$S(t_k^s) = \begin{bmatrix} s_{11} & a_{12}(t_k^s) & \cdots & a_{1m}(t_k^s) \\ s_{21}(t_k^s) & a_{22} & \cdots & a_{2m}(t_k^s) \\ \vdots & \vdots & \ddots & \vdots \\ s_{m1}(t_k^s) & s_{m2}(t_k^s) & \cdots & a_{mm} \end{bmatrix}, \tag{4}$$

where at $(t_k^s) : k = 0, 1, \cdots$, the ith agent receives velocity, orientation and position information from agent j, if $s_{ij}(t_k^s) = 1$. Otherwise, if $s_{ij}(t_k^s) = 0$, there is no communication between agent i and agent j.

In generaly, there are two types of communication topologies, directed and undirected.

2.2.1 Undirected Communication Topologies

In the undirected communication topologies, there is no restriction on the direction of communication, which means all agents have the ability to both receive and transmit data (such as position information). For example, the mesh topology represents a network which ensures that every agent is connected to all the remaining agents in the fleet, as described in (5) and figure 2.3. This topology's main advantage is fault tolerance.

$$S(t_k^s) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \tag{5}$$

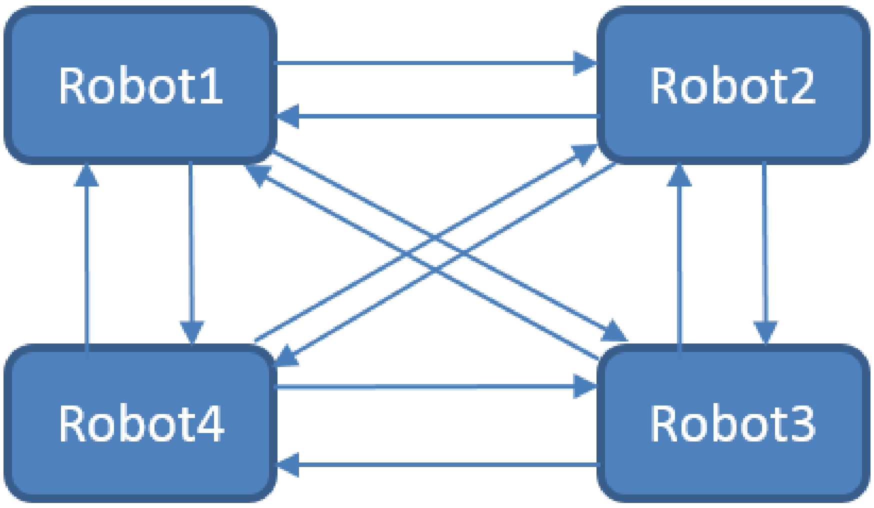

FIGURE 2.3: Global Communication Diagram

There is another common undirected communication topology named leader-follower model, as described in (6) and figure 2.4. One leader agent in the fleet communicates with all the remaining agents. In this topolgy, the number of communication channels and the required communication capacity are reduced. This topology works well as long as the leader is functioning normally and has sufficient communication capability. This model is widely used in cooperative fleet control.

$$S(t_k^s) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}, \tag{6}$$

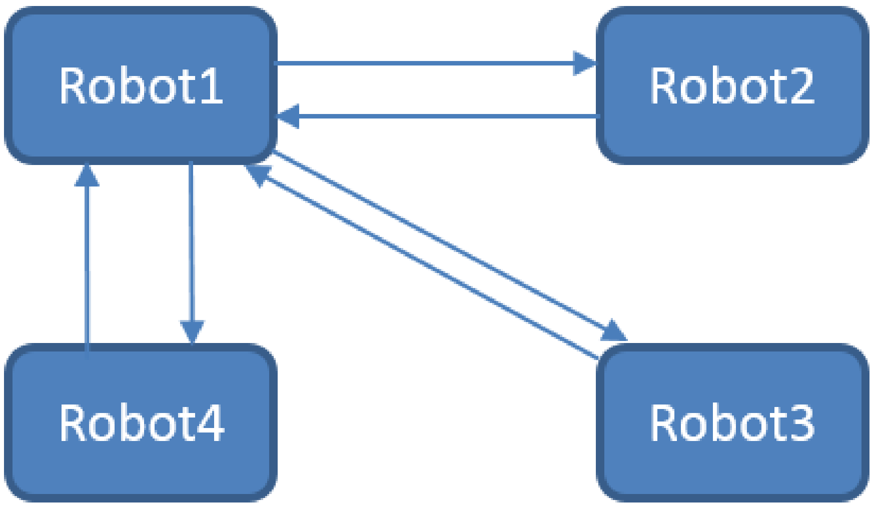

FIGURE 2.4: Leader Follower Communication Diagram

2.2.2 Directed Communication Topologies

Directed communication topologies are defined as: there is only one way communication between two agents, either receiving or transmitting. In this thesis, we seek a directed communication topology that is least demanding (restrictive) while still guarantees multi-agent system consensus. To assure system consensus, this least restrictive topology should satisfy the so-called sequential completeness condition according to [1]. To understand this condition, we first note that if the sensing matrix sequence is irreducible (corresponding directed graph is strongly connected), it is sufficient to guarantee consensus. But there is redundancy in communication. If the sensing matrix sequence is reducible, it can be converted into a lower triangular form. Intuitively, the least restrictive condition should ensure that no subset of agents are disconnected from the remaining agents permanently. From the graph theory perspective, a spanning tree should exist. Mathematically, we can have the following two definitions regarding the sequential completeness concept, which specifies the least restrictive communication topology. An example of such a communication topology matrix and its corresponding digram is presented in (7) and Figure 2.5.

Definition 1: Sensing matrix sequence $S(t)$ is sequentially lower-triangularly complete, if it is sequentially lower-triangular and in every row i of its lower triangular canonical form, there is at least one $j < i$ such that, the corresponding block is uniformly non vanishing.

Definition 2: Sensing/communication matrix sequence $S(t)$ is sequentially complete, if the sequence contains an infinite subsequence that is sequentially lower triangularly complete. Equivalent to a spanning tree in graph theory.

$$S(t_k^s) = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}, \tag{6}$$

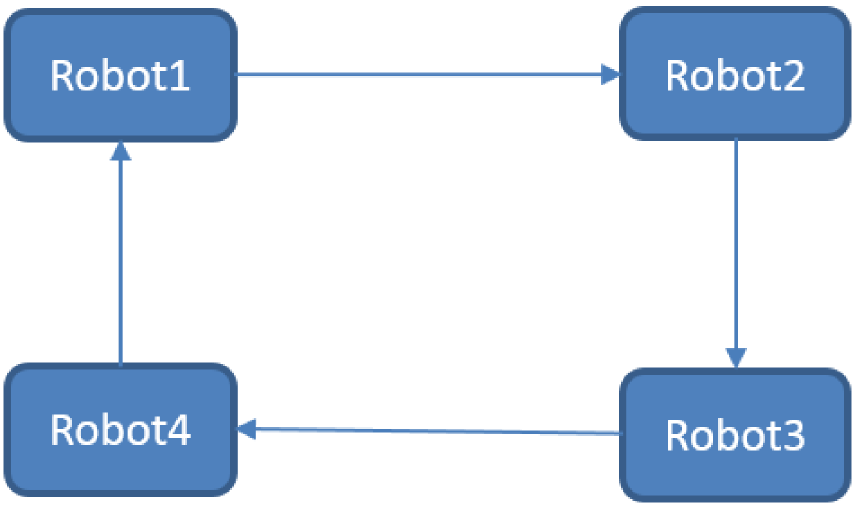

FIGURE 2.5: Least Restrictive Communication Diagram

2.3 Cooperative Fleet Control

2.3.1 Nonholonomic Ground Vehicles

In the general nonlinear system, impressive development in the cooperative control design has been made. Nonetheless, there are still no unified cooperative control laws that can be directly applied to dynamic systems with multiple agents with inherent nonlinear kinematic constraints. For example, a nonholonomic system is a system whose state depends on the path taken to achieve it [4][8][10].

In robotics, if the total degree of freedom are larger than the controllable parameters, we have a nonholonomic system. It has wide applications, such as ground vehicles with wheels and steering, fixed wing jets, etc. In our project, Amigo mobile robots, as shown in figure 2.6, were used in hardware experiments to validate the newly proposed cooperative fleet control.

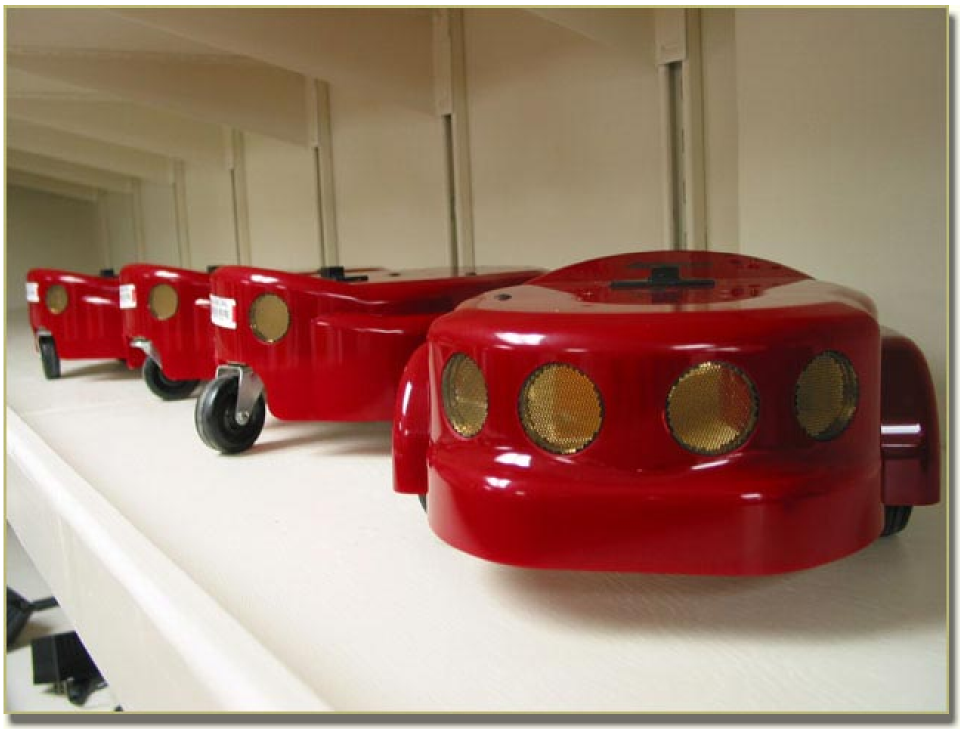

FIGURE 2.6: Amigo Mobile Robot

2.3.2 Underwater Vehicles

The newly proposed consensus algorithm is able to operate in tough environments, especially in the presence of strong interference and with low transmission/receiving communication rate. The underwater environment provides such an experimental scenario to test the consonsus algorithm. For the past couple of years, I have been the project leader of the Eco-dolphins team at ERAU, and we have been building a fleet of AUVs, which can be used to validate cooperative fleet control algorithms.

We sequentially developed the eco-dolphins during the past two years. We have three underwater vehicles (Eco-dolphins), as shown in figure 2.7. The Yellow Dolphin was fully tested in its mission destinations Indiana River Lagoon and littoral water near Key Largo. The Blue Dolphin was assembled but needed additional calibration. The Red Dolphin is still under construction. The streamlined hulls are elliptical shells with four feet along the major axe and one foot along the minor axe. Besides the three AUVs, the Eco-Dolphin system also includes a laptop that installs the ground station program and user interface, an acoustic positioning subsystem, and dual mode communication subsystem. The position of AUVs can be tracked by the acoustic positioning subsystem installed on three buoys while submerged and by GPS while surfaced. The communication subsystem can relay wireless signals transmitted from distant ground station to underwater sonar signals for submerged AUVs. With existing mechatronic devices and successful MATLAB simulations, the fleet of AUVs are expected to run cooperatively with minimal supervision from the ground station. The fleet of three AUVs is expected to perform with full functionalities in summer 2015.

The first phase of design, production and assembly of the yellow Eco-Dolphin prototype, was performed in twelve months. The design includes an internal attitude control system, combined with internal propulsion from brushless direct current thrusters, thus allowing the vehicle to ascend and descend.

The team has also successfully completed the second phase of the program, which involved tracking the Eco-Dolphins while submerged underwater. Work has been conducted to add a GPS system for surface tracking. Converting the acoustic system from tethered to wireless made the ground station more robust. The Eco-Dolphin is configured with recently developed control system software that utilizes a relay combination of Wireless, Sonar and GPS radio wave communication.

The current progress on the blue Eco-dolphin was achieved by the end of summer 2014, and the dolphin was tested in littoral waters of central Florida, blue spring, mosquito Largo, Key Largo. As part of NASA Extreme Environment Mission Operations 19 teams, Eco-dolphin finished the extreme environment testing in Key Largo with cooperative wireless fleet control and navigation mission. The stories appeared in the headlines of Daytona Beach News Journal's Science and Education section twice. Also, the Eco-dolphin team was invited to NASA NEEMO 20 in 2015.

The Fourth phase involves the addition of three sequential (yellow, blue, red) vehicles, therefore allows for better position and orientation data to be sent to the teams buoy network. The three vehicles and the three-buoy communication structure increase the data points collected for surveillance and underwater mapping purposes.

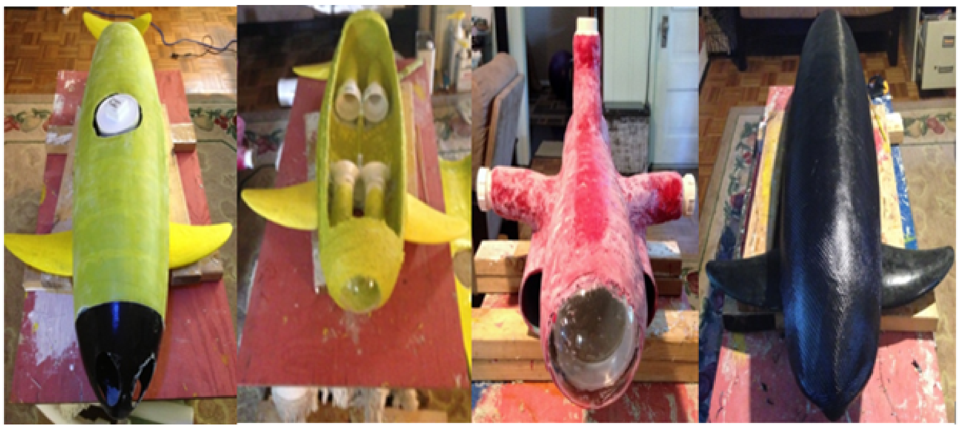

FIGURE 2.7: Underwater Vehicles

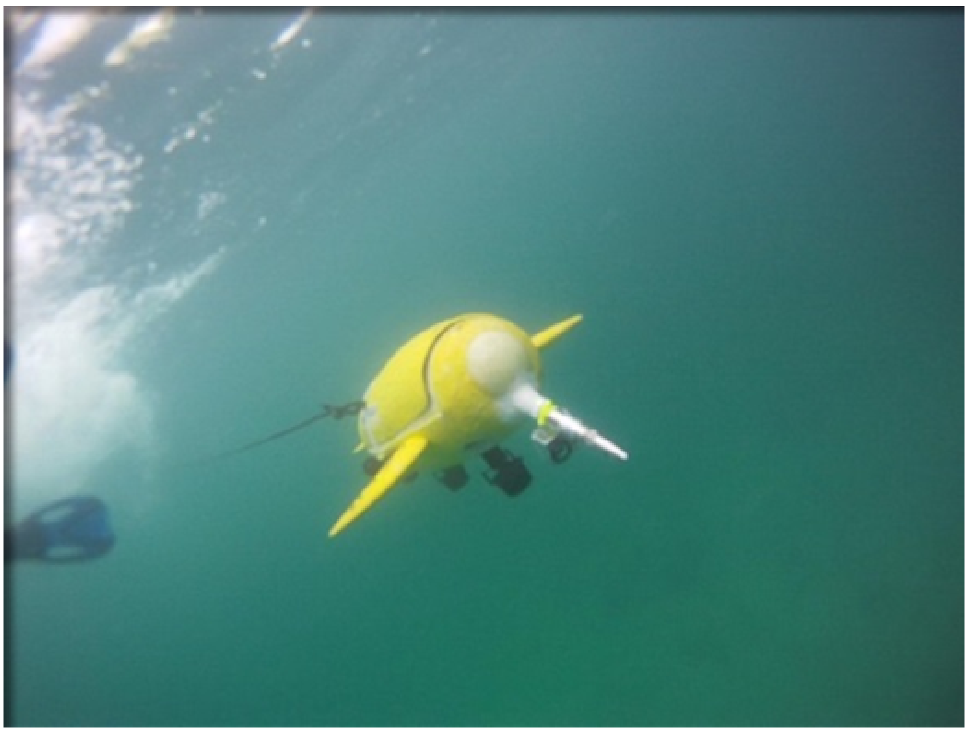

FIGURE 2.8: Testing Figure

Chapter 3

Research Methodology

3.1 Problem Formulation

The 2D dynamics of a group of mobile agents are expressed as

$$\dot{x}_i = v_i \cos\theta_i, \dot{y}_i = v_i \sin\theta_i, \dot{\theta}_i = \omega_i \tag{1}$$

where x_i and y_i denote the position of the ith agent, θ_i shows the orientation which is based on the driving velocity v_i and steering velocity ω_i. In this case $(x_i, y_i) \in \mathbb{R}^2$, $(v_i, \omega_i) \in \mathbb{R}$ and $i \in 1, \cdots, n$.

Let us define the desired trajectory for the group of agents as

$$q_0(t) = [x_0(t), y_0(t)]^T \in \mathbb{R}^2 \tag{2}$$

And the motion frame is denoted as $\mathbb{F}(t)$, which can be considered as a constraint in geometric coordinates in terms of relative positions of the robots. $\mathbb{F}(t)$ consists of $q_0(t)$ and the orthonormal vectors, e_1, e_2, as defined below,

$$e_1(t) = \begin{bmatrix} e_{11}(t) \\ e_{12}(t) \end{bmatrix} = \begin{bmatrix} \frac{\dot{x}_0(t)}{\sqrt{[\dot{x}_0(t)]^2 + [\dot{y}_0(t)]^2}} \\ \frac{\dot{y}_0(t)}{\sqrt{[\dot{x}_0(t)]^2 + [\dot{y}_0(t)]^2}} \end{bmatrix},$$

$$e_2(t) = \begin{bmatrix} e_{21}(t) \\ e_{22}(t) \end{bmatrix} = \begin{bmatrix} \frac{\dot{x}_0(t)}{\sqrt{[\dot{x}_0(t)]^2+[\dot{y}_0(t)]^2}} \\ \frac{\dot{y}_0(t)}{\sqrt{[\dot{x}_0(t)]^2+[\dot{y}_0(t)]^2}} \end{bmatrix}, \tag{3}$$

This 2D orthonormal vectors can be easily convert to 3D, and $e_3(t)$ is a vector cross product by $e_1(t)$ and $e_2(t)$ (4),

$$\begin{aligned} e_3(t) &= cross(e_1(t), e_2(t)), \\ e_3(t) &= e_1(t) * e_2(t), \end{aligned} \tag{4}$$

Based on the orthonormal vectors $e_i(t)$ and the trajectory information $q_0(t)$, the agent position in 2D and 3D are given as,

$$\begin{aligned} P_i(t) &= \alpha_{i1} e_1(t) + \alpha_{i2} e_2(t), \\ P_i(t) &= \alpha_{i1} e_1(t) + \alpha_{i2} e_2(t) + \alpha_{i3} e_3(t), \end{aligned} \tag{5}$$

where $P_i(t)$ is the position of the ith robot, and α_{ij} are constants determining the formation shape.

The sensing/communication information exchange among the fleet agents can be expressed by the sensing/communication matrix,

$$S(t_k^s) = \begin{bmatrix} s_{11} & a_{12}(t_k^s) & \cdots & a_{1m}(t_k^s) \\ s_{21}(t_k^s) & a_{22} & \cdots & a_{2m}(t_k^s) \\ \vdots & \vdots & \ddots & \vdots \\ s_{m1}(t_k^s) & s_{m2}(t_k^s) & \cdots & a_{mm} \end{bmatrix}, \tag{6}$$

where at $(t_k^s) : k = 0, 1, \cdots$, the ith agent receives velocity, orientation and position information from agent j, if $s_{ij}(t_k^s) = 1$. Otherwise, if $s_{ij}(t_k^s) = 0$, there is no communication between agent i and agent j.

Define the $sign(z)$ function as,

$$sign(z) = \begin{cases} 1, & z > 0 \\ 0, & z = 0 \\ -1, & z < 0 \end{cases} \tag{7}$$

For $s_{ij}(t_k^s) \neq 0$, the control model can be expressed as,

$$u_i(t) = \sum_{j=1}^{n} \alpha(s_{ij}(t_k^s), P_j(t_k^s)) sign(P_j(t) - P_i(t)), \tag{8}$$

where $t \in [t_k^s, t_{k+1}^s]$, and $\alpha(,)$ is a nonlinear control gain.

3.2 Control Design

In this propject, we use formation control to demonstrate the consensus of multiagent systems with the new control law (8). First, we use the robot model (1), and define $\hat{x} = x + Rcos\theta, \hat{y} = y + Rsin\theta$. Therefore,

$$\dot{\hat{x}} = vcos\theta - Rsin\theta\omega, \dot{\hat{y}} = vsin\theta - Rcos\theta\omega, \tag{9}$$

Now, we can linearize the robot model as (10), and the real control inputs are expressed as (11).

$$\dot{\hat{x}} = u_x, \dot{\hat{x}} = u_y, \tag{10}$$

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -\frac{sin\theta}{R} & \frac{cos\theta}{R} \end{bmatrix} + \begin{bmatrix} u_x \\ u_y \end{bmatrix},$$

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} u_x cos\theta + u_y sin\theta \\ -\frac{u_x sin\theta}{R} + \frac{u_y cos\theta}{R} \end{bmatrix}, \tag{11}$$

The moving velocity for the ith robot during formation to a certain shape following certain predetermined track can be expressed as (12) based on (2),

$$q_i(t) = q_0(t) + \sum_{j=1}^{2} \alpha(s_{ij}(t_k^s), P_j(t_k^s)), \tag{12}$$

This can be considered as the derivative value based on the robot's relative velocity.

The ith robot's control design without time-delays is based on (2)(3)(4)(8)(12), and is given by,

$$u_i = \sum_{j=1}^{n} \alpha(s_{ij}(t_k^s), P_j(t_k^s)) sign(P_j(t) - P_i(t)) + \dot{q}_i(t), \tag{13}$$

in (13), $\alpha(s_{ij}(t_k^s)$ is the nonlinear control gain. $S(t_k^s)$ can be changed to reflect different types of communication strategies such as leader-follower, global communication and neighbor-follower, etc.

The design of nonlinear control gain for global communication and simple directed communication without time-delay can be given as,

$$\alpha(s_{ij}(t_k^s)) = \frac{s_{ij}(t_k^s)}{\sum_{l=1} s_{il}(t_k^s)}, \tag{14}$$

which has been studied in [1]. In this thesis, we show through computer simulations and robots experiments that, the control gain (14) is sufficient for systems with continuous control laws, i.e., no sign function in (13). However, the control law with control gain (14) may fail to achieve consensus. Therefore, we adopt a new nonlinear control law recently proposed in [1]. Let the nonlinear control gain α_{ij} be designed as,

case 1: if $P_i(t_k^s) = max_{j \in N_i} P_j(t_k^s) = min_{j \in N_i} P_j(t_k^s), \alpha_{ij}$can be any bounded positive value.

case 2: if $P_i(t_k^s) \geq max_{j \in N_i} P_j(t_k^s)$, then $alpha(s_{ij}$ can be ranged,

$$0 \leq \sum_{j \in N_i}^{n} \alpha(s_{ij}(t_k^s) < \frac{P_i(t_k^s) - min_{j \in N_i} P_j(t_k^s)}{c}, \tag{15}$$

case 3: if $P_i(t_k^s) \leq min_{j \in N_i} P_j(t_k^s)$, then $alpha(s_{ij}$ can be selected,

$$0 \leq \sum_{j \in N_i}^{n} \alpha(s_{ij}(t_k^s) < \frac{max_{j \in N_i} P_j(t_k^s) - P_i(t_k^s)}{c}, \tag{16}$$

case 4: if $min_{j \in N_i} P_j(t_k^s) < P_j(t_k^s) < max_{j \in N_i} P_j(t_k^s)$, let $alpha(s_{ij}$ be selected to satisfy,

$$0 \leq \sum_{j \in N_i}^{n} \alpha(s_{ij}(t_k^s) < min[(9),(10)], \tag{17}$$

where c could be any positive constant. As proved in [1] analytically, the system consensus is guaranteed with this choice of piecewise nonlinear control gain.

Chapter 4

Simulations and Experiments

4.1 Software Results

In the Matlab environment, we demonstrate the new control algorithm for multiagent systems. All the demos consist of four agents performing the rendezvous or formation control tasks with two communication topologies. One is the undirected communication topology, which means for each pair of two connecting agents, both know each other's information in terms of velocity, position and orientation. The other one is directed communication topology, in which an agent may know a neighboring agent's information, but not vice versa.

4.1.1 2D Software Simulations

To implement the control law shown in (13), for the undirected communication topology, we consider a group of four mobile robots achieving a rhombus formation while following a circular movement. For the general setting as expressed in (12) (13), we set $q_0(t) = [2cost, 2sint]^T$, and $e_1(t) = [-sint, cost]^T, e_2(t) = [-cost, -sint]^T$. The four robots are following a circle, and the control gain applied is (14). The update time $t_s = 0.05s$. First, the control input (13) is adopted, which contains the sign function. Second, to simulate the control law without the sign function, simply remove the sign function part.

In the first set of simulations, the undirected communication topology is adopted and we compare the system performance with and without the sign function. Four robots are designed to either converge to one point or form a certain shape (while making circular movements). $s_{ij}(t_k)$ is applied as,

$$s1 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}, s2 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$s3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, s4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \tag{18}$$

The obtained trajectories are shown in figures 4.1-4.4.

Figure 4.1 shows rendezvous of four agents without the sign function with undirected communication topology. They start from four initial positions and converge to one point in T=4.7s. Figure 4.2 shows rendezvous of four agents with the sign function and undirected communication topology, starting from four different positions and converging to one point. Time to consensus is T=3.6s.

Figure 4.3 shows four robots starting from different positions and forming a rhombus while following a circle. We use the sign function and the undirected communication topology. Time to the desired formation is T=3.3s. Figure 4.4 shows four robots starting from different positions forming a rhombus while following a circle. Here we do not use the sign function, and adopt the undirected communication topology. Time to the desired formation is t=4.4s

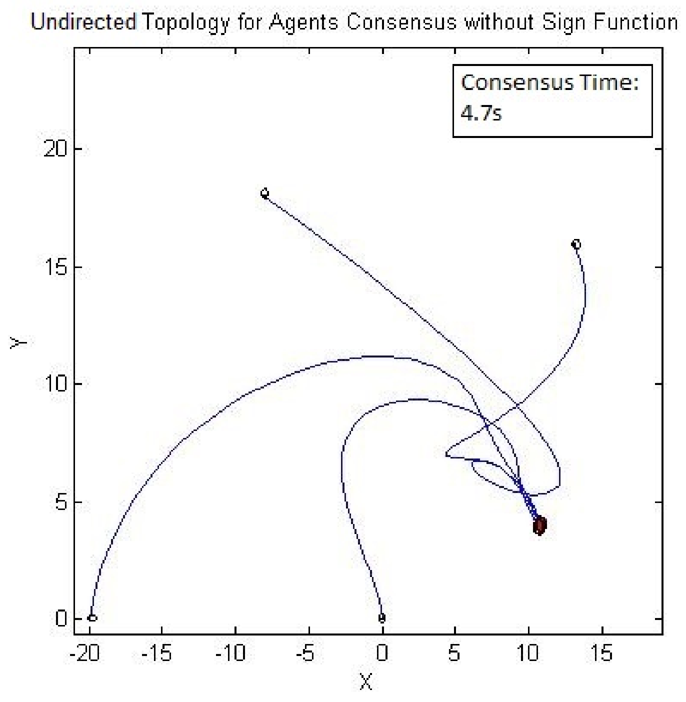

FIGURE 4.1: Rendezvous of four agents without the sign function with undirected communication topology

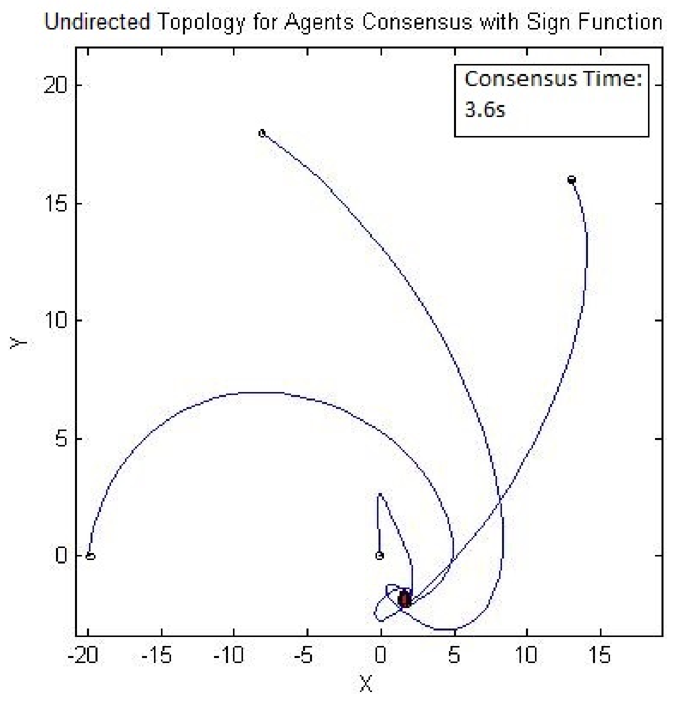

FIGURE 4.2: Rendezvous of four agents with the sign function with undirected communication topology

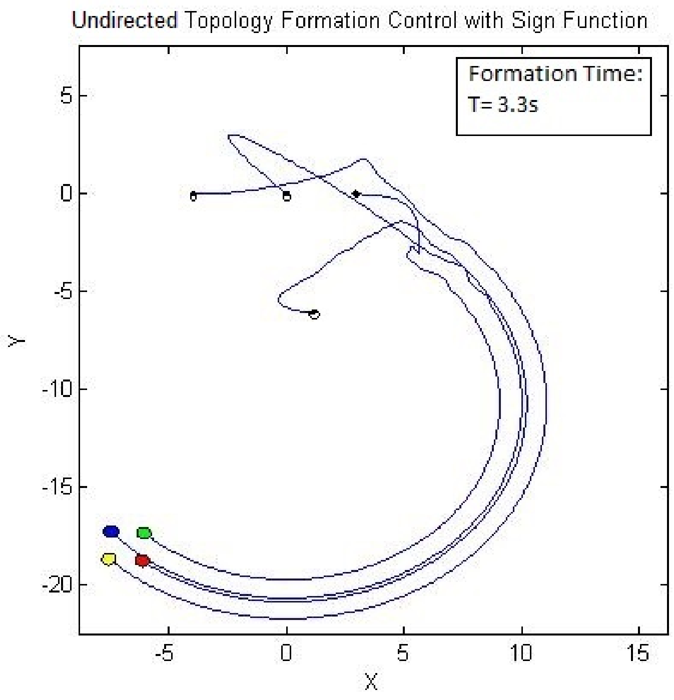

FIGURE 4.3: Formation control with sign function and undirected communication topology

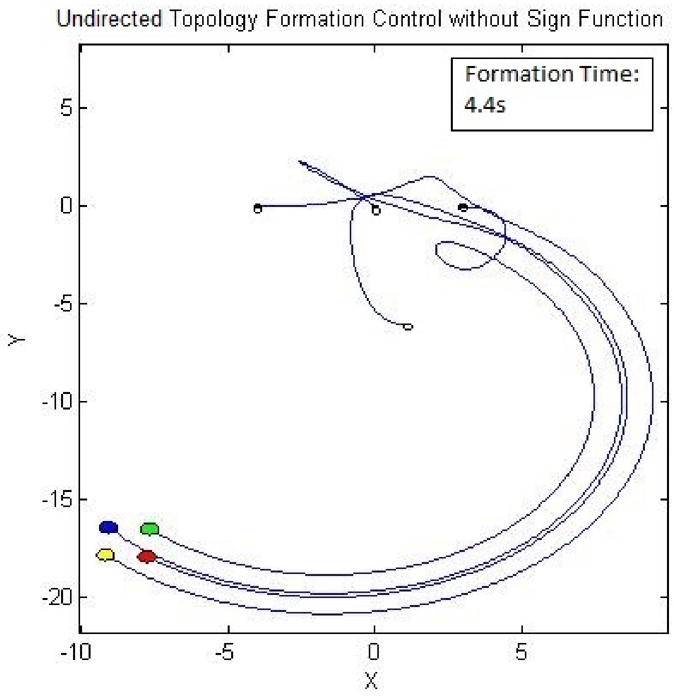

FIGURE 4.4: Formation control without sign function with undirected communication topology

For the directed communication topology, we adopt the least-restrictive neighbor communication topology, with each robot only communicating with a neighboring agent. There is no leader in the system, and minimum data transferring is incurred. $s_{ij}(t_k)$ is set as,

$$s1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}, s2 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$s3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, s4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{19}$$

The control law with time-delays and the sign function can be expressed as (21), with τ_{ij} set to one second,

$$u_i = \sum_{l=1}^{n} \alpha(s_{ij}(t_k^s), P_j(t_k^s) - \tau_{ij}) sign(P_j(t - \tau_{ij}) - P_i(t)) + \dot{q}_i(t), \tag{21}$$

The control gain design is specified in (14)(15)(16)(17), i.e., every time an agent receives a neighbor's position information, the software compares this value with the maximum or minimum value as shown in (14)(15)(16)(17), then the control gain is selected accordingly.

As we mentioned before, the control gain in (14) may not be appropriate for the control law with the sign function. Agents may fail to form the desired shape. Figure 4.5 below gives an example of such scenario with the continuous control gain (14). The four agents fail to converge within 60s and the trajectory contains oscillation.

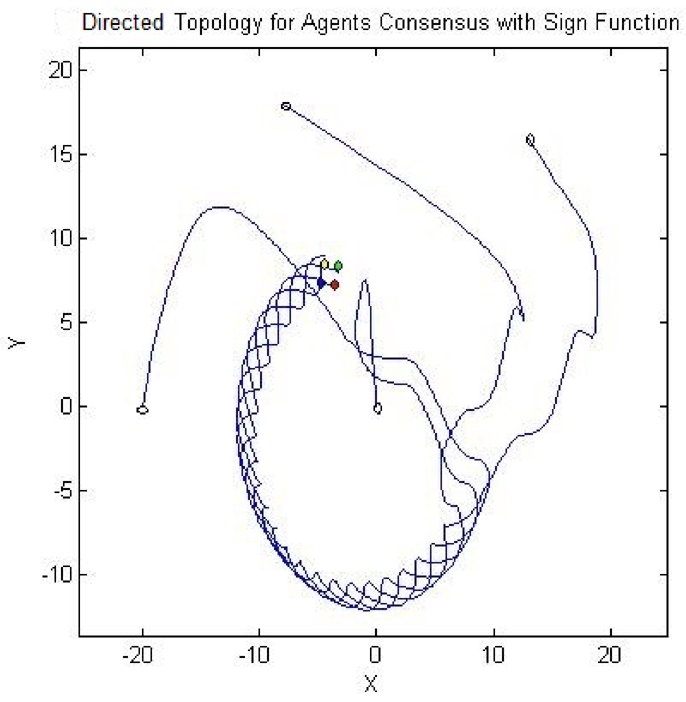

FIGURE 4.5: Rendezvous of four agents with the sign function and time-delays with the directed communication topology

As we apply the rules (14)(15)(16)(17), the system works satisfactorily under the same conditions, as shown in figure 4.6. A formation example is also shown in figure 4.7.

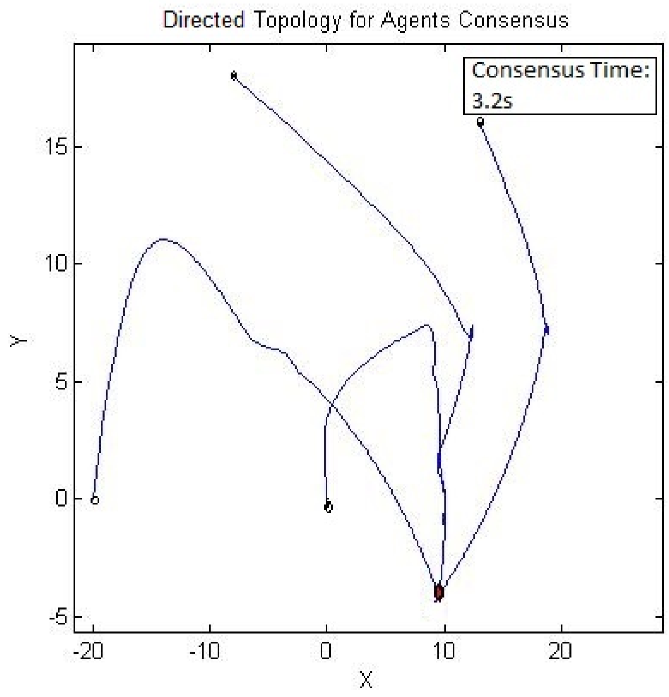

FIGURE 4.6: Rendezvous of four agents using new control laws with the sign function and time-delays and the directed communication topology

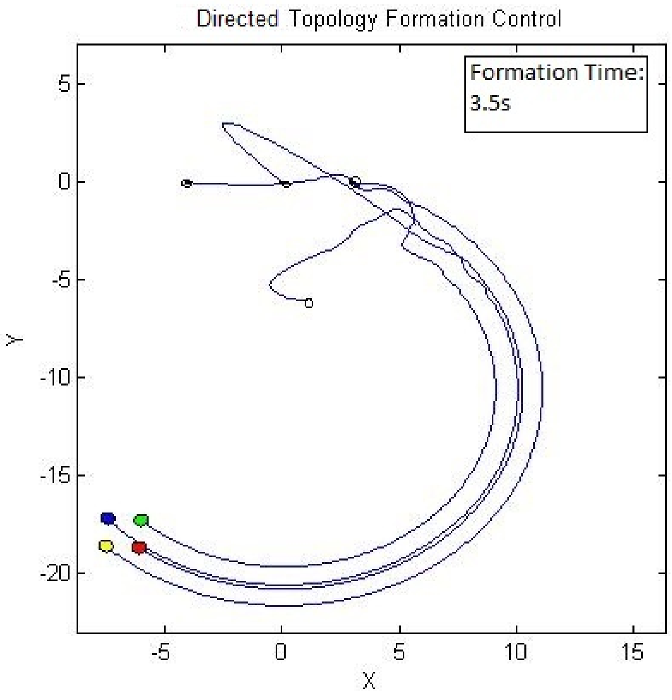

FIGURE 4.7: Formation control using new control laws with the sign function and time-delays and the directed communication topology

4.1.2 3D Software Simulations

We also implemented the new control law in 3D conditions in MATLAB. Four agents with initial positions: r1(0,0,0), r2(0,0,0), r3(0,0,0), r4(0,0,0) converge to the formation shape: r1(0,0,0), r2(-1,1,0), r3(-1,-1,0), r4(-2,0,0).

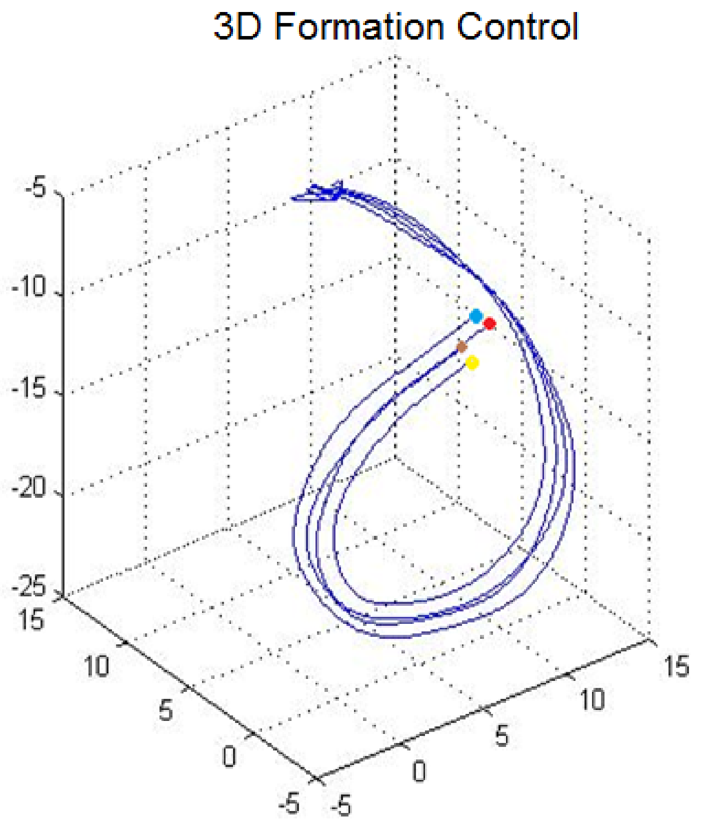

FIGURE 4.8: Formation control in 3D: a cuboid moving in the circle

4.2 Hardware Experiments

4.2.1 Aira Ground Vehicles Experiments

We implemented the new control law in Aira mobile robots. The robots know the initial position of themselves, but when they are moving, each robot only knows the velocity and the position information from one of its neighboring robots. $\dot{q}_i(t)$ is set based on the requirement of the consensus speed. All the Aira demos shown are for the distributed formation control of multiagent systems with switching topologies. $\dot{q}_i(t)$ is set to 200*mm* in both x and y orientations. The communication topology (19) is adopted with the control law (13). At this time, the control gains are designed as Kx=120 and Ky=20. Figure 4.9 shows the position information of robot 1, and figure 4.10 compares the experimental results with the theory.

Referring to Robot 1 and Robot 2, Y is the distance between two robots in x and y domains. We sampled 120 points, one point per second.

From figure 4.9 we can see the experimental value Kx=120 fits the theoretical control gain very well based on (15). Similarly, for the y domain, the experimental value fits well with the theoretical value (17).

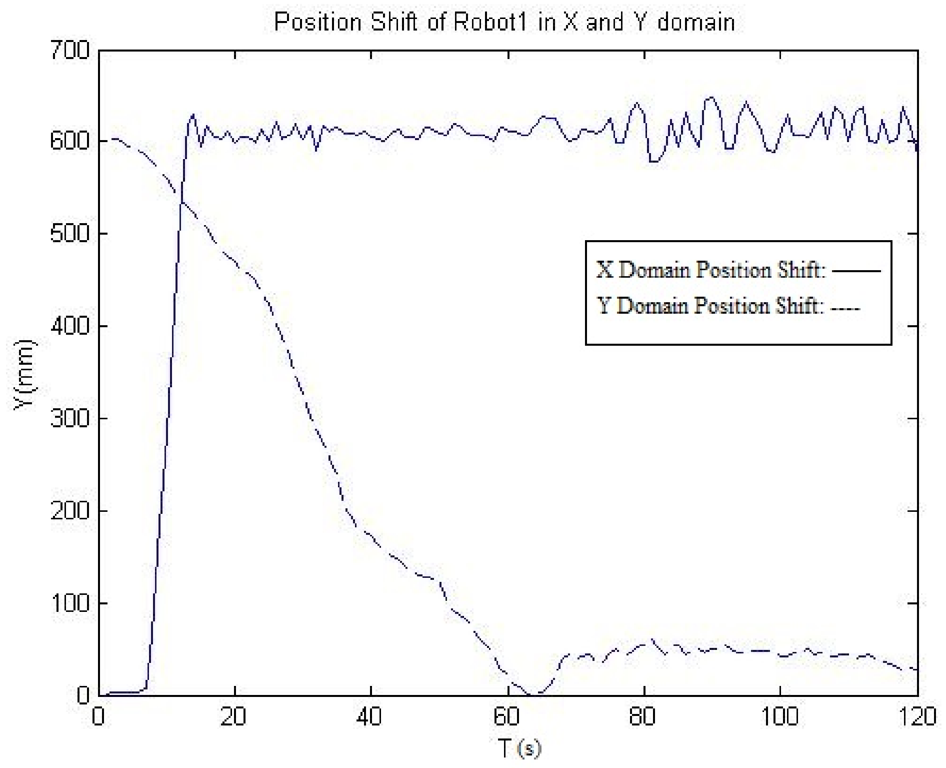

FIGURE 4.9: Practical position shift data from robot1 and robot2

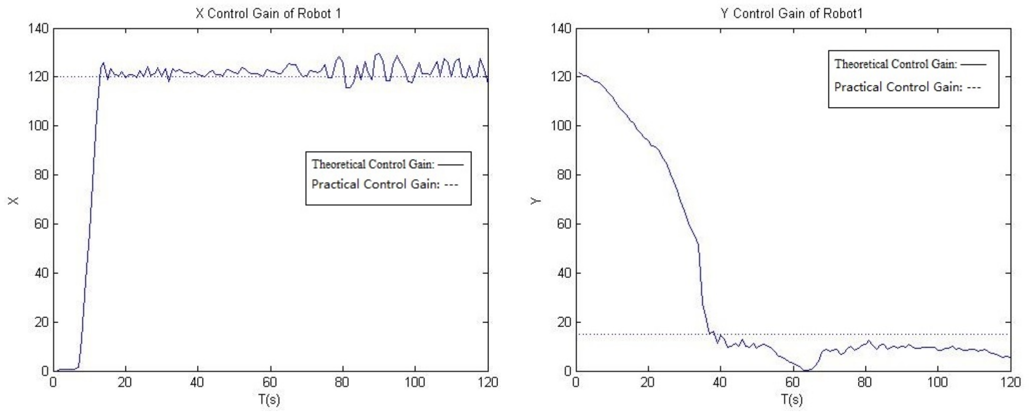

FIGURE 4.10: Comparison between theoretical values with experimental values in discontinuous control gain

Figures 4.11 and 4.12 are a series of images showing the Aira robot simulations in our lab.

Figure 4.11 shows the Rectangle-to-line and line-to-rectangle formation control with directed communication and the sign function.

Figure 4.12 shows the formation shape changing from rectangle to line, then to rhombus, and finally converging to one point. K is set to 60, and the moving velocity $\dot{q}_i(t) = 100mm$ with the same communication topology (19).

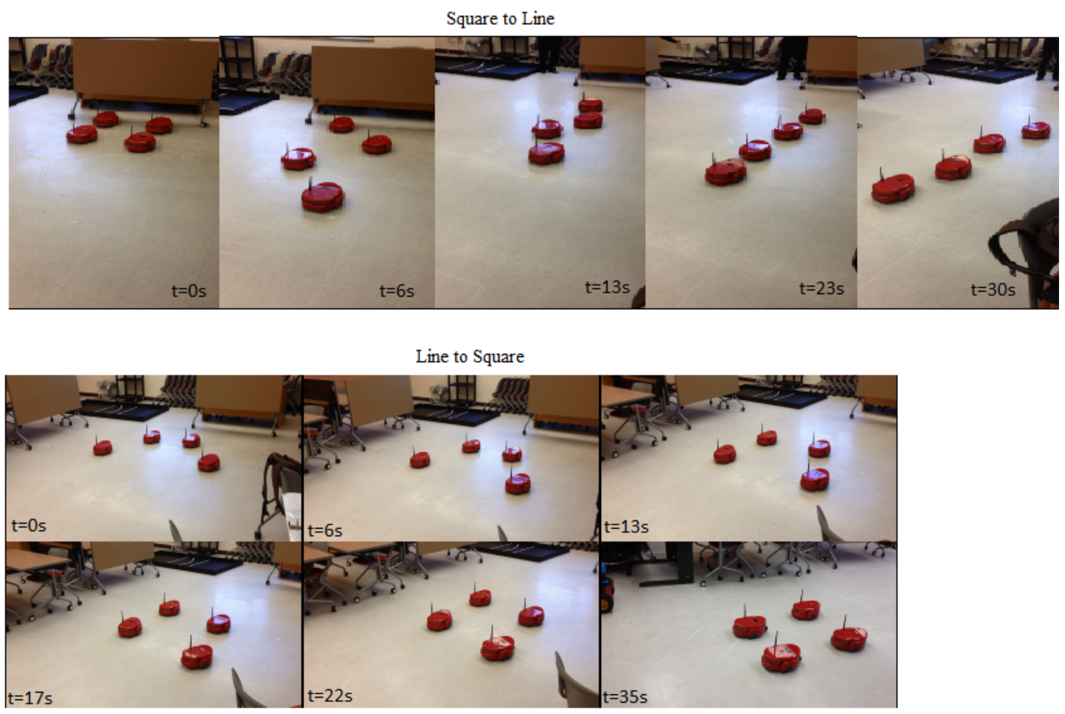

FIGURE 4.11: Rectangle-to-line and line-to-rectangle formation control with directed communication and the sign function

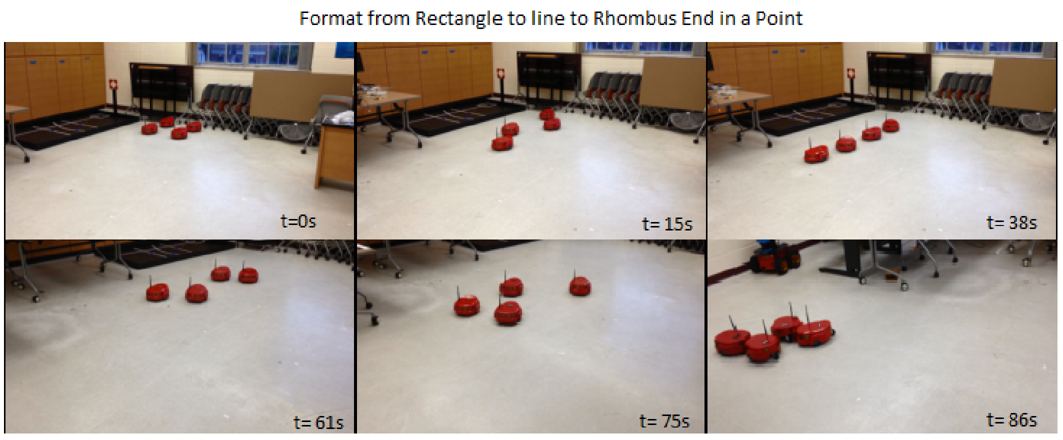

FIGURE 4.12: Formation change from rectangle to line to rhombus, ending with a point with directed communication and the sign function

4.2.2 Eco-dolphin Underwater Vehicles Experiments

The AUVs share the same ground station, hull mold, type of thrusters, as well as similar autopilot and positioning programs, electronic circuit control board, etc. However, the three AUVs are also designed differently with complementary features.

4.2.2.1 System Requirements

Here we describe the key components that distinguish the Eco-Dolphin fleet from other robots. The system is divided into three major components: 1) Three AUVs, 2) Ground Station, and 3) Communication Network. The three AUVs run as a cooperative fleet to collect heterogeneous data near the boat where the ground station is located. This function demands sophisticated communication systems among AUVs, as well as between individual AUV and ground Station. Since electromagnetic radio signals do not work under salty water for its high conductivity, acoustic communication is required when the AUVs are submerged. Radio communication is also required to connect to ground stations when AUVs are surfaced. Since AUVs cannot receive GPS signals underwater, the fleet has to rely on acoustic positioning system. Hence, the Eco-Dolphin requires dual communication systems as well as dual positioning system. Based on the size of data collection sites and coverage of acoustic positioning range, the cruising distance from the boat is designed to be about a square kilometer. The maximum cruise speed is about 2 meters per second so that it can work under mild tidy current conditions. Blue Dolphin can dive to calm water in 10 feet deep to improve acoustic communication. The Yellow Dolphin can dive to 30 feet, and the Red Dolphin is designed to dive to 100 feet deep. The onboard battery needs to run for an hour when all thrusters run at maximum speed, and two hours continuously when the AUV runs normally. This means that the AUV can have four-hour service time under water if the thrusters are on half of the time. The Yellow dolphin is equipped with camera and hydrophones, and the Red Dolphin is equipped with biochemical sensors and water sample collection devices. The Red Dolphin shares the similar system diagram with the Yellow Dolphin, except for the diving depth and payloads. Figure 4.13 and 4.14 illustrate the slightly different system diagrams of the Yellow Dolphin and the Blue Dolphin.

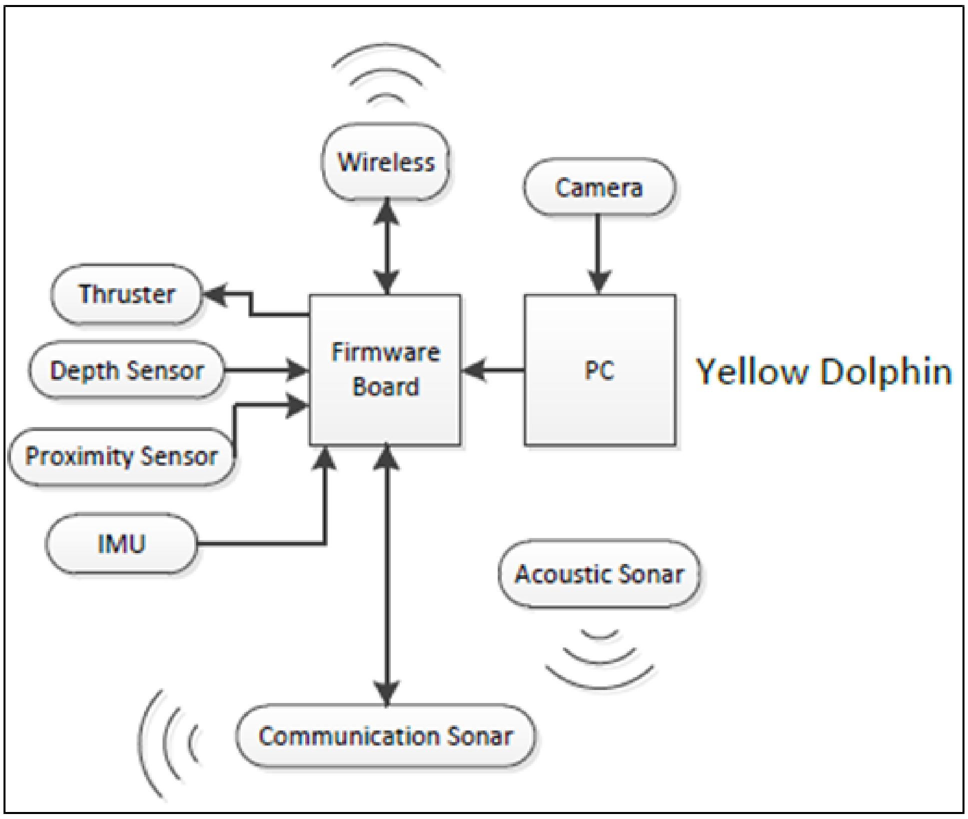

FIGURE 4.13: Yellow dolphin system diagram

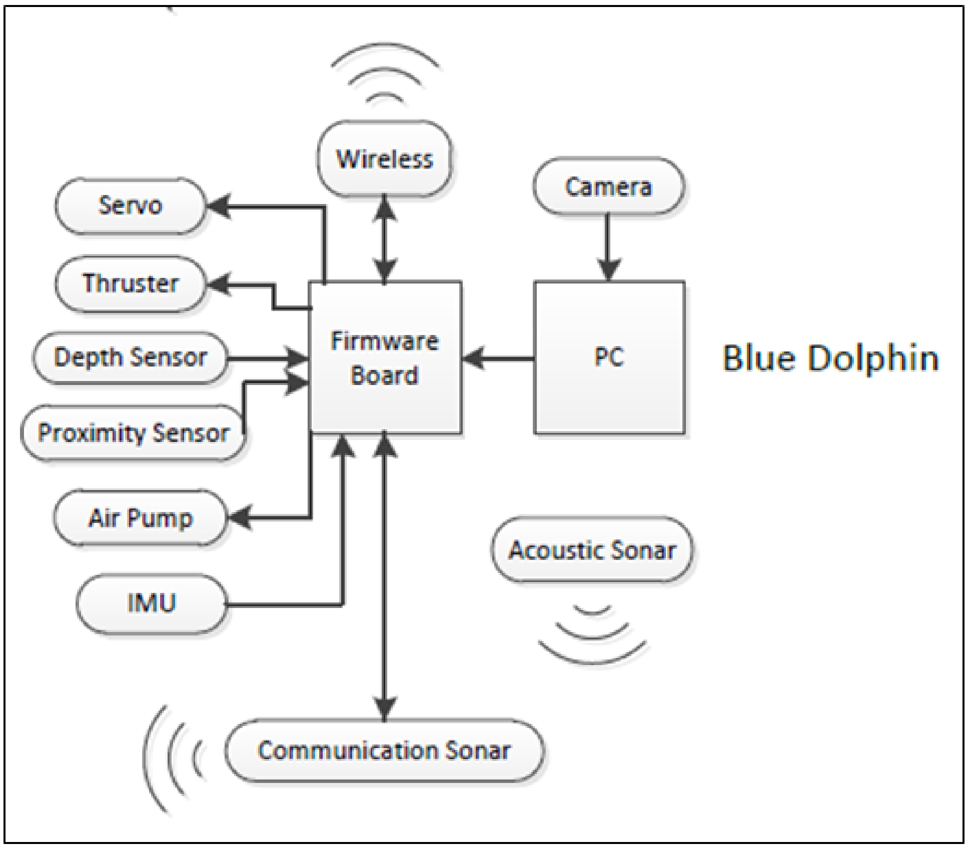

FIGURE 4.14: Blue dolphin system diagram

The top layer subsystems include a ground station, three AUVs, and scientific data collection payloads on AUVs. In the middle layer, the ground station includes three acoustic positioning transducers on buoys, WIFI and acoustic communication modems, and a laptop hosting the user interface for sending commands and observing AUV states and positions. An AUV includes a Supervising Program installed in a SBC (Single Board Computer, e.g. PC104) and an Autopilot Program installed in Arduino. In the low layer hardware, each AUV includes propellers (thrusters), ballasts, two acoustic modems for communication and positioning, Navigation devices such as GPS, IMU (Inertial Measurement Unit), Compass, Depth Sonar, Short Range Obstacle Avoidance Sonar and nexus of sensors for system safety, such as depth/temperature/humidity sensors. The seven ovals in the middle layer represent the seven transactions logically connecting hardware and software objects. Any path is a chain of intertwining objects and processes that represent typical application scenarios.

4.2.2.2 Contributions and Future Work

The set of three buoys allows the Eco-Dolphin team to track the location of a group of AUVs while they are submerged under water. The buoy system acts as the relay station from surface operations to underwater operations, which is a necessary link in communication required for navigation and spatial awareness of the platform design. This segment started with designing the three buoys, taking the product into manufacturing, and redesigning details to fulfill the tasks required of the network. Testing was performed on both the buoyancy and stability of the buoys in Key Largo. Tracking of an underwater vehicle was performed on three accounts under varying conditions to give the team a baseline for future research and implementation of cooperative spatial awareness.

During the NEEMO mission, we only tested the sonar communication between one dolphin and ground station via communication sonar, and recorded the position information by positioning sonar. This was the first step to simulate cooperative distributed formation control, and we will further obtain the position information and establish the communication with two sonars.

In the future, we will improve the communication system and test the wireless (WIFI) communication on the surface in order to transmit commands to the dolphins.

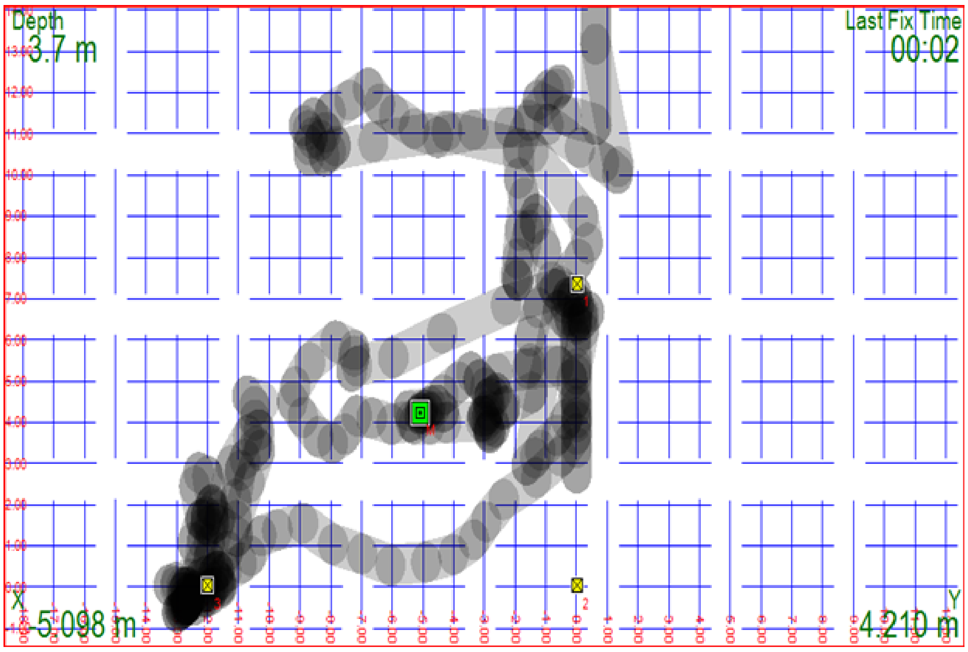

FIGURE 4.15: Position observed by the Position System

Chapter 5

Discussions, Conclusions and Recommendations

In this thesis, we demonstrated a newly proposed distributed cooperative control for consensus of multi-agent systems with switching topologies and time-delays using Matlab and Aira mobile robots. Matlab simulations in 2D and 3D confirmed the advantages of using the sign function with piecewise control gain, which, when compared with the traditional control law without the sign function, guarantees consensus and results in shorter consensus time. The simulation and experiental results validated the effectiveness of the new control design in rendezvous and formation control problems with the least restrictive inter-agent communication requirements.

Another important benefit of the new control law is that, it can be easily implemented in robots, therefore practical implementation issues can be studied experimentally, such as time-delays, hard ware limitations, etc. Formation control experiments with changing topologies and time-delay were simulated with 2D mobile ground vehicles. Currently, each robot does not have the capability of self-computing, and a computer in communication with all robots performs all calculations and sends commands to robot agents. In this sense, our current hardware experiments with Aira Robots are in essence hardware simulations.

For future work, we propose to demonstrate the new control design in 3D with a fleet of autonomous underwater vehicles, with each of them having computing capabilities. The high interference and challenging nature of communication in underwater environment enable us to test the robustness of the new control design. To prepare for this next stage of experimentss, we have finished building two AUVs, and successfully tested communication, motion, navigation and control systems individually. The immediate next step is to implement the new consensus algorithm in these two AUVs.

Appendix A

Appendix: Software Programs

A.1 Partial Matlab Code

A.1.1 System Dynamics and Control in 2D and 3D Simulations

```
% Robot model:
%   \dot\phi_{i1} = \phi_{i2},
%   \dot\phi_{i2} = v_i,
%   \phi_{i1}, \phi_{i2}, v_i \in R^2

% Number of vehicles: Nagent

% The control objective is to move with the leader while coordinating with
% the neighboring agents as well as maintaining the relative positions to the leader

% Leader: psi(t) \in \Re^2 and its time derivatives psidot(t) and \psiddot(t)
%   Case 1: constant velocity as \omega=(0.2, 0.2), omegax=0.2, omegay=0.2;
%   psi =  \omega t,
%   psidot = \omega,
%   psiddot =0,

% The moving frame attached to the leader: e_j(t) \in \Re^2, where
%   e_1(t) = [ psidot_1(t)/\sqrt{ psidot_1^2(t) + psidot_2^2(t) }
%                psidot_2(t)/\sqrt{ psidot_1^2(t) + psidot_2^2(t) } ]
%   e_2(t) = [ - psidot_2(t)/\sqrt{ psidot_1^2(t) + psidot_2^2(t) }
%                psidot_1(t)/\sqrt{ psidot_1^2(t) + psidot_2^2(t) } ]
% Need to calculate the derivatives: edot_i(t) and eddot_i(t)

% Coordinates of the i-th vehicle in the moving frame:
%   delta_i(t) = \alpha_{i1} e_1(t) + \alpha_{i2} e_2(t),
% Its associated matrix is
%   Almatrix = [ \alpha_{11} \alpha_{12}; ... ; \alpha_{i1} \alpha_{i2}; ...]

% State transformation:
%   x_{i1}=\phi_{i1} - psi - delta_i,
%   x_{i2}=\phi_{i2}+ x_{i1}- psidot - deltadot_i ,
% Output
%   y_i = x_{i1},

% Decentralized control and its canonical form:
%   v_i = -  x_{i2} - x_{i1} + u_i + psiddot + deltaddot_i
% under which
%  \dot{x}_{i1} = -x_{i1} + x_{i2};
%  \dot{x}_{i2} = -x_{i1} + u_i

% Cooperative control:
% u_i = G_i y,
% or u= G y

%function zdot=cformation3d(t, z)

%global Nagent S Almatrix;

% For Nagent robots: z \in R^(Nagent*4) since each robot has 4
% differential equations to be solved (2 axis, 2nd order each)

% Leader: psi(t) \in \Re^2 and its time derivatives psidot(t) and \psiddot(t)
%2d circle where z is constant
%psi=[10*sin(0.1*t) 10*cos(0.1*t) 0];
%psidot=[cos(0.1*t) -sin(0.1*t) 0];
%psiddot=[-0.1*sin(0.1*t) -0.1*cos(0.1*t) 0];
%psi3dot = [-0.01*cos(0.1*t) 0.01*sin(0.1*t) 0];

%3d
%psi=[10*sin(0.1*t)*sin(0.1*t) 10*sin(0.1*t)*cos(0.1*t) 10*cos(0.1*t)];%10*cos(0.1*t)];
%psidot=[sin(0.2*t) cos(0.1*t)*cos(0.1*t)-sin(0.1*t)*sin(0.1*t) -sin(0.1*t)];%-sin(0.1*t)];
%psiddot=[0.2*sin(0.2*t) -0.2*sin(0.2*t) -0.1*cos(0.1*t)];%-0.1*cos(0.1*t)];
%psi3dot = [0.04*cos(0.2*t) -0.04*cos(0.2*t) 0.01*sin(0.1*t)];%0.01*sin(0.1*t)];

% The moving frame attached to the leader: e_j(t) \in \Re^2, where
%   e_1(t) = [ psidot_1(t)/\sqrt{ psidot_1^2(t) + psidot_2^2(t) }
%               psidot_2(t)/\sqrt{ psidot_1^2(t) + psidot_2^2(t) } ]
%   e_2(t) = [ - psidot_2(t)/\sqrt{ psidot_1^2(t) + psidot_2^2(t) }
%               psidot_1(t)/\sqrt{ psidot_1^2(t) + psidot_2^2(t) } ]
% Need to calculate the derivatives: deltadot_i(t) and deltaddot_i(t)

%e1 = [ psidot(1)/sqrt(psidot(1)^2 + psidot(2)^2); psidot(2)/sqrt(psidot(1)^2 + psidot(2)^2) ];
%e2 = [ -psidot(2)/sqrt(psidot(1)^2 + psidot(2)^2); psidot(1)/sqrt(psidot(1)^2 + psidot(2)^2) ];

%edot1 = [psiddot(1)/sqrt(psidot(1)^2 + psidot(2)^2)-psidot(1)/sqrt((psidot(1)^2 + psidot(2)^2)^3)
*(psidot(1)*psiddot(1)+psidot(2)*psiddot(2));
%    psiddot(2)/sqrt(psidot(1)^2 + psidot(2)^2)-psidot(2)/sqrt((psidot(1)^2 + psidot(2)^2)^3)
```

```
*(psidot(1)*psiddot(1)+psidot(2)*psiddot(2))];

%edot2 = [-edot1(2); edot1(1)];

%eddot1 = [ psi3dot(1)/sqrt(psidot(1)^2 + psidot(2)^2) - 2*psiddot(1)/sqrt((psidot(1)^2 +
psidot(2)^2)^3)*(psidot(1)*psiddot(1)+psidot(2)*psiddot(2))+3*psidot(1)/sqrt((psidot(1)^2+psidot(2)^2)
%psi3dot(2)/sqrt(psidot(1)^2 + psidot(2)^2) - 2*psiddot(2)/sqrt((psidot(1)^2 + psidot(2)^2)^3)*(psidot

%eddot2 = [-eddot1(2); eddot1(1)];

%will need to find e3, and proper e1 and e2 for third dimension
%e1 = [ psidot(1)/sqrt(psidot(1)^2 + psidot(2)^2 + psidot(3)^2); psidot(2)/sqrt(psidot(1)^2 + psidot(2
%e2 = [ -psidot(2)/sqrt(psidot(1)^2 + psidot(2)^2 + psidot(3)^2); psidot(1)/sqrt(psidot(1)^2 + psidot(
%e3 = cross(e1,e2);

%edot1 = [psiddot(1)/sqrt(psidot(1)^2 + psidot(2)^2)-psidot(1)/sqrt((psidot(1)^2 + psidot(2)^2)^3)*(ps
%    psiddot(2)/sqrt(psidot(1)^2 + psidot(2)^2)-psidot(2)/sqrt((psidot(1)^2 + psidot(2)^2)^3)*(psidot(

%edot2 = [-edot1(2); edot1(1); edot1(3)];

%edot3 = cross(edot1, edot2);

%eddot1 = [ psi3dot(1)/sqrt(psidot(1)^2 + psidot(2)^2) - 2*psiddot(1)/sqrt((psidot(1)^2 + psidot(2)^2)
% psi3dot(2)/sqrt(psidot(1)^2 + psidot(2)^2) - 2*psiddot(2)/sqrt((psidot(1)^2 + psidot(2)^2)^3)*(psido

%eddot2 = [-eddot1(2); eddot1(1); eddot1(3)];

%eddot3 = cross(eddot1, eddot2);

% Coordinates of the i-th vehicle in the moving frame:
%   delta_i(t) = \alpha_{i1} e_1(t) + \alpha_{i2} e_2(t),
% Its associated matrix is
%   Almatrix = [ \alpha_{11} \alpha_{12}; ... ; \alpha_{i1} \alpha_{i2};
%   ...]

%for i=1:Nagent
%    delta(i,:) = Almatrix(i,1)* e1' + Almatrix(i,2) * e2' + Almatrix(i,3) * e3';
%    deltadot(i,:) =  Almatrix(i,1)* edot1' + Almatrix(i,2) * edot2' + Almatrix(i,3) * edot3';
%    deltaddot(i,:) =    Almatrix(i,1)* eddot1' + Almatrix(i,2) * eddot2' + Almatrix(i,3) * eddot3';
%end

%for i=1:Nagent
%    rowsum(i)=0;
%    for j=1:Nagent
%        rowsum(i) = rowsum(i) + S(i,j);
%    end
%end

%for i=1:Nagent
% compute u(i,:) for agent i
%u(i,:) = - [z((i-1)*4+1) z((i-1)*4+3)];
```

```
%for l=1:Nagent
%    u(i,:) = u(i,:) + S(i,l)/rowsum(i) * (([z((l-1)*4+1) z((l-1)*4+3)]) + delta(i,:) - delta(l,:));
%end
% compute v(i,:) for agent i
%v(i,:) = u(i,:) - [z((i-1)*4+2) z((i-1)*4+4)] + psidot + psiddot + deltadot(i,:) + deltaddot(i,:);

% compute u(i,:) for agent i
%    u(i,:) = -[z((i-1)*6+1) z((i-1)*6+3) z((i-1)*6+5)];
%    for l=1:Nagent
%        u(i,:) = u(i,:) + S(i,l)/rowsum(i) * (([z((l-1)*6+1) z((l-1)*6+3) z((l-1)*6+5)]) + delta(i,:)
%    end
% compute v(i,:) for agent i
%    v(i,:) = u(i,:) - [z((i-1)*6+2) z((i-1)*6+4) z((j-1)*6+6)] + psidot + psiddot + deltadot(i,:) + d
%end

% System model (this part can also be written as matrix form)
%for the i-th vehicle:
%z((i-1)+1) = \phi_{i1}(1); z((i-1)+2) = \phi_{i1}(2);
%z((i-1)+3) = \phi_{i2}(1); z((i-1)+4) = \phi_{i2}(2);
%for i=1:Nagent
%zdot((i-1)*4+1) = z((i-1)*4+2);
%zdot((i-1)*4+3) = z((i-1)*4+4);

%zdot((i-1)*4+2) = v(i,1);
%zdot((i-1)*4+4) = v(i,2);

%zdot((i-1)*6+1) = z((i-1)*6+2);
%zdot((i-1)*6+3) = z((i-1)*6+4);
%zdot((i-1)*6+5) = z((i-1)*6+6);

%zdot((i-1)*6+2) = v(i,1);
%zdot((i-1)*6+4) = v(i,2);
%zdot((i-1)*6+6) = v(i,3);

%end
%zdot=zdot';
```

A.1.2 Main Program for 2D and 3D Simulations

```
global Nagent S Almatrix;
% Number of agents
%Nagent=10;
Nagent=4;

% Formation shape
%Almatrix = [0 0; -1 1; -1 -1; -2 2; -2 0; -2 -2; -3 3; -3 1; -3 -1; -3 -3];

%2d formation
%Almatrix = [0 0 ; -1 1; -1 -1; -2 0];
```

```
%3d formation
Almatrix = [0 0 0; -1 1 0; -1 -1 0; -2 0 0];

%Almatrix = [0 0; 0 0; 0 0; 0 0];

% Simulation time
%T=46;

% The computing step time
%T1=0.4;
%tt=0:T1:T;

%for j=1:length(tt)-1
%   ttt(j)=tt(j);
%end

%clc;
%c=fix(clock);
%fprintf('started at %2i.%2i.%2i\n\n', ...
%   c(4),c(5),c(6));

% Initial condition

%xx0=[0 -0.5 0 0 -1 1 0 0 0 -1 0 0 1 0 0 0 2 1 0 0 2 3 0 0 4 2.5 0 0 2 5 0 0 3 1 0 0 3 -1 0 0]';
%xx0=[0 -0.5 0 4 -20 3 0 12 13 6 16 -7 -8 20 18 -9]';
%xx0=[0 -5 0 5 -4 4 0 0 3 3 0 0 1 -2 -6 0]';

%[x1 vx1 y1 vy1 x2 vx2 y2 vy2 x3 vx3 y3 vy3 x4 vx4 y4 vy4 ...
%xyz0=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]';

%[x1 vx1 y1 vy1 z1 vz1 ...
xyz0=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]';

%for i=1:length(tt)-1

%    fprintf(' \n i= %i', i);

% if rem(i,2)==1

   %S =  [1   1   1   1;
   %      1   1   1   1;
   %      1   1   1   1;
   %      1   1   1   1;
   % ];

   % else

%  S = [1   1   1   1;
%      1   1   1   1;
%      1   1   1   1;
%      1   1   1   1;
%     ];

%   if rem(i,2)==1

%   S =  [1   1   1   1   1   1;
%         1   1   1   1   1   1;
%         1   1   1   1   1   1;
%         1   1   1   1   1   1;
%     ];

%    else

%     S =[1   1   1   1   1   1;
%         1   1   1   1   1   1;
%         1   1   1   1   1   1;
%         1   1   1   1   1   1;
%     ];

%    end

%    clear t
%    clear yy
     % yyy(i,:)=xx0';
     % [t, yy]=ode23('cformation', [tt(i) tt(i+1)], xx0); %, 0.001);
   %xx0=yy(length(t), :)';

%   yyy(i,:)=xyz0';

%   [t, yy]=ode23('cformation3d', [tt(i) tt(i+1)], xyz0); %, 0.001);
%   xyz0=yy(length(t), :)';
%end

%c=fix(clock);
%fprintf('end at %2i.%2i.%2i\n\n', ...
%   c(4),c(5),c(6));

%for i=1:400
%    for l=1:16
%        tmp(i,l)=0;
%    end
%end

%figure
%for i=1:Nagent

        %plot3(yyy(:,(i-1)*4+1), yyy(:,(i-1)*4+3), tmp(:,i));

%    plot3(yyy(:,(i-1)*6+1), yyy(:,(i-1)*6+3), yyy(:,(i-1)*6+5));
%    axis square
%    axis([-5 15 -5 15 -25 -5]);
%    grid on
%    hold on
%end

%for i=1:Nagent

    % initial position
%    rectangle('Curvature',[1 1], 'Position', [yyy(1,(i-1)*4+1)-0.1 yyy(1,(i-1)*4+3)-0.1 0.2 0.2]);

    % final position
%    rectangle('Curvature',[1 1], 'Position', [yyy(1000,(i-1)*4+1)-0.2 yyy(1000,(i-1)*4+3)-0.2 0.4 0.4

end
k=92;
%    rectangle('FaceColor','b','Curvature',[1 1], 'Position', [yyy(k,1)-0.3 yyy(k,3)-0.3 0.6 0.6]);
%    rectangle('FaceColor','y','Curvature',[1 1], 'Position', [yyy(k,5)-0.3 yyy(k,7)-0.3 0.6 0.6]);
%    rectangle('FaceColor','g','Curvature',[1 1], 'Position', [yyy(k,9)-0.3 yyy(k,11)-0.3 0.6 0.6]);

%    rectangle('FaceColor','r','Curvature',[1 1], 'Position', [yyy(k,13)-0.3 yyy(k,15)-0.3 0.6 0.6]);
%    rectangle('FaceColor','r','Curvature',[1 1], 'Position', [yyy(k,17)-0.3 yyy(k,19)-0.3 0.6 0.6]);
%    rectangle('FaceColor','r','Curvature',[1 1], 'Position', [yyy(k,21)-0.3 yyy(k,23)-0.3 0.6 0.6]);

%    rectangle('FaceColor','y','Curvature',[1 1], 'Position', [yyy(k,25)-0.3 yyy(k,27)-0.3 0.6 0.6]);
%    rectangle('FaceColor','y','Curvature',[1 1], 'Position', [yyy(k,29)-0.3 yyy(k,31)-0.3 0.6 0.6]);
%    rectangle('FaceColor','y','Curvature',[1 1], 'Position', [yyy(k,33)-0.3 yyy(k,35)-0.3 0.6 0.6]);
%    rectangle('FaceColor','y','Curvature',[1 1], 'Position', [yyy(k,37)-0.3 yyy(k,39)-0.3 0.6 0.6]);
%    rectangle('FaceColor','b','Curvature',[1 1], 'Position', [yyy(k,1)-0.3 yyy(k,3)-0.3 0.6 0.6]);
%    rectangle('FaceColor','y','Curvature',[1 1], 'Position', [yyy(k,5)-0.3 yyy(k,7)-0.3 0.6 0.6]);
%    rectangle('FaceColor','g','Curvature',[1 1], 'Position', [yyy(k,9)-0.3 yyy(k,11)-0.3 0.6 0.6]);

%    rectangle('FaceColor','r','Curvature',[1 1], 'Position', [yyy(k,13)-0.3 yyy(k,15)-0.3 0.6 0.6]);

% To generate the animation file
%axis square
%fig=figure;
%set(fig,'DoubleBuffer','on');
%rect = get(fig,'Position');
%rect(1:2) = [0 0];
%mov = VideoWriter('formation.avi','compression','indeo5');
%writerObj = VideoWriter('formation.avi');
%open(writerObj);
%for k=1:length(ttt)

%           for i=1:4
%plot3(yyy(1:k,(i-1)*4+1), yyy(1:k,(i-1)*4+3), tmp(1:k,i));
%    plot3(yyy(1:k,(i-1)*6+1), yyy(1:k,(i-1)*6+3), yyy(1:k,(i-1)*6+5));
%    hold on
%    end

%    h1 = rectangle('FaceColor','b','Curvature',[1 1], 'Position', [yyy(k,1)-0.3 yyy(k,3)-0.3 0.6 0.6]
%    h2 = rectangle('FaceColor','y','Curvature',[1 1], 'Position', [yyy(k,5)-0.3 yyy(k,7)-0.3 0.6 0.6]
%    h3 = rectangle('FaceColor','g','Curvature',[1 1], 'Position', [yyy(k,9)-0.3 yyy(k,11)-0.3 0.6 0.6

%    h5 = rectangle('FaceColor','r','Curvature',[1 1], 'Position', [yyy(k,13)-0.3 yyy(k,15)-0.3 0.6 0.
%    h6 = rectangle('FaceColor','r','Curvature',[1 1], 'Position', [yyy(k,17)-0.3 yyy(k,19)-0.3 0.6 0.
%    h7 = rectangle('FaceColor','r','Curvature',[1 1], 'Position', [yyy(k,21)-0.3 yyy(k,23)-0.3 0.6 0.

%    h8 = rectangle('FaceColor','y','Curvature',[1 1], 'Position', [yyy(k,25)-0.3 yyy(k,27)-0.3 0.6 0.
%    h9 = rectangle('FaceColor','y','Curvature',[1 1], 'Position', [yyy(k,29)-0.3 yyy(k,31)-0.3 0.6 0.
%    h10 = rectangle('FaceColor','y','Curvature',[1 1], 'Position', [yyy(k,33)-0.3 yyy(k,35)-0.3 0.6 0
%    h11 = rectangle('FaceColor','y','Curvature',[1 1], 'Position', [yyy(k,37)-0.3 yyy(k,39)-0.3 0.6 0

%    h1 = rectangle('FaceColor','b','Curvature',[1 1], 'Position', [yyy(k,1)-0.3 yyy(k,3)-0.3 0.6 0.6]
%    h2 = rectangle('FaceColor','y','Curvature',[1 1], 'Position', [yyy(k,5)-0.3 yyy(k,7)-0.3 0.6 0.6]
%    h3 = rectangle('FaceColor','g','Curvature',[1 1], 'Position', [yyy(k,9)-0.3 yyy(k,11)-0.3 0.6 0.6

%    h5 = rectangle('FaceColor','r','Curvature',[1 1], 'Position', [yyy(k,13)-0.3 yyy(k,15)-0.3 0.6 0.

    %axis([-25 20 -25 20 -25 0]);
%    grid on
%    axis square
%    frame = getframe(fig, rect);
%    mov = addframe(mov,F);
%    writeVideo(writerObj,frame);
%end
%close(writerObj);
```

A.2 Partial C++ Code for Ground Vehicles

```
/*
        Program to connect four robots
        formation.
        Created By:
        9/22/13
*/

#include "Aria.h"
#include <cmath>
#include <iostream>
#include <fstream>

//constant
#define PI  3.14159
#define KV  0.6666

using namespace std;

int sgn(double d)
       {
           if(d<0)  return -1;
           else if (d==0)    return 0;
           else    return  1;
        }

int main(int argc, char** argv)
{
int ret;
std::string str;
int i = 1;
int Vx = 0, Vy=0;
int KV1 = 0, KV2 = 0, KV3 = 0, KV4 = 0;
int r1_x = 0, r1_y= 0, r2_x = 0, r2_y = 0, r3_x = 0, r3_y = 0, r4_x = 0, r4_y = 0;
//User needs to specify the location of each robot relative to robot 1.
//For robot 1, just use the values (0,0) since it is the robot the others will reference
//each other off of
int robot1_location_x, robot1_location_y;
int robot2_location_x, robot2_location_y;
int robot3_location_x, robot3_location_y;
int robot4_location_x, robot4_location_y;

cout << "-----------------" << endl;
cout << "Input x and y coordinates of robot 1" << endl;
cout << "Robot 1 x: ";
cin >> robot1_location_x;
cout << "Robot 1 y: ";
cin >> robot1_location_y;
cout << "-----------------" << endl;
cout << "Input x and y coordinates of robot 2" << endl;
cout << "Robot 2 x: ";
cin >> robot2_location_x;
cout << "Robot 2 y: ";
cin >> robot2_location_y;
cout << "-----------------" << endl;
cout << "Input x and y coordinates of robot 3" << endl;
cout << "Robot 3 x: ";
cin >> robot3_location_x;
cout << "Robot 3 y: ";
cin >> robot3_location_y;
cout << "-----------------" << endl;
cout << "Input x and y coordinates of robot 4" << endl;
cout << "Robot 4 x: ";
cin >> robot4_location_x;
cout << "Robot 4 y: ";
cin >> robot4_location_y;
cout << "-----------------" << endl;

//get hostnames and port numbers
ArArgumentParser argParser(&argc, argv);
char* host1 = argParser.checkParameterArgument("-rh1");
if(!host1) host1 = "localhost";
```

```
char* host2 = argParser.checkParameterArgument("-rh2");
if(!host2) host2 = "localhost";
char* host3 = argParser.checkParameterArgument("-rh3");
if(!host3) host3 = "localhost";
char* host4 = argParser.checkParameterArgument("-rh4");
if(!host4) host4 = "localhost";

int port1 = 8101;
int port2 = 8101;
int port3 = 8101;
int port4 = 8101;

//if same host, it must be using two ports
if(strcmp(host1, host2) == 0)
                port2++;

if(strcmp(host1, host3) == 0)
                port3 = port3 + 2;

if(strcmp(host1, host4) == 0)
                port4 = port4 + 3;

bool
argSet = false;
argParser.checkParameterArgumentInteger("-rp1", &port1);
if(!argSet) argParser.checkParameterArgumentInteger("-rrtp1", &port1);

argSet = false;
argParser.checkParameterArgumentInteger("-rp2", &port2);
if(!argSet) argParser.checkParameterArgumentInteger("-rrtp2", &port2);

argSet = false;
argParser.checkParameterArgumentInteger("-rp3", &port3);
if(!argSet) argParser.checkParameterArgumentInteger("-rrtp3", &port3);

argSet = false;
argParser.checkParameterArgumentInteger("-rp4", &port4);
if(!argSet) argParser.checkParameterArgumentInteger("-rrtp4", &port4);

//add the key handler to aria
ArKeyHandler keyHandler;
Aria::setKeyHandler(&keyHandler);

//First robot variables
ArTcpConnection con1;
ArRobot robot1;

//Second robot variables
ArTcpConnection con2;
ArRobot robot2;
```

```
//Third robot variables
ArTcpConnection con3;
ArRobot robot3;

//Fourth robot variables
ArTcpConnection con4;
ArRobot robot4;

//Attach the key handler to a robot now
robot1.attachKeyHandler(&keyHandler);
robot2.attachKeyHandler(&keyHandler);
robot3.attachKeyHandler(&keyHandler);
robot4.attachKeyHandler(&keyHandler);

//start up Aria stuff
Aria::init();

/*===================================*/
//Start up robot 1
ArLog::log(ArLog::Normal, "Connecting to first robot at %s:%d...", host1, port1);
if ((ret = con1.open(host1, port1)) != 0)
{
        str = con1.getOpenMessage(ret);
        printf("Open failed to robot 1: %s\n", str.c_str());
        Aria::exit(1);
        return 1;
}

robot1.setDeviceConnection(&con1);

if(!robot1.blockingConnect())
{
        printf("Could not connect to robot 1...abort\n");
        Aria::exit(1);
        return 1;
}

//turn on motors, turn off sounds
robot1.comInt(ArCommands::ENABLE, 1);
robot1.comInt(ArCommands::SOUNDTOG, 0);

/*===================================*/
//Start up robot 2
ArLog::log(ArLog::Normal, "Connecting to second robot");
if ((ret = con2.open(host2, port2)) != 0)
{
        str = con2.getOpenMessage(ret);
        printf("Open failed to robot 2: %s\n", str.c_str());
        Aria::exit(1);
        return 1;
}
```

```
robot2.setDeviceConnection(&con2);

if(!robot2.blockingConnect())
{
        printf("Could not connect to robot 2...abort\n");
        Aria::exit(1);
        return 1;
}
printf("Turning on motors for robot 2\n");
//turn on motors, turn off sound
robot2.comInt(ArCommands::ENABLE, 1);
robot2.comInt(ArCommands::SOUNDTOG, 0);

/*==================================*/
//Start up robot 3

ArLog::log(ArLog::Normal, "Connecting to third robot");
if ((ret = con3.open(host3, port3)) != 0)
{
        str = con3.getOpenMessage(ret);
        printf("Open failed to robot 3: %s\n", str.c_str());
        Aria::exit(1);
        return 1;
}

robot3.setDeviceConnection(&con3);

if(!robot3.blockingConnect())
{
        printf("Could not connect to robot 3...abort\n");
        Aria::exit(1);
        return 1;
}

printf("Turning on motors for robot 3\n");
//turn on motors, turn off sound
robot3.comInt(ArCommands::ENABLE, 1);
robot3.comInt(ArCommands::SOUNDTOG, 0);

/*==================================*/
//Start up robot 4
ArLog::log(ArLog::Normal, "Connecting to second robot");
if ((ret = con4.open(host4, port4)) != 0)
{
        str = con4.getOpenMessage(ret);
        printf("Open failed to robot 4: %s\n", str.c_str());
        Aria::exit(1);
        return 1;
}
```

```
robot4.setDeviceConnection(&con4);

if(!robot4.blockingConnect())
{
        printf("Could not connect to robot 4...abort\n");
        Aria::exit(1);
        return 1;
}
printf("Turning on motors for robot 4\n");
//turn on motors, turn off sound
robot4.comInt(ArCommands::ENABLE, 1);
robot4.comInt(ArCommands::SOUNDTOG, 0);

/*==================================*/

//setup vars
double robot1_x, robot1_y;      // x/y position
double robot2_x, robot2_y;
double robot3_x, robot3_y;
double robot4_x, robot4_y;

double robot1_th, robot2_th, robot3_th, robot4_th; // angular orientation
double robot1_radius, robot2_radius, robot3_radius, robot4_radius;      //robots radius

double robot1_x_hat, robot1_y_hat;      //parameters
double robot2_x_hat, robot2_y_hat;
double robot3_x_hat, robot3_y_hat;
double robot4_x_hat, robot4_y_hat;

double robot1_control_1, robot1_control_2;
double robot2_control_1, robot2_control_2;
double robot3_control_1, robot3_control_2;
double robot4_control_1, robot4_control_2;

double robot1_velocity, robot1_rotation_velocity;
double robot2_velocity, robot2_rotation_velocity;
double robot3_velocity, robot3_rotation_velocity;
double robot4_velocity, robot4_rotation_velocity;

robot1_radius = robot1.getRobotRadius();
robot2_radius = robot2.getRobotRadius();
robot3_radius = robot3.getRobotRadius();
robot4_radius = robot4.getRobotRadius();

//run robots on background threads
robot1.runAsync(true);
robot2.runAsync(true);
robot3.runAsync(true);
robot4.runAsync(true);
printf("Running robots\n");
```

```
ofstream in;
in.open("r1.txt",ios::trunc);

while(1)
{
/*      if (i++<100000000) //follow slant line format to a squre and then a line SOFTWARE SIMULATION
        r1_x = 707, r1_y= -707, r2_x = -707, r2_y = 707, r3_x = 707, r3_y = 707, r4_x = 0, r4_y = 1414
else
        r1_x = -1414, r1_y= -1414, r2_x = 707, r2_y = 707, r3_x = 1414, r3_y = 1414, r4_x = 2121, r4_y
*/
/*      if (i++<40000000) //follow slant line format to a squre and then a line SOFTWARE SIMULATION
        r1_x = -609, r1_y= 0, r2_x = -609, r2_y = 0, r3_x = -609, r3_y = 0, r4_x = 1827, r4_y = 0, Vx=
else if(i++>=40000000 && i++<80000000)
        r1_x = -800, r1_y= 0, r2_x = -800, r2_y = 0, r3_x = -800, r3_y = 0, r4_x = 2400, r4_y = 0, Vy=
else
        r1_x = 0, r1_y= -800, r2_x = -800, r2_y = 800, r3_x = 0, r3_y = -800, r4_x = 800, r4_y = 800,
*/
/*      if (i++<5000000) //follow slant line format to a squre and then a line SOFTWARE SIMULATION
        r1_x = -609, r1_y= 0, r2_x = -609, r2_y = 0, r3_x = -609, r3_y = 0, r4_x = 1827, r4_y = 0, Vx=
else if(i++>=5000000 && i++<90000000)
        r1_x = -609, r1_y= 0, r2_x = -609, r2_y = 0, r3_x = -609, r3_y = 0, r4_x = 1827, r4_y = 0, Vy=
else if(i++>=90000000 && i++<147000000)
        r1_x = 0, r1_y= -609, r2_x = -609, r2_y = 609, r3_x = 0, r3_y = -609, r4_x = 609, r4_y = 609,
else
        r1_x = 0, r1_y= 0, r2_x = 0, r2_y = 0, r3_x = 0, r3_y = 0, r4_x = 0, r4_y = 0, Vy=0, Vx=0, KV1
*/
//      r1_x = -609, r1_y= 0, r2_x = -609, r2_y = 0, r3_x = -609, r3_y = 0, r4_x = 1827, r4_y = 0, KV1

//      r1_x = 0, r1_y= -800, r2_x = -800, r2_y = 0, r3_x = 0, r3_y = 800, r4_x = 800, r4_y = 0;// squ

//      r1_x = 0, r1_y= -609, r2_x = -609, r2_y = 0, r3_x = 0, r3_y = 609, r4_x = 609, r4_y = 0;// squ

//      r1_x = 0, r1_y= 609, r2_x = -609, r2_y = 0, r3_x = 0, r3_y = -609, r4_x = 609, r4_y = 0;// squ

//      r1_x = 0, r1_y= 0, r2_x = 0, r2_y = 0, r3_x = 0, r3_y = 0, r4_x = 0, r4_y = 0;

if (i++<50000000) //follow slant line format to a squre and then a line SOFTWARE SIMULATION
        r1_x = -1500, r1_y= 0, r2_x = -1500, r2_y = 0, r3_x = -1500, r3_y = 0, r4_x = 4500, r4_y = 0,
else if(i++>=500000000 && i++<1470000000)
        r1_x = 0, r1_y= -1500, r2_x = -1500, r2_y = 0, r3_x = 0, r3_y = 1500, r4_x = 1500, r4_y = 1500
else
        r1_x = 0, r1_y= 0, r2_x = 0, r2_y = 0, r3_x = 0, r3_y = 0, r4_x = 0, r4_y = 0, Vy=0, Vx=0, KV1

//get robot initial position
        robot1_x = robot1.getX() + robot1_location_x;
        robot1_y = robot1.getY() + robot1_location_y;

        robot2_x = robot2.getX() + robot2_location_x;
        robot2_y = robot2.getY() + robot2_location_y;

        robot3_x = robot3.getX() + robot3_location_x;
        robot3_y = robot3.getY() + robot3_location_y;

        robot4_x = robot4.getX() + robot4_location_x;
        robot4_y = robot4.getY() + robot4_location_y;

        robot1_th = robot1.getTh() * PI/180;
        robot2_th = robot2.getTh() * PI/180;
        robot3_th = robot3.getTh() * PI/180;
        robot4_th = robot4.getTh() * PI/180;

//control logic for robot1
        robot1_x_hat = robot1_x + robot1_radius*cos(robot1_th);
        robot1_y_hat = robot1_y + robot1_radius*sin(robot1_th);

        robot2_x_hat = robot2_x + robot2_radius*cos(robot2_th);
        robot2_y_hat = robot2_y + robot2_radius*sin(robot2_th);

        robot3_x_hat = robot3_x + robot3_radius*cos(robot3_th);
        robot3_y_hat = robot3_y + robot3_radius*sin(robot3_th);

        robot4_x_hat = robot4_x + robot4_radius*cos(robot4_th);
        robot4_y_hat = robot4_y + robot4_radius*sin(robot4_th);

        in<<robot1_x_hat<<"\t"<<robot1_y_hat<<"\t"<<robot2_x_hat<<"\t"<<robot2_y_hat<<"\t"<<robot3_x_h

/***************************************************************************************************
/*//All communications

//Control Logic for robot 1
        robot1_control_1 = KV*(robot2_x_hat - robot1_x_hat + robot3_x_hat - robot1_x_hat + robot4_x_ha
        robot1_control_2 = KV*(robot2_y_hat - robot1_y_hat + robot3_y_hat - robot1_y_hat + robot4_y_ha

        robot1_velocity = robot1_control_1*cos(robot1_th) + robot1_control_2*sin(robot1_th);
        robot1_rotation_velocity = -(robot1_control_1/robot1_radius)*sin(robot1_th) + (robot1_control_

//Control logic for robot 2
        robot2_control_1 = KV*(robot3_x_hat - robot2_x_hat + robot4_x_hat - robot2_x_hat + robot1_x_ha
        robot2_control_2 = KV*(robot3_y_hat - robot2_y_hat + robot4_y_hat - robot2_y_hat + robot1_y_ha

        robot2_velocity = robot2_control_1*cos(robot2_th) + robot2_control_2*sin(robot2_th);
        robot2_rotation_velocity = -(robot2_control_1/robot2_radius)*sin(robot2_th) + (robot2_control_

//Control Logic for robot 3
        robot3_control_1 = KV*(robot4_x_hat - robot3_x_hat + robot2_x_hat - robot3_x_hat + robot1_x_ha
        robot3_control_2 = KV*(robot4_y_hat - robot3_y_hat + robot2_y_hat - robot3_y_hat + robot1_y_ha

        robot3_velocity = robot3_control_1*cos(robot3_th) + robot3_control_2*sin(robot3_th);
        robot3_rotation_velocity = -(robot3_control_1/robot3_radius)*sin(robot3_th) + (robot3_control_

//Control Logic for robot 4
        robot4_control_1 = KV*(robot1_x_hat - robot4_x_hat + robot2_x_hat - robot4_x_hat + robot3_x_ha
        robot4_control_2 = KV*(robot1_y_hat - robot4_y_hat + robot2_y_hat - robot4_y_hat + robot3_y_ha

        robot4_velocity = robot4_control_1*cos(robot4_th) + robot4_control_2*sin(robot4_th);
        robot4_rotation_velocity = -(robot4_control_1/robot4_radius)*sin(robot4_th) + (robot4_control_
*/
/*****************************************************************************************************

/*      //Control Logic for robot 1 leader follow
        //robot1_control_1 = KV*(robot2_x_hat - robot1_x_hat + r1_x) + 200;     //(500/sqrt(2.0))) + 2
        //robot1_control_2 = KV*(robot2_y_hat - robot1_y_hat + r1_y) +200;
        robot1_control_1 = 200;
        robot1_control_2 = 200;

        robot1_velocity = robot1_control_1*cos(robot1_th) + robot1_control_2*sin(robot1_th);
        robot1_rotation_velocity = -(robot1_control_1/robot1_radius)*sin(robot1_th) + (robot1_control_

        //Control logic for robot 2
        robot2_control_1 = KV*(robot1_x_hat - robot2_x_hat + r2_x) + 200; //(1000/sqrt(2.0))) + 200;
        robot2_control_2 = KV*(robot1_y_hat - robot2_y_hat + r2_y) +200;                //(1000/sqrt(2

        robot2_velocity = robot2_control_1*cos(robot2_th) + robot2_control_2*sin(robot2_th);
        robot2_rotation_velocity = -(robot2_control_1/robot2_radius)*sin(robot2_th) + (robot2_control_

//Control Logic for robot 3
        robot3_control_1 = KV*(robot1_x_hat - robot3_x_hat + r3_x) + 200;               //(1500/sqrt(2
        robot3_control_2 = KV*(robot1_y_hat - robot3_y_hat + r3_y) +200;                        //(500

        robot3_velocity = robot3_control_1*cos(robot3_th) + robot3_control_2*sin(robot3_th);
        robot3_rotation_velocity = -(robot3_control_1/robot3_radius)*sin(robot3_th) + (robot3_control_

//Control Logic for robot 4
        robot4_control_1 = KV*(robot1_x_hat - robot4_x_hat + r4_x) + 200;               //(1500/sqrt(2
        robot4_control_2 = KV*(robot1_y_hat - robot4_y_hat + r4_y) +200;                        //(500

        robot4_velocity = robot4_control_1*cos(robot4_th) + robot4_control_2*sin(robot4_th);
        robot4_rotation_velocity = -(robot4_control_1/robot4_radius)*sin(robot4_th) + (robot4_control_

/*****************************************************************************************************

/*      //Control Logic for robot 1 leader sgn function KV=100
        robot1_control_1 = KV1*(sgn(robot2_x_hat - robot1_x_hat + r1_x)) + 100; //(500/sqrt(2.0))) + 2
        robot1_control_2 = KV1*(sgn(robot2_y_hat - robot1_y_hat + r1_y)) ;

        robot1_velocity = robot1_control_1*cos(robot1_th) + robot1_control_2*sin(robot1_th);
        robot1_rotation_velocity = -(robot1_control_1/robot1_radius)*sin(robot1_th) + (robot1_control_

//Control logic for robot 2
        robot2_control_1 = KV2*(sgn(robot3_x_hat - robot2_x_hat + r2_x)) + 100; //(1000/sqrt(2.0))) +
        robot2_control_2 = KV2*(sgn(robot3_y_hat - robot2_y_hat + r2_y)) ;              //(1000/sqrt(2

        robot2_velocity = robot2_control_1*cos(robot2_th) + robot2_control_2*sin(robot2_th);
        robot2_rotation_velocity = -(robot2_control_1/robot2_radius)*sin(robot2_th) + (robot2_control_

//Control Logic for robot 3
        robot3_control_1 = KV3*(sgn(robot4_x_hat - robot3_x_hat + r3_x)) + 100;         //(1500/sqrt(2
        robot3_control_2 = KV3*(sgn(robot4_y_hat - robot3_y_hat + r3_y)) ;                      //(500

        robot3_velocity = robot3_control_1*cos(robot3_th) + robot3_control_2*sin(robot3_th);
        robot3_rotation_velocity = -(robot3_control_1/robot3_radius)*sin(robot3_th) + (robot3_control_

//Control Logic for robot 4
        robot4_control_1 = KV4*(sgn(robot1_x_hat - robot4_x_hat + r4_x)) + 100;         //(1500/sqrt(2
        robot4_control_2 = KV4*(sgn(robot1_y_hat - robot4_y_hat + r4_y)) ;                      //(500

        robot4_velocity = robot4_control_1*cos(robot4_th) + robot4_control_2*sin(robot4_th);
        robot4_rotation_velocity = -(robot4_control_1/robot4_radius)*sin(robot4_th) + (robot4_control_
*/
/****************************************************************************************************
/*
//All communication control Logic for consensus to a point, let KV=3000 initial
        robot1_control_1 = KV1*sgn(robot2_x_hat - robot1_x_hat + robot3_x_hat - robot1_x_hat + robot4_
        robot1_control_2 = KV1*sgn(robot2_y_hat - robot1_y_hat + robot3_y_hat - robot1_y_hat + robot4_

        robot1_velocity = robot1_control_1*cos(robot1_th) + robot1_control_2*sin(robot1_th);
        robot1_rotation_velocity = -(robot1_control_1/robot1_radius)*sin(robot1_th) + (robot1_control_

//Control logic for robot 2
        robot2_control_1 = KV2*sgn(robot3_x_hat - robot2_x_hat + robot4_x_hat - robot2_x_hat + robot1_
        robot2_control_2 = KV2*sgn(robot3_y_hat - robot2_y_hat + robot4_y_hat - robot2_y_hat + robot1_

        robot2_velocity = robot2_control_1*cos(robot2_th) + robot2_control_2*sin(robot2_th);
        robot2_rotation_velocity = -(robot2_control_1/robot2_radius)*sin(robot2_th) + (robot2_control_

//Control Logic for robot 3
        robot3_control_1 = KV3*sgn(robot4_x_hat - robot3_x_hat + robot2_x_hat - robot3_x_hat + robot1_
        robot3_control_2 = KV3*sgn(robot4_y_hat - robot3_y_hat + robot2_y_hat - robot3_y_hat + robot1_

        robot3_velocity = robot3_control_1*cos(robot3_th) + robot3_control_2*sin(robot3_th);
        robot3_rotation_velocity = -(robot3_control_1/robot3_radius)*sin(robot3_th) + (robot3_control_

//Control Logic for robot 4
        robot4_control_1 = KV4*sgn(robot1_x_hat - robot4_x_hat + robot2_x_hat - robot4_x_hat + robot3_
        robot4_control_2 = KV4*sgn(robot1_y_hat - robot4_y_hat + robot2_y_hat - robot4_y_hat + robot3_

        robot4_velocity = robot4_control_1*cos(robot4_th) + robot4_control_2*sin(robot4_th);
        robot4_rotation_velocity = -(robot4_control_1/robot4_radius)*sin(robot4_th) + (robot4_control_
*/
/****************************************************************************************************
/*
//One line communication for consensus to a point function KV=3000,
        robot1_control_1 = KV1*(sgn(robot2_x_hat - robot1_x_hat + r1_x)) +v;    //(500/sqrt(2.0))) + 2
        robot1_control_2 = KV1*(sgn(robot2_y_hat - robot1_y_hat + r1_y)) ;
        robot1_velocity = robot1_control_1*cos(robot1_th) + robot1_control_2*sin(robot1_th);
        robot1_rotation_velocity = -(robot1_control_1/robot1_radius)*sin(robot1_th) + (robot1_control_

//Control logic for robot 2
        robot2_control_1 = KV2*(sgn(robot3_x_hat - robot2_x_hat + r2_x)) +v; //(1000/sqrt(2.0))) + 200
        robot2_control_2 = KV2*(sgn(robot3_y_hat - robot2_y_hat + r2_y)) ;             //(1000/sqrt(2

        robot2_velocity = robot2_control_1*cos(robot2_th) + robot2_control_2*sin(robot2_th);
        robot2_rotation_velocity = -(robot2_control_1/robot2_radius)*sin(robot2_th) + (robot2_control_

//Control Logic for robot 3
        robot3_control_1 = KV3*(sgn(robot4_x_hat - robot3_x_hat + r3_x)) +v;            //(1500/sqrt(2
        robot3_control_2 = KV3*(sgn(robot4_y_hat - robot3_y_hat + r3_y)) ;                    //(500

        robot3_velocity = robot3_control_1*cos(robot3_th) + robot3_control_2*sin(robot3_th);
        robot3_rotation_velocity = -(robot3_control_1/robot3_radius)*sin(robot3_th) + (robot3_control_

//Control Logic for robot 4
        robot4_control_1 = KV4*(sgn(robot1_x_hat - robot4_x_hat + r4_x)) +v;            //(1500/sqrt(2
        robot4_control_2 = KV4*(sgn(robot1_y_hat - robot4_y_hat + r4_y)) ;                    //(500

        robot4_velocity = robot4_control_1*cos(robot4_th) + robot4_control_2*sin(robot4_th);
        robot4_rotation_velocity = -(robot4_control_1/robot4_radius)*sin(robot4_th) + (robot4_control_
*/
/*****************************************************************************************************

/*      //One line communication for changing  (0,0) (0,609) (609,0) (609,609)
        robot1_control_1 = KV1*(sgn(robot2_x_hat - robot1_x_hat + r1_x)) +Vx;   //(500/sqrt(2.0))) + 2
        robot1_control_2 = KV1*(sgn(robot2_y_hat - robot1_y_hat + r1_y)) +Vy ;  //(500/sqrt(2.0))) + 2

        robot1_velocity = robot1_control_1*cos(robot1_th) + robot1_control_2*sin(robot1_th);
        robot1_rotation_velocity = -(robot1_control_1/robot1_radius)*sin(robot1_th) + (robot1_control_

//Control logic for robot 2
        robot2_control_1 = KV2*(sgn(robot3_x_hat - robot2_x_hat + r2_x)) +Vx; //(1000/sqrt(2.0))) + 20
        robot2_control_2 = KV2*(sgn(robot3_y_hat - robot2_y_hat + r2_y)) +Vy;           //(1000/sqrt(2

        robot2_velocity = robot2_control_1*cos(robot2_th) + robot2_control_2*sin(robot2_th);
        robot2_rotation_velocity = -(robot2_control_1/robot2_radius)*sin(robot2_th) + (robot2_control_

//Control Logic for robot 3
        robot3_control_1 = KV3*(sgn(robot4_x_hat - robot3_x_hat + r3_x)) +Vx;           //(1500/sqrt(2
        robot3_control_2 = KV3*(sgn(robot4_y_hat - robot3_y_hat + r3_y)) +Vy;                 //(500

        robot3_velocity = robot3_control_1*cos(robot3_th) + robot3_control_2*sin(robot3_th);
        robot3_rotation_velocity = -(robot3_control_1/robot3_radius)*sin(robot3_th) + (robot3_control_

//Control Logic for robot 4
        robot4_control_1 = KV4*(sgn(robot1_x_hat - robot4_x_hat + r4_x)) +Vx;           //(1500/sqrt(2
        robot4_control_2 = KV4*(sgn(robot1_y_hat - robot4_y_hat + r4_y)) +Vy;                 //(500
        robot4_velocity = robot4_control_1*cos(robot4_th) + robot4_control_2*sin(robot4_th);
        robot4_rotation_velocity = -(robot4_control_1/robot4_radius)*sin(robot4_th) + (robot4_control_
*/
/*************************************************************************************************
//One line communication for changing  (0,0) (0,609) (609,0) (609,609)
        robot1_control_1 = KV1*(sgn(robot2_x_hat - robot1_x_hat + r1_x)) +Vx;   //(500/sqrt(2.0))) + 2
        robot1_control_2 = KV1*(sgn(robot2_y_hat - robot1_y_hat + r1_y)) +Vy //(500/sqrt(2.0))) + 200;

        robot1_velocity = robot1_control_1*cos(robot1_th) + robot1_control_2*sin(robot1_th);
        robot1_rotation_velocity = -(robot1_control_1/robot1_radius)*sin(robot1_th) + (robot1_control_

//Control logic for robot 2
        robot2_control_1 = KV2*(sgn(robot3_x_hat - robot2_x_hat + r2_x)) +Vx; //(1000/sqrt(2.0))) + 20
        robot2_control_2 = KV2*(sgn(robot3_y_hat - robot2_y_hat + r2_y)) +Vy;          //(1000/sqrt(2

        robot2_velocity = robot2_control_1*cos(robot2_th) + robot2_control_2*sin(robot2_th);
        robot2_rotation_velocity = -(robot2_control_1/robot2_radius)*sin(robot2_th) + (robot2_control_

//Control Logic for robot 3
        robot3_control_1 = KV3*(sgn(robot4_x_hat - robot3_x_hat + r3_x)) +Vx;          //(1500/sqrt(2
        robot3_control_2 = KV3*(sgn(robot4_y_hat - robot3_y_hat + r3_y)) +Vy;                  //(500

        robot3_velocity = robot3_control_1*cos(robot3_th) + robot3_control_2*sin(robot3_th);
        robot3_rotation_velocity = -(robot3_control_1/robot3_radius)*sin(robot3_th) + (robot3_control_

//Control Logic for robot 4
        robot4_control_1 = KV4*(sgn(robot1_x_hat - robot4_x_hat + r4_x)) +Vx;          //(1500/sqrt(2
        robot4_control_2 = KV4*(sgn(robot1_y_hat - robot4_y_hat + r4_y)) +Vy;                  //(500

        robot4_velocity = robot4_control_1*cos(robot4_th) + robot4_control_2*sin(robot4_th);
        robot4_rotation_velocity = -(robot4_control_1/robot4_radius)*sin(robot4_th) + (robot4_control_

/*************************************************************************************************
//set velocity and rotational velocity on robots
        robot1.setVel(robot1_velocity);
        robot1.setRotVel(robot1_rotation_velocity);

        robot2.setVel(robot2_velocity);
        robot2.setRotVel(robot2_rotation_velocity);

        robot3.setVel(robot3_velocity);
        robot3.setRotVel(robot3_rotation_velocity);

        robot4.setVel(robot4_velocity);
        robot4.setRotVel(robot4_rotation_velocity);

        }
        in.close();
        Aria::shutdown();
}
```

Appendix B

Appendix: Photos and Tables

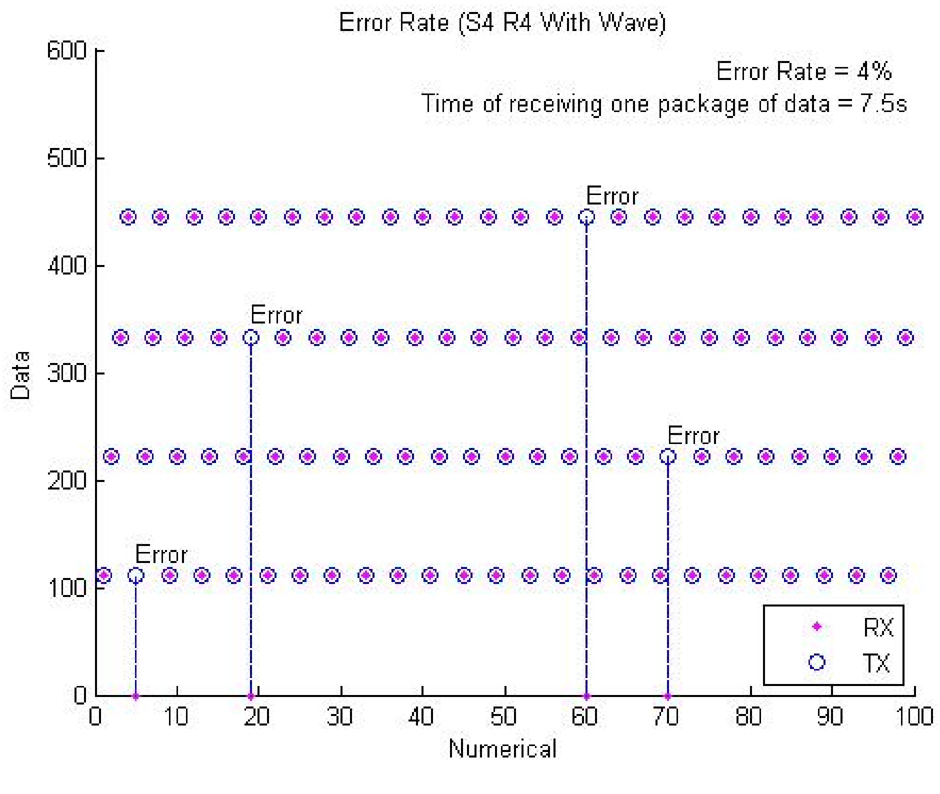

FIGURE B.1: Sonar communication bit error rate without Sign function 1
R4 is the sonar receive speed by 5 bits per second, and S4 is the transmit speed by 5 bits per second. Wave condition means a wave maker to generate the wave environment.

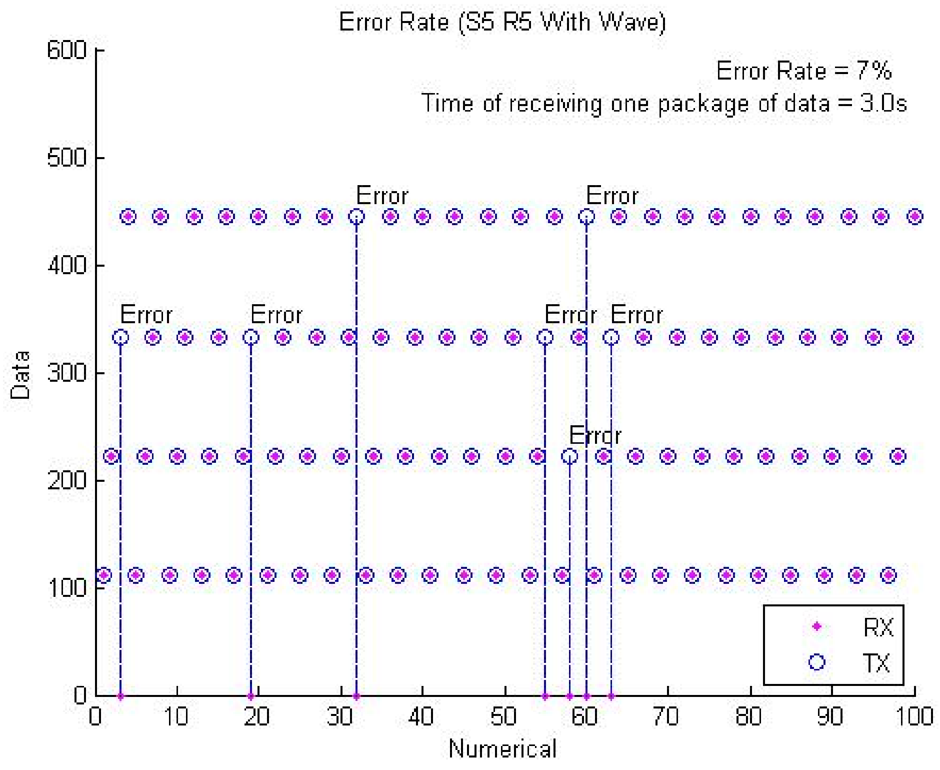

FIGURE B.2: Sonar communication bit error rate without Sign function 2
R5 is the sonar receive speed by 13 bits per second, and S5 is the transmit speed by 13 bits per second. Wave condition means a wave maker to generate the wave environment.

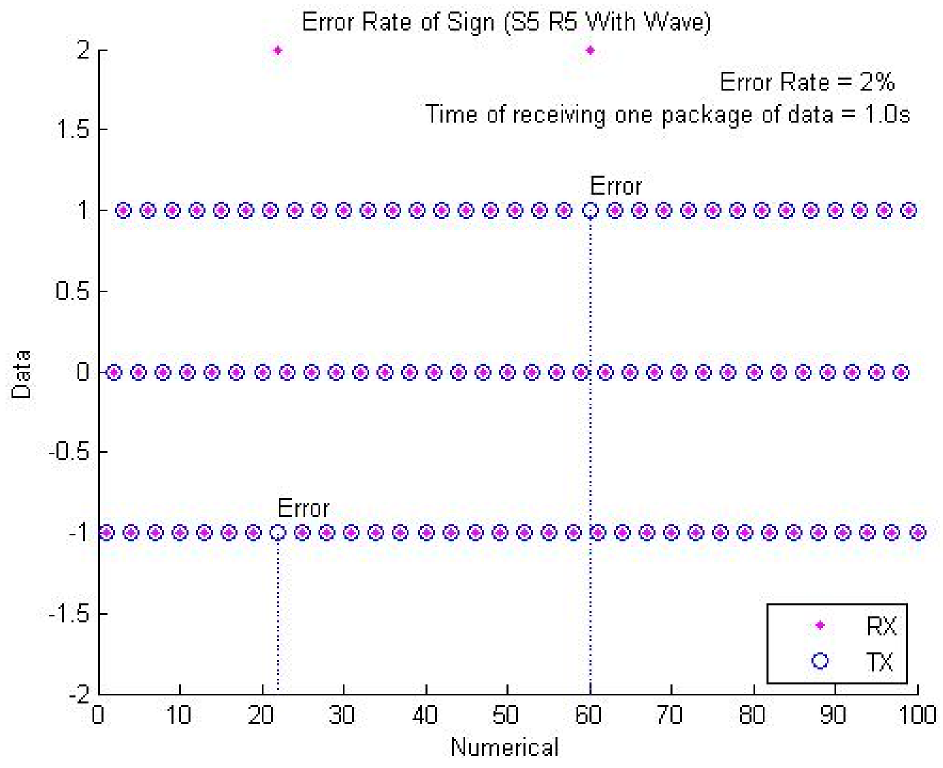

FIGURE B.3: Sonar communication bit error rate with Sign function 3
R5 is the sonar receive speed by 13 bits per second, and S5 is the transmit speed by 13 bits per second. Wave condition means a wave maker to generate the wave environment.

Appendix C

Appendix: Websites and Media Links

1. News Journal: http://www.news-journalonline.com/article/20140903/NEWS/140909807/1040?p=3&tc=pg

2. News Journal: http://www.news-journalonline.com/article/20140912/news/140919749

3. Free News Pos: http://www.freenewspos.com/en/home-news-article/a/1205281/today/embry-riddle-teams-with-nasa-for-underwater-mission

Bibliography

[1] J. Wang, M. Obeng, Z. Qu, T. Yang, G. Staskevich and B. Abbe, Discontinuous cooperative control for consensus of multiagent systems with switching topologies and time-delays. Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on, , 0743-1546, 2013.

[2] J.Wang, M. Obeng, Distributed formation control for nonholonomic mobile robots. Sensor Array and Multichannel Signal Processing Workshop (SAM), 2012 IEEE 7th, 1551-2282, 2012.

[3] Z. Lin, B. Francis, and M. Maggiore, State agreement for continuous time coupled nonlinear systmes, SIAM Journal on Control and Optimization, vlo. 49, pp. 288-307, 2007.

[4] J. Wang, Z. Qu, C.M. Ihlefeld, and R.A. Hull, A control design based solution to robotic ecology: Autonomy of achieving cooperative behavior from a high-level astronaut command, Autonomous Robots, vol. 20, pp. 97-112, 2006.

[5] R. O. Saber, J. A. Fax, and R. M. Murray, Consensus and cooperation in networked multiagent systems, Proceedings of the IEEE, vol. 95, pp. 215-233, 2007.

[6] Q. Hui, Finite time rendezvous algorithms for mobile autonomous agents, IEEE Trans. Automa. Control , vol. 56, pp. 1213-1223, 2008.

[7] B. Paden and S. Sastry, A calculus for computing filippov's differential inclusion with application to the variable structure control of robot mainpulators, IEEE Transactions on Circuits and systems, vol. 34, pp. 73-82, 1987.

[8] N. Moshtagh and A. Jadbabaie. Distributed geodesic control laws for flocking of nonholonomic agents. IEEE Transactions on Automatic Control, 52:681-686, 2007.

[9] Z. Qu, Cooperative control of dynamical systems. Springer-Verlag, London, 2009.

[10] M. Arcak. Passivity as a design tool for group coordination. IEEE Trans. Automat. Contr., 52:1380-1389, 2007.

[11] J. Wang, Z. Qu, and M. Obeng. A distributed cooperative steering control with application to nonholonomic robots. In 49th IEEE conference on Dec. and Ctrl, pages 4571-4576, Atlanta, GA, Dec 2010.

[12] W. Ren and R. W. Beard. Distributed consensus in multi-vehicle cooperative control. Springer-Verlag, London, 2008.

[13] A. Jadbabaie, J. Lin, and A.S. Morse. Coordination of groups of vehicle formations. IEEE transactions on automatic control. 49: 1465-1476, 2004

[14] F. Bullo, J. Cortes, and S. Martinez, Distributed control robotic networks, ser. Applied Mathematics Series. Princeton University Press, 2009, electronically available at http://coordinationbllk.info.

[15] Z. Butler and D. Rus, "Controlling mobile sensors for monitoring events with coverage constrains," in Proceedings of IEEE International Conference of Robotics and Automation, New Orleans, LA, April 2004, pp. 1563-1573

[16] M. Schwager, J. McLukrin, and D. Rus, "Distributed coverage control with sensory feedback for networked robots," in Proceedings of Robotics: Science and Systems, Philadelphia, PA, August 2006.

[17] D.V. Dimarogonas and K. J. Kyriakopoulos, "Connectedness preserving distributed swarm aggregation for multiple kinematic robots." IEEE Transactions on Robotics, vol. 24, pp. 1213-1223, 2008.

[18] Z. Lin, M. Brouchke, and B. Francis, "Local control strategies for groups of mobile autonomous agents using nearest neighbor rules," IEEE Trans. on Automatic Control, vol.622-629, 2004.

[19] W. Wang and J.J.E. Slotine, "A theoretical study of different leader roles in networks," IEEE Transactions on Automatic Control, vol. 51, no.7, pp. 1156-1161, July 2006.

[20] G. T. Sibley, M. H. Rahimi, and G. S. Sukhatme, ”Robomote: A tiny mobile robot platform for large-scale sensor networks,” in Proceedings of IEEE International Conference on Robotics and Automation(ICRA), 2002.

[21] R. Olfati-Saber and R. R. Murray,”Consensus problems in networks of agents with switching topology and time-delays,” IEEE Transactions on Automatic Control, vol. 48, no.9, pp. 1520-1533, September 2004.

[22] W. H. Wang, X, Q. Chen, et al. Design of Low-Cost Unmanned Underwater Vehicle for Shallow Waters.

[23] B. Benson, Y. Li, R. Kastner, B.Faunce, K. Domond, D. Kimball, C. Schurgers, Design of a Low-Cost, Underwater Acoustic Modem for Short-Range Sensor Networks,

[24] Zaihan Jiang, Underwater Acoustic Networks Issues and Solutions, INTERNATIONAL JOURNAL OF INTELLIGENT CONTROL AND SYSTEMS, VOL. 13, NO. 3, SEPTEMBER 2008,152-161.

[25] Joseph Baker, Optimization of an Autonomous Underwater Vehicle, Project Number: IIH-0002, WPI, 2011.

[26] Ethem M. Sozer, Milica Stojanovic, and John G. Proakis, Life Fellow, IEEE JOURNAL OF OCEANIC ENGINEERING, VOL. 25, NO. 1, JANUARY 2000.

[27] Mihaela Cartei, Energy-efficient scheduling and hybrid communication architecture for underwater littoral surveillance, Computer Communications, www.sciencedirect.com, 2006.