UNIVERSITY OF CALIFORNIA
RIVERSIDE

Experimental Implementation of Multi-Agent Distributed Optimization Algorithms With Shortest Distance to Convex Regions

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

Master of Science

in

Electrical Engineering

by

Hao Wang

December 2016

Thesis Committee:

Dr. Wei Ren, Chairperson
Dr. Ertem Tuncel
Dr. Daniel Wong

The Thesis of Hao Wang is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I would like to express my gratitude to my advisor Professor Wei Ren of Electrical and Computer Engineering Department in University of California Riverside, without whose help, I would not have been here. During my master study in University of California Riverside, I received great support and encouragement from him. His guidance helped me in all the time of research and writing of this thesis.

I would like to thank Ounan Ding for helping me on programming problems. I would like to give my thanks for Chao Zhang, Jianyang Sun, Xuefeng Dong and Zhuofu Deng, spending their spare time in helping me carrying heavy robots from lab to experiment ground during physical experiments and data collection. Thanks also goes out to my labmates in COVEN lab, they gave me many constructive suggestions for my project and thesis. Also I would like to thank Jiahui Lu for sharing his ideas and codes at my bottleneck period.

Last but not the least, I would like to thank my parents for giving birth to me at the first place and supporting me spiritually throughout my life.

ABSTRACT OF THE THESIS

Experimental Implementation of Multi-Agent Distributed Optimization Algorithms With Shortest Distance to Convex Regions

by

Hao Wang

Master of Science, Graduate Program in Electrical Engineering
University of California, Riverside, December 2016
Dr. Wei Ren, Chairperson

The focus of this thesis is on the multi-agent rendezvous problem where the rendezvous location has the total squared shortest distance to certain regions. Three algorithms are experimentally implemented under an undirected communication topology. Simulation plots and experimental results are demonstrated on a multi-robot platform to validate the proposed algorithms. First, an implementation of the distributed subgradient shortest-distance rendezvous algorithm is implemented. Second, an alternative distributed shortest-distance rendezvous algorithm which introduces a kind of diminishing step sizes is validated. Third, an algorithm with employment of two internal states of the dynamic averaging estimator is implemented. This thesis illustrates the three algorithms by presenting simulation plots in Matlab, robots' moving trajectories in the multi-robot platform, and plots of the shortest distance performance function.

Contents

List of Figures

Chapter 1

Introduction

Distributed multi-agent cooperative control has received significant research attention in recent years due to the development of various applications in both military and civilian areas. The booming fields like, unmanned aerial vehicle probe, sweeping robot development, and autonomous driving will all benefit from the exploration of multi-agent cooperative control. It has great potentials in formation control, mobile sensor networks, mapping, localization and so on.

By having a group of agents working together in distributed manner, we can complete certain tasks more easily and efficiently. Common problems in distributed multi-agent cooperative control area include formation, mapping, localization, synchronization. And why distributed control is so important? Because distributed control can achieve global objectives through local interactions without knowing the whole information. For example, a formation of several unmanned aerial vehicles assigned with different tasks need to rendezvous to a center and wait for new assignments. So these aerial vehicles need to find an

optimal rendezvous point with shortest traveling distances in order to save energy. However, there remains some problems to be considered and solved in this field, like communication disturbance elimination, collision avoidance, time-varying communication topology and so on.

1.1 Literature Review

A discussion of distributed control problems is presented in detail in [9] which provides a survey of consensus problems in multi-agent cooperative control area. There are many aspects to be considered in this area. Communication rate is an important factor for distributed coordination of multi-agent networks. The article [6] proves that for a connected network, average consensus can be achieved with an exponential convergence rate based on certain bit information exchange between each connected agent.

The disturbance factor is studied in [5] for the finite-time consensus problem with the employment of double-integrator dynamics. This paper uses Lyapunov theory and graph theory to prove that the states of all agents can reach a consensus in finite time in the absence of disturbances and one example is introduced to verify the efficiency of the proposed algorithm. Similarly, [4] also studies average consensus with noisy channels, random topologies and presents detailed methods. Besides communication disturbance, the path obstacles are considered in [12]. A new optimal control approach is proposed to achieve both multi-agent consensus and obstacle avoidance capability with minimized control efforts. It introduces an innovative nonquadratic obstacle avoidance cost function in distributed scheme.

For the communication topology concept, the author in [8] studies the problem of information consensus among multiple agents with limited and unreliable information exchange with dynamically changing interaction topologies. The article [8] shows that consensus under dynamically changing interaction topologies can be achieved asymptotically if the union of the directed interaction graphs have a spanning tree frequently enough as the system evolves. Theoretical results about consensus seeking under both time-invariant and dynamically changing information exchange topologies are presented. It also introduces applications of consensus protocols to multi-agent coordination. [13] studies some necessary and sufficient conditions for second-order consensus in multi-agent dynamical systems. In these cases, the second-order dynamics are governed by the position and velocity terms and the asymptotic velocity is constant for each agent. This article finally shows that consensus problem can be solved in a multi-agent system whose network topology contains a directed spanning tree if and only if the time delay is less than a critical value. And the conclusion derived in [13] is consistent to the results proposed in [8] as we mentioned before. Furthermore, in application area, [10] presents an elementary introduction of information consensus in multi-vehicle cooperative control. The theories studied in this paper will enable the ability in autonomous vehicles with embedded computational resources.

The article [7] gives the methods of solving multi-agent rendezvous problem where the rendezvous location has the total squared shortest distance to certain regions. Firstly, a distributed subgradient shortest distance rendezvous algorithm is introduced, however the employment of sign function in this algorithm will result in chattering. Secondly, an alternative algorithm is proposed with kind of diminishing step sizes. Thirdly, another

distributed algorithm proves that all agents will rendezvous in finite time while minimizing the distance performance function.

The three algorithms proposed in [7] are implemented and validated in this thesis. The algorithms will be simulated in Matlab and tested on multi-robot experimental platform. Related plots, trajectories and analysis will be shown in following chapters.

1.2 Problem Statement

This paper introduces the situation of finding rendezvous point while minimizing summation of squared distances to it from convex regions. We will implement the distributed shortest-distance algorithms presented in [7] by using a group of ground robots. Here is an example to help understand this problem. We suppose there are four managers doing business in their own areas. For convenience, we set the names of four managers as A, B, C, and D. The four managers are each in charge of one area and responsible for local business affairs. They need to find a rendezvous location which is not far from their respective areas so they can meet up and share business information. Because they are not in same company, B can only talk to A, while A, C and D can communicate with each other as shown in Fig. 1.1. They can only communicate with their neighbors as indicated in this topology and need to find one optimal location to meet up. The article [7] provides three algorithms to solve this problem, and the algorithms are proved in continuous time domain. So we need to convert them into discrete time domain and present the corresponding implementations and validations by our multi-robot experimental platform.

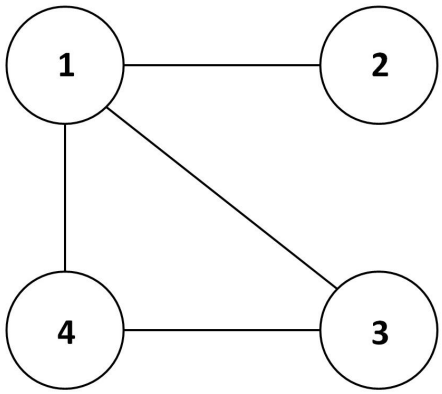

Figure 1.1: Communication topology of agents

1.3 Organization Outline

Chapter 1 gives the related research on multi-agent control area and problem statement of the algorithms to be implemented in this thesis.

Chapter 2 shows the mathematical expression of algorithms derived in [7].

Chapter 3 is about my work and contributions in this research.

Chapter 4 covers the top level diagram, structure of server program and client program and introduces how client program cooperates with server program.

Chapter 5 introduces the multi-robot experimental platform. It also includes related hardware and software used in experiments.

Chapter 6 provides simulation results, experimental results and the trajectories of performance function.

Chapter 7 goes over the conclusion and future work to be improved for the implementations presented in this thesis.

Chapter 2

Main Algorithms Introduction

2.1 Notations and Preliminaries

Suppose there is a multi-agent system consisting of n agents with the continuous-time dynamics

$$\dot{x_i}(t) = u_i(t), \tag{2.1}$$

where $x_i(t)$ is the state of agent i, and $u_i(t)$ is the control input of agent i. And we suppose that a nonempty closed convex set $X_i(t) \in R^m$ is known only to agent i. The agents can only interact with their local neighbors. The objective is to drive all agents to rendezvous while minimizing the distance performance function defined in [7]

$$P_{in}(t) = \frac{1}{2}\sum_{i=1}^{n} \|x_i(t) - P_{X_i}(x_i(t))\|^2. \tag{2.2}$$

In Fig. 2.1 we can have a better understanding of our objective. The red parts denote the agents, and the agents only have the information of the convex regions in the

same dotted squares. Every agent can communicate with other agents connected by solid black straight line. For example, agent 4 could share information with agent 1 and agent 3, but it knows nothing about agent 2. And the black star is the optimal point we desire to drive all agents to rendezvous. No agent knows the optimal point priorly but instead they need to collectively find the optimal point by interaction with their neighbors. The agents need to rendezvous at the optimal location or form a formation with their center located at the optimal location.

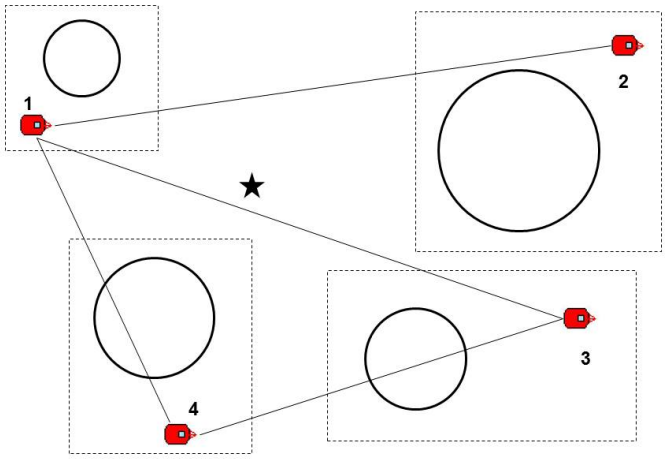

Figure 2.1: Problem statement

Here if agent i is outside the convex region, $P_{X_i}(x_i(t))$ is the projected point from agent i to its convex region X_i. If agent i is inside the convex region, the projected point $P_{X_i}(x_i(t))$ will be $x_i(t)$ itself. To have a better understanding of the following algorithms, the other notations which will be used are introduced below.

- $N_i(t)$ is the neighbor set of agent i at time t.

- n is the number of agents employed in the algorithm.
- $G(t)$ is the undirected agent communication topology graph of order n.
- $P_{X_i}(x_i(t))$ is the state of the projected point from agent x_i to convex region X_i at time t. $\|x_i(t) - P_{X_i}(x_i(t))\|$ is the distance between agent i and the projected point. If agent i is inside the convex region, the term $\|x_i(t) - P_{X_i}(x_i(t))\|$ will be 0.
- g is defined as there exists a constant $g > 0$ such that $\|x\| \leq g$ for any $x \in X_i$.
- X_i is the convex region set which will be assigned to agent i. In the following implementation and validation, we will use circles as convex regions defined as

$$(x-p)^2 + (y-q)^2 = r^2, \tag{2.3}$$

where r is the radius of circles and the center point of the circle is (p, q).

2.2 Algorithms

In this section, we will introduce the mathematical statements of the three algorithms [7] used in the following implementations in detail. Before we go over the algorithms, let's look at a constrained rendezvous problem studied before. In [11], a continuous-time algorithm is proposed for (2.1) as

$$u_i(t) = \alpha \sum_{j \in N_i(t)} (x_j(t) - x_i(t)) + P_{X_i}(x_i(t)) - x_i(t). \tag{2.4}$$

However this approach does not work for the case where the intersection set is empty. And the next three algorithms will remove the assumption that the intersection set

is nonempty.

2.2.1 Distributed Discontinuous Shortest-distance Rendezvous Algorithm

The first algorithm is proposed for (2.1) as

$$u_i(t) = \alpha \sum_{j \in N_i(t)} sgn(x_j(t) - x_i(t)) + \gamma[P_{X_i}(x_i(t)) - x_i(t)]. \tag{2.5}$$

Here $sgn(x)$ is sign function defined as $sgn(x) = 1$ when $x > 0$, $sgn(x) = -1$ when $x < 0$, $sgn(x) = 0$ when $x = 0$, as shown in Fig. 2.2. It is assumed that the graph $G(t)$ is connected and X_i is a nonempty closed convex set.

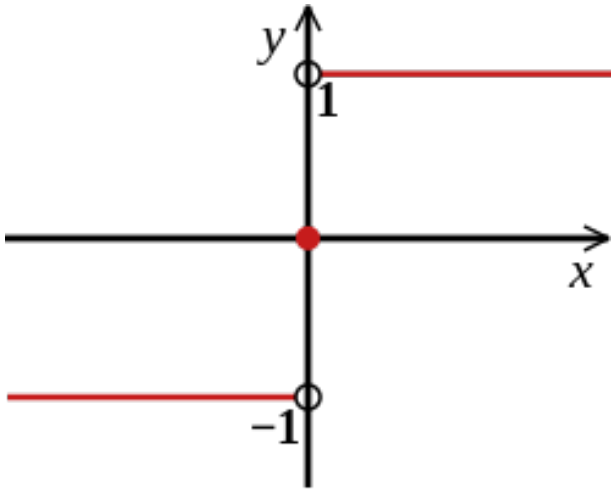

Figure 2.2: Signum function[1]

The algorithm (2.5) requires $\alpha > 0$ and $\gamma > 0$ to be two constants. If $\frac{\alpha}{\gamma} > 2ng$, the first term, $\alpha \sum_{j \in N_i(t)} sgn(x_j(t) - x_i(t))$ will drive all agents to rendezvous in finite time. And the second term, $\gamma[P_{X_i}(x_i(t)) - x_i(t)]$ will help minimize the performance function (2.2) but it also requires the convex regions to be bounded. For brevity, the algorithm (2.5) is

denoted as Algorithm 1 in the following chapters. Because of the introduction of the sign function, the algorithm (2.5) is not continuous.

2.2.2 An Alternative Distributed Shortest-distance Rendezvous Algorithm

In algorithm (2.5), the employment of the sign function will result in chattering. For this situation, an alternative continuous algorithm for (2.1) is presented as

$$u_i(t) = \sum_{j \in N_i(t)} [x_j(t) - x_i(t)] + \frac{1}{\sqrt{t}}[P_{X_i}(x_i(t)) - x_i(t)]. \tag{2.6}$$

The term $\frac{1}{\sqrt{t}}[P_{X_i}(x_i(t)) - x_i(t)]$ requires the convex regions to be bounded. When the graph G(t) is connected and X_i is a nonempty closed convex set and all agents will rendezvous and minimize the performance function (2.2) as $t \to +\infty$. For brevity, the algorithm (2.6) is donated as Algorithm 2 in the following chapters.

2.2.3 Distributed Finite-Time Shortest-distance Rendezvous Algorithm

In subsection 2.2.1 and 2.2.2, the proposed algorithms require that the convex sets be bounded. In this subsection, the assumption that the convex sets be bounded is removed and a distributed finite-time rendezvous algorithm is proposed for (2.1) as

$$\begin{aligned} \dot{\phi}_i(t) &= \alpha_1 \sum_{j \in N_i(t)} sgn(\theta_j(t) - \theta_i(t)) \\ \theta_i(t) &= \phi_i(t) + P_{X_i}(x_i(t)) \\ u_i(t) &= \frac{\alpha_2(\theta_i(t) - x_i(t))}{\|\theta_i(t) - x_i(t)\|}, \end{aligned} \tag{2.7}$$

where $\theta_i(t)$ and $\phi_i(t)$ are the internal states of the dynamic averaging estimator with $\phi_i(0) = 0$ for all i, and $\alpha_1 > 0$ and $\alpha_2 > 0$ are two constants.

It is supposed that the graph $G(t)$ is connected. If $\frac{\alpha_1}{\alpha_2} > 2n$, all agents will rendezvous in finite time while minimizing the performance function (2.2) as t$\rightarrow \infty$. The introduction of normalization in algorithm (2.7) removes the assumption that the convex sets be bounded. For brevity, the algorithm (2.7) is denoted as Algorithm 3 in the following chapters.

The algorithm (2.5), (2.6) and (2.7) will be proved to implemented on the multi-robot experimental platform at University of California Riverside.

Chapter 3

My Work and Contributions

In the research of this thesis, the robot experimental platform is established, server programs and client programs are implemented, theoretical algorithms are converted into practice and the experimental data is analyzed.

3.1 Establishing the Multi-robot Experimental Platform

The Pioneer 3-AT robots are assembled. At the very beginning, there is no operation system on the robot, so the Linux operating system is installed to every robot. The main C++ library used to control the robot is *ARIA*, and the *Aria* library is installed to robot. Every robot is adjusted into correct status and the example programs provided by *Aria* are tested to make sure the robot is in good status.

For example, the computer serial port configuration should always be checked. The default IRQ (interrupt request) for serial port /dev/ttyS0 (used for micro-controller) is 4. Sometimes after the robot on-board computer is installed with a new system, the IRQ

is not consistent to default settings. So all robots' configurations are checked and set to correct status.

3.2 Implementing Related Programs

The client programs and server programs are written for the robots and laptop. The IP addresses and port numbers are assigned to robots and laptop, and the remote communications are established by the network provided by a local router.

3.3 Converting Theory into Practice

The algorithms are converted from theory to practice. They are discretized as following. The control input $u_i(t)$ is replaced with $u_i(kT)$, and agent state $x_i(t)$ is replaced with $x_i(kT)$ in the algorithms. Here, $k = 0, 1, 2, 3, ..., n$ and $T = 100ms$. T is the sampling time applied in the experiments. After the discretization, the the algorithms are simulated in Matlab. The C++ codes are written to implement the discretized algorithms. The picture shown in Fig. 3.1 is a scene from the experiments. After the simulation and experimental record is acquired, the data is analyzed and the simulation trajectories, experimental trajectories and performance function trajectories are plotted in Matlab.

Figure 3.1: A scene from the experiment

Chapter 4

Architecture of Programs Implementation

4.1 Top Level Diagram

Fig. 4.1 shows the top level diagram of C++ programs and robots. Each robot has its own server program and client program. The client program queries the positions of corresponding robots from server programs and then uses the proposed algorithm to calculate the control input for its own robot. The server program receives control input information packet from its client program and relays the control commands to the connected robot. The server program also relays information packet about its robot to client program upon request.

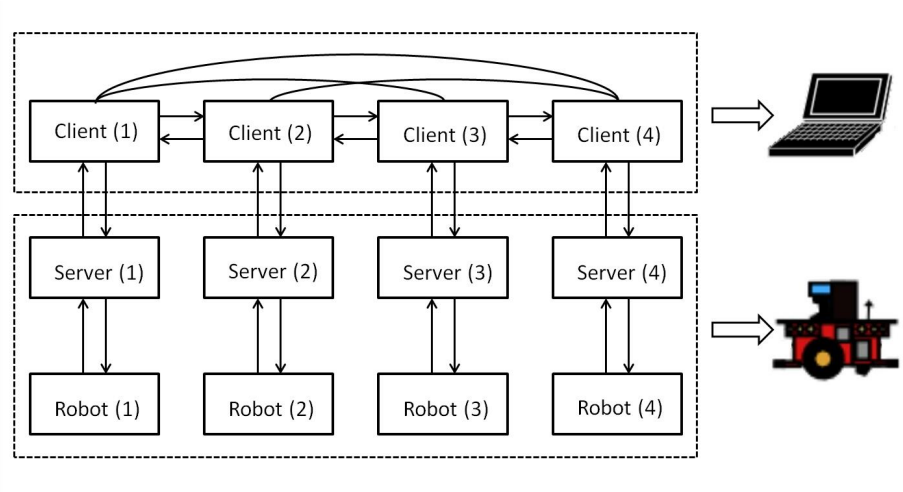

Figure 4.1: Top level diagram of the implementation

4.2 C++ Library Introduction

To implement the server programs and client programs, main C++ library used in this paper is ARIA (Advanced Robot Interface for Applications). ARIA is able to dynamically control robot's velocity, heading, relative heading, and other motion parameters both through simple low-level commands and high-level actions. For the network and communication, ARIA also enables robots to share information packet, position estimates, sonar readings from other robots and computers via TCP/UDP by sub-library ArNetworking.

4.3 Server Program Design

As shown in Fig. 4.1, the server program is used to receive commands from client program, transfer commands to robot micro-controller and send robots current status information packets back to client program.

The core function in server program of transferring commands from client program

to robot micro-controller is *ArActiongoto* function. This function drives robot towards to the next position given by *setGoal*. It travels at the certain speed (mm/sec) given by *getSpeed*. The action stops when it gets a certain distance (determined by *closeDist*) from the goal pose given by *getCloseDist*. This function is easy to control the robot's behaviors and it also ignores the kinetic model of robot. *ArActiongoto* function only regards robot as a particle and drives robot to designated destinations naturally. So the headings, linear velocities, angular velocities and rotations of front and rear wheels should be studied in future work.

To run server program on robot's on-board computer, we need to input and parse program command-line arguments for usage by ARIA functions. Essential arguments needed to be parsed include: host name ($-localhost$), robot TCP port to Aria ($-rrtp$), server port number ($-serverPort$), robot number ($-rnumber$), robot starting location ($-px, -py$) and other default arguments.

Another three critical components in server program are *main* function, $OutputPose$ function and $HandleNewPose$ function. The $OutputPose$ and $HandleNewPose$ function run simultaneously with two parallel threads.

The main function is composed as following parts: Aria initialization, robot object establishing, arguments parsing, gyro establishing, connection from server program to robot micro-controller, sending robot current status to client program and receiving commands from client program. The gyro is introduced to help improve robot's moving trajectories and once it is turned on, it will sense the robot's heading and location changes without reading any data from encoder. The gyro works even when the wheels don't work. The *outputPose*

function broadcast the robot's current location. This function acquires robot's current location from robot micro-controller and sends this information packet via TCP network to the client program with the data named *Pose* for recognition. The $HandleNewpose$ function receives command packets named $NewPose$ from its client program to control the robot.

4.4 Client Program Design

The client program is used to send commands to server program, and receives robots current status information packets from server program. To run client program on the laptop, we need to input and parse program command-line arguments for usage by ARIA functions. Essential arguments needed to be parsed include: robot IP address ($-ipaddress$, like 192.168.1.XX), port number ($-port$), robot number ($-rnumber$) and other default arguments.

The four critical components of the client program are *main* function, $HandlePose$ function, $ProjectionPoint$ function and $SendNewPose$ function. The *main* function is composed of following parts: Aria initialization, arguments parsing, connection from client program to server program, algorithm calculation loop and sending commands to server program. The $HandlePose$ function requests information packet named *Pose* from the network and sends it to algorithm loop to be processed. The $ProjectionPoint$ function is used to calculate the projected point from robot's location to its assigned convex region. Then it transfers the projected point to algorithm calculation loop. The $SendNewPose$ function is the core function in client program. This function includes the algorithm calcu-

lation loop and the part of sending commands to server program. The algorithm calculation loop is designed as following. Firstly, it receives robot current location information packet from $HandlePose$ function and projected point from $ProjectionPoint$ function. Secondly, the control input is calculated in every loop by the proposed multi-agent distributed optimization algorithms and once the value of control input $u_i(t)$ exceeds the threshold, it will be sent to server program via TCP packet and applied on the robot's micro-controller.

The detailed server and client program cooperative processing flow chart is shown in Fig. 4.2. The client program and server program will exit only when all the robots rendezvous to optimal point within the error tolerance.

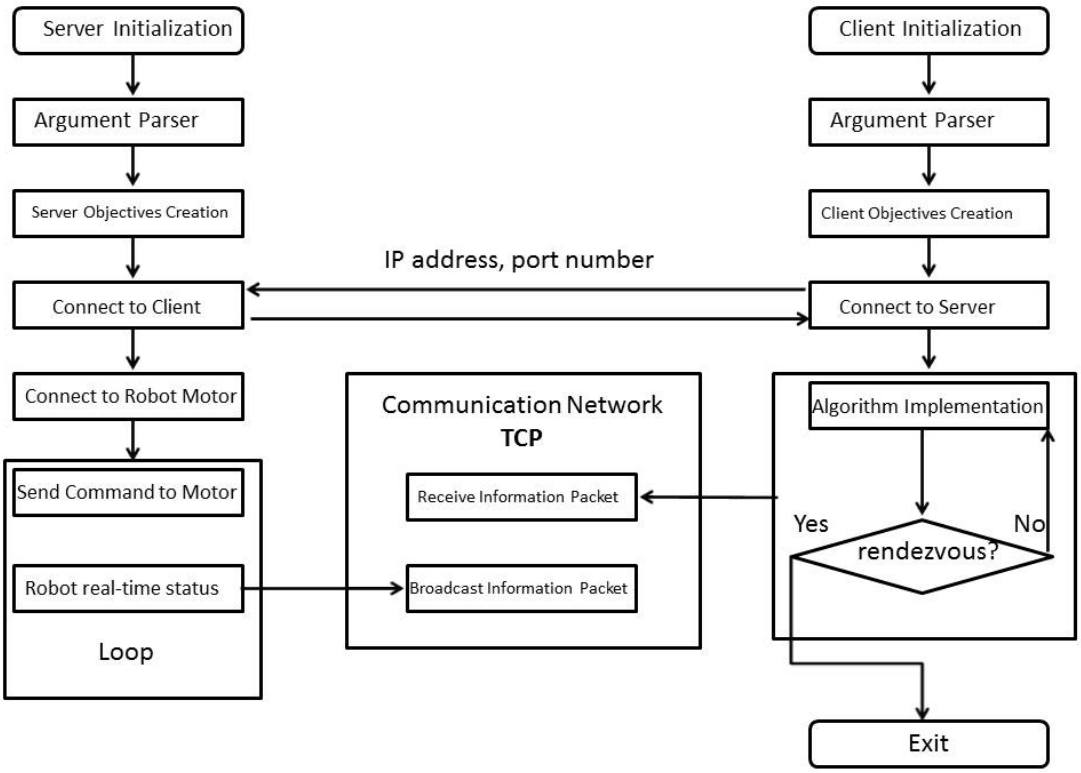

Figure 4.2: Server and client working flow chart

Chapter 5

Multi-robot Experimental Platform

5.1 Hardware

The hardware listed below are used in the implementation and validation of the proposed algorithms.

- Robots - Pioneer 3-AT ground robot is used in the experiment which is a four wheel drive robotic platform as shown in Fig. 5.1. Pioneer 3-AT offers an embedded computer option, opening way for on-board vision processing, Ethernet-based communications, laser, DGPS, and other autonomous functions. The robot's motor is connected to the on-board computer by serial port connection. The communications between robots and laptop are established by TCP (Transmission Control Protocol) and the

Figure 5.1: Multi-robot experimental platform at University of California Riverside

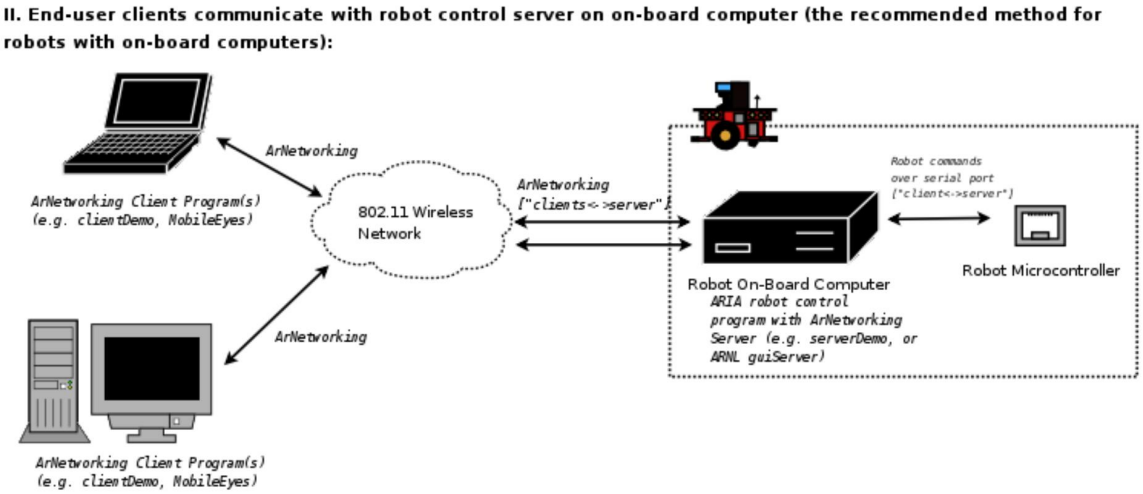

Figure 5.2: Robots and laptop connection option[2]

connection relationship is as shown in Fig. 5.2.

- Robot on-board computer - Pioneer 3AT robot is equipped with a on-board computer: Versalogic "Mamba" EBX-37. The on-board computer is installed with Linux Ubuntu 12.04 operation system.

- WiFi-Router - The WIFI-Router provides the network used for the remote communications between robots and laptop.

- Central Computer - A laptop with Windows 10 operating system is used as the central computer for client programs. The client programs run on separate threads without knowing any other robots' information which are not in the neighbor set defined by communication topology. Each client program calculates its own data and shares data with robots within pre-defined neighborhood. So the central computer acts as four independent computers. There are four terminals in this computer to control the robots' status and displaying the information.

5.2 Software

Here is the list of software used in the experiments. All software are authorized and licensed.

- Operation system - The four ground robots with four server programs running on is installed with Linux Ubuntu 12.04 and the laptop used as the central computer in the experiment runs on Windows 10.

- C++ library - *ARIA* is short for MobileRobots' Advanced Robot Interface. It

also includes ArNetworking library providing remote networking communication via TCP/UDP protocol.

- MobileSim - A simulation software for simulating MobileRobots/ActivMedia platforms and their environments, for debugging and experimentation with ARIA. At the beginning of debugging coding, MobileSim is a good choice to test the codes. MobileSim provides different robot testing options like: robot model, number of robots on separate TCP ports, TCP port number and verbose logging option as shown in Fig. 5.3. It also offers different map options to be loaded. For example, Fig. 5.5 shows the main interface while Mobilesim is working with a map loaded. The red polyhedron is the Pioneer 3-AT robot and the black contours are simulated surroundings and obstacles. But there remains one problem in this software. No matter what initial locations (for example, you want four robots placed as a square shape) are given to the robots, all robots will always form a line shape with same abscissas as shown in Fig. 5.4. Because at this time MobileSim only recognizes the abscissas argument and ignore ordinate argument, and the abscissas argument only determines the starting point for all of the robots in X axis. It does not have separate starting points for each robot in Y axis.

- Visual Studio 2013 - A source code editor developed by Microsoft. It compiles source codes and generates executable files. Batch files are employed to run the executable files together.

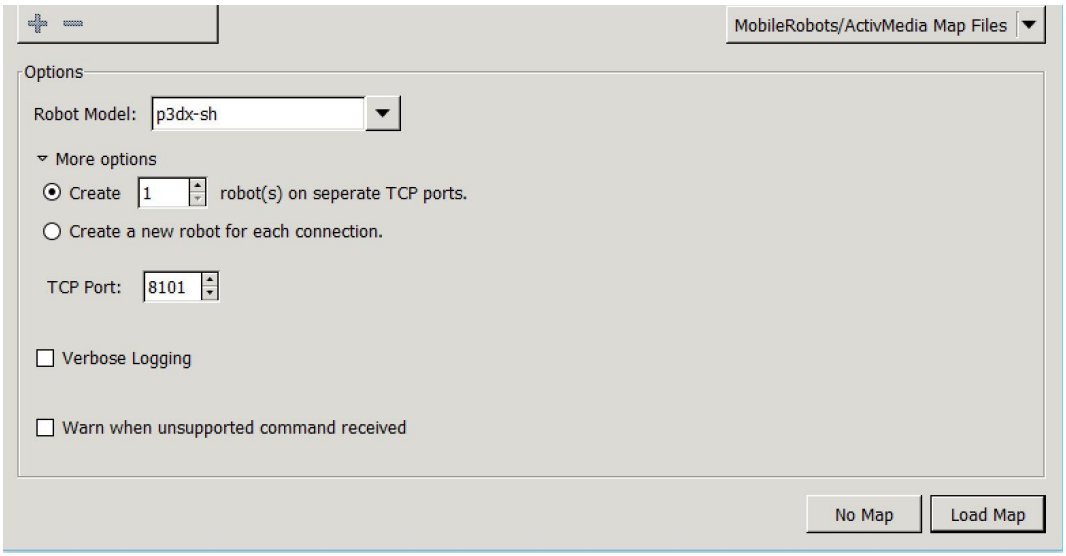

Figure 5.3: MobileSim starting options

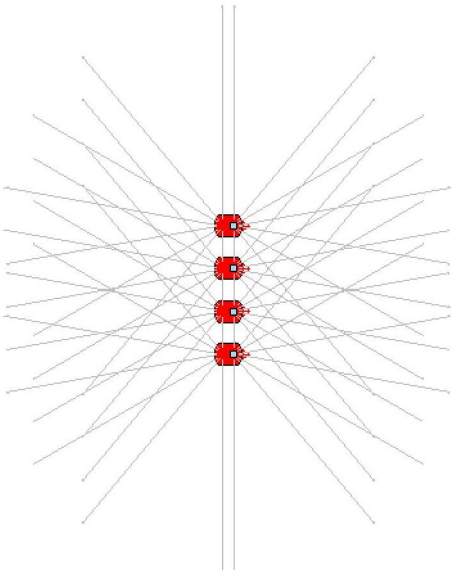

Figure 5.4: Robots starting formation problem

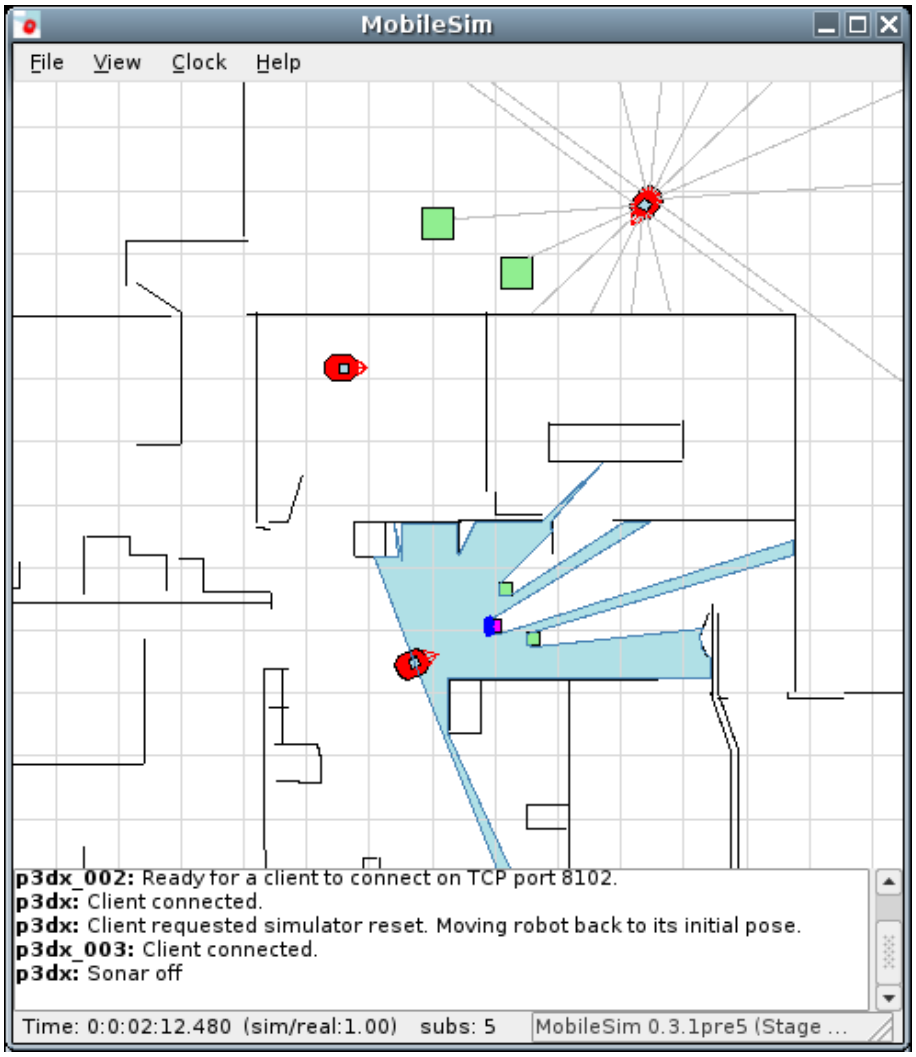

Figure 5.5: MobileSim main interface with map loaded [3]

- MATLAB - We use MATLAB to calculate the optimal rendezvous point and simulate the algorithms in theory before implementing them into C++ programs.

5.3 Auxiliary Programs Running on Pioneer 3-AT robots

The on-board computer is embedded inside the Pioneer 3-AT robot and there is no monitor on the robot. So it is very inconvenient to have the robots turned on and server programs run with the monitor in lab then move all robots to experiment ground. Here a script in Linux is introduced to solve this problem. The script placed in the directory */etc/init/runserver.conf* (*runserver.conf* is the name of the script) will start the server program automatically once the robot is turned on. After experiment, typing *sudo service runserver stop* in terminal will easily shut down the server program. The log files are employed to record the robots' real time locations and they are stored in the directory */var/log/runserver.log* (*runserver.log* is the name of the log file).

Chapter 6

Simulation and Experimental Results

In the implementation, the size of experiment area is 9000 mm $\times$ 7750 mm. It is supposed that a nonempty closed convex set $X_i \in R^m$ is only known to agent i. The corresponding sets in the experiment are composed of four circles. As mentioned before, there exists a constant $g > 0$ such that $\|x\| \leq g$ for all $x \in X_i$, so the constant g in the experiments equals to 9000 mm. Fig. 6.17 shows the experiment area, robots initial positions, convex regions, and optimal rendezvous point. The colored, filled squares are robots, and the circles with same colors are robots' assigned convex regions. The black star is the optimal rendezvous point. In theory, the coordinate of optimal point which has shortest total squared distances to four convex regions is (-439.98, 768.10). There are four robots ($n = 4$) deployed in this area randomly. Initial locations in clockwise are (-4000, 3000), (5000, 4500), (4000, -1500) and (-2000, -3500). The four nonempty closed convex

regions are each centered at (-3000, 4000) with radius 500 mm, centered at (3000, 2500) with radius 1000 mm, centered at (-1500, -2500) with radius 750 mm, and centered at (-2500, -1500) with radius 1000 mm in clockwise.

6.1 Matlab Simulation and Experimental Validation Results

6.1.1 Matlab Simulation Results

The Algorithm 1 requires $\frac{\alpha}{\gamma} > 2ng$. So in this experiment, we have $\alpha = 800$, $\gamma = 0.022$. Fig. 6.1 shows the simulation trajectory generated by Matlab. All agents reach rendezvous in finite time and move to optimal point.

For Algorithm 2, it removes the sign function which could result in chattering. The corresponding simulation trajectory is shown in Fig. 6.2.

The Algorithm 3 requires $\frac{\alpha_1}{\alpha_2} > 2n$. So in this experiment, we have $\alpha_1 = 2000$ and $\alpha_2 = 200$. With the employment of two internal states of the dynamic averaging estimator, we remove the assumption that the convex sets should be bounded. From the simulation trajectory in Fig. 6.3, we can see that the four robots rendezvous in finite time finally.

6.1.2 Experimental Validation Results

Four ground robots' initial locations, final locations and distances to optimal rendezvous point recorded in experiments are shown in Fig. 6.4 (unit: millimeter). The optimal point is (-439.98, 768.10). The trajectories in physical experiments with four ground robots are shown in Fig. 6.5 - Fig. 6.7. All robots finally arrived estimated locations within accept-

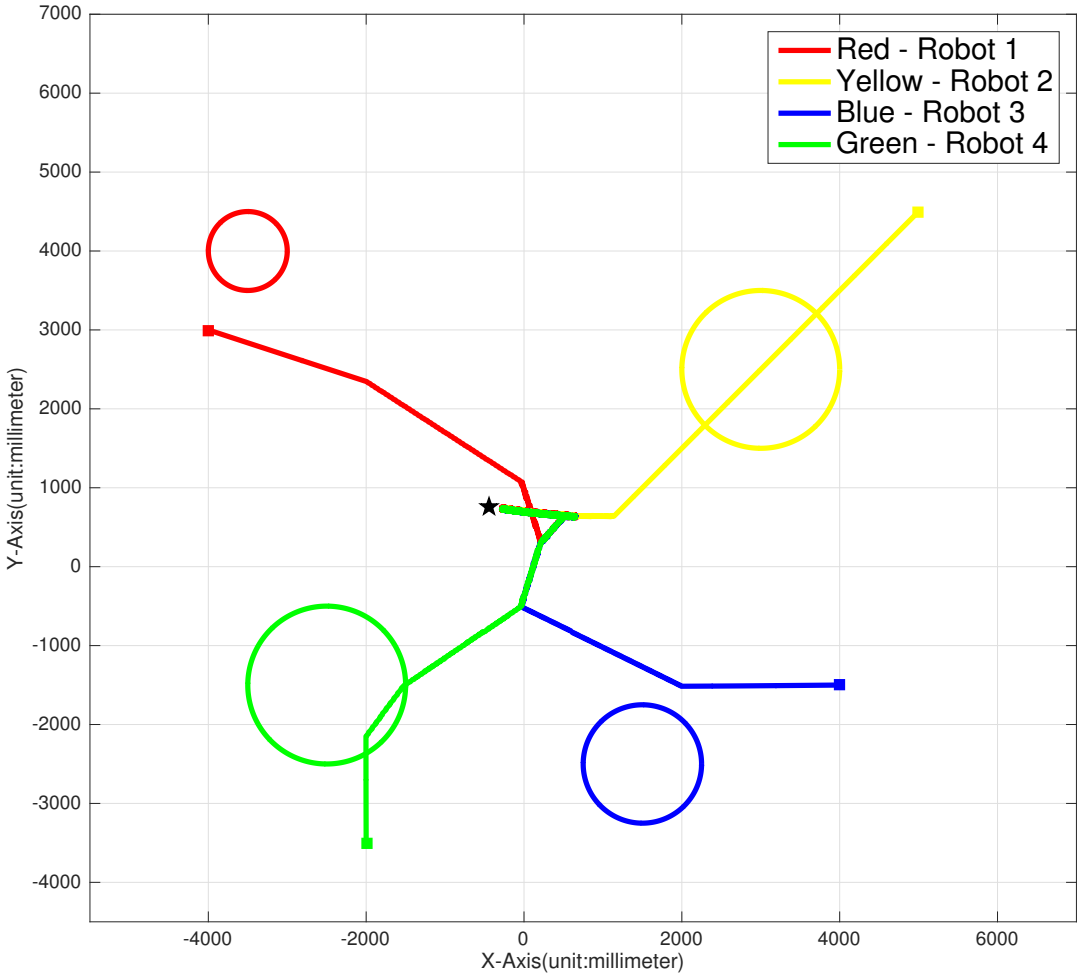

Figure 6.1: Matlab simulation trajectory of Algorithm 1

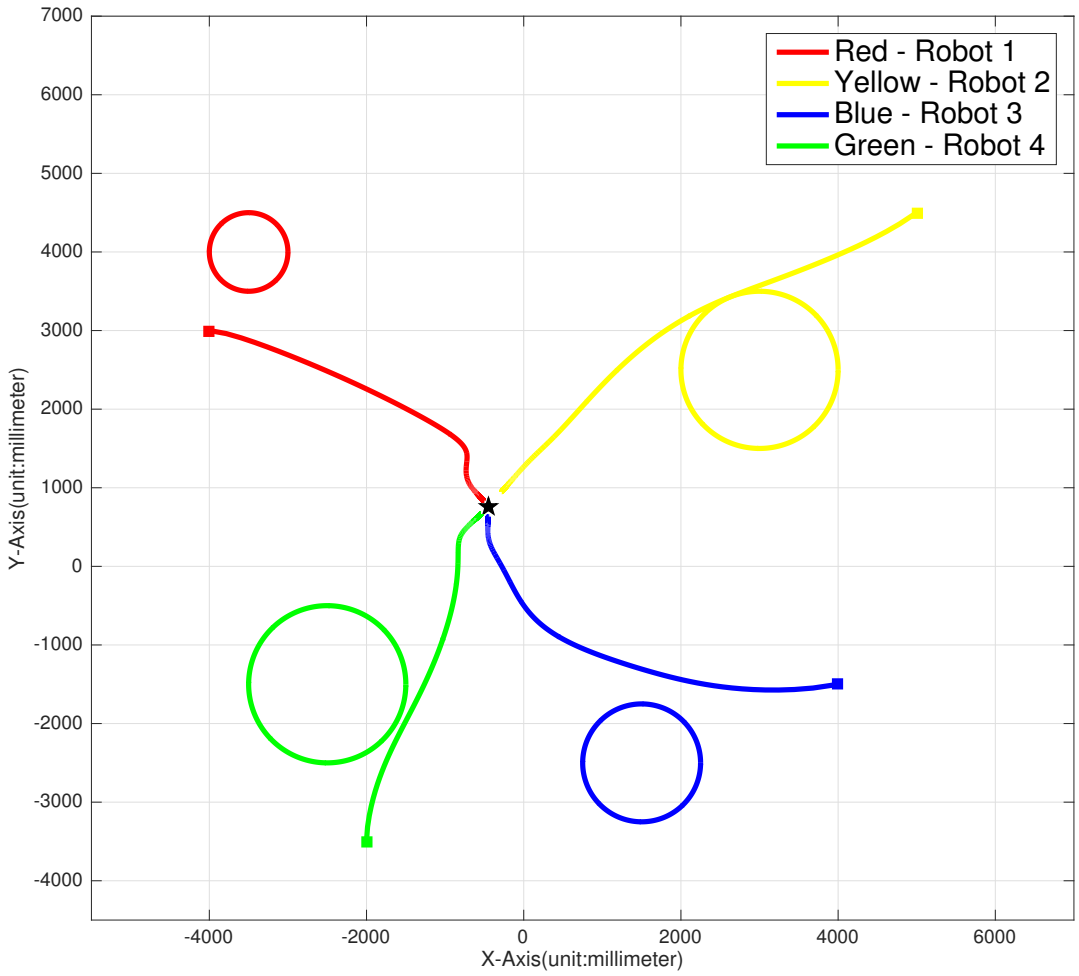

Figure 6.2: Matlab simulation trajectory of Algorithm 2

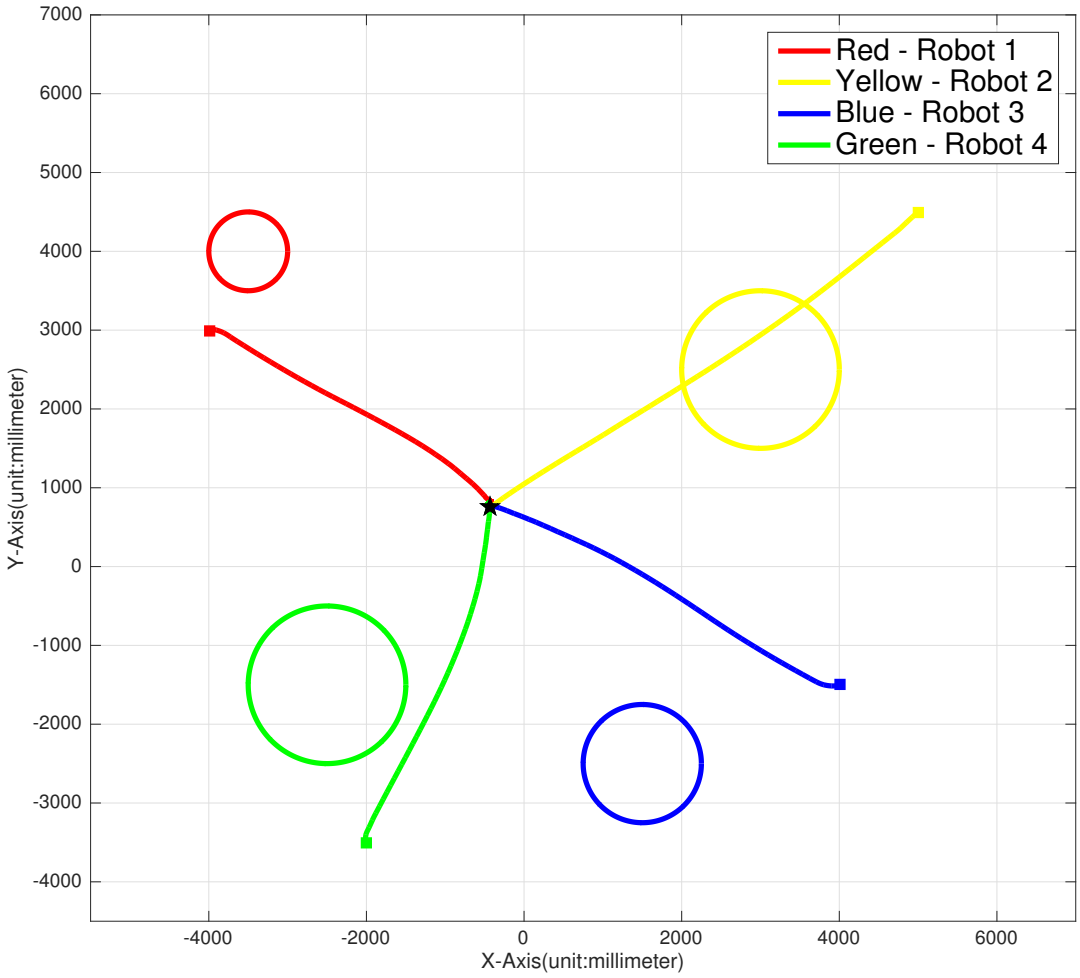

Figure 6.3: Matlab simulation trajectory of Algorithm 3

able errors while minimizing the performance index $P_{in}(t)$. The trajectories are consistent to the plots simulated in Matlab.

As mentioned before, we set an error tolerance $e = 300$ mm. The distance differences between optimal rendezvous point and robots final locations in Algorithm 1 and Algorithm 3 are less than this error. Although the distance difference in the validation of Algorithm 2 exceeds 300 mm, it is still consistent with our theory. Because all robots in Algorithm 2 will rendezvous only when $t \to \infty$. Considering physical experiment limited duration time, algorithm discretization and ground robots deviation, this result is good enough to prove the theory.

Robots starting location

	Algorithm(1)	Algorithm(2)	Algorithm(3)
Robot(1) location	(-4000,3000)	(-4000,3000)	(-4000,3000)
Robot(2) location	(5000,4500)	(5000,4500)	(5000,4500)
Robot(3) location	(4000,-1500)	(4000,-1500)	(4000,-1500)
Robot(4) location	(-2000,-3500)	(-2000,-3500)	(-2000,-3500)

Robots final location

	Algorithm(1)	Algorithm(2)	Algorithm(3)
Robot(1) final location	(-332,754)	(-578,1037)	(-424,905)
Robot(2) final location	(-366,772)	(-218,1003)	(-362,808)
Robot(3) final location	(-352,747)	(-397,462)	(-387,753)
Robot(4) final location	(-374,743)	(-599,502)	(-417,730)

Distance between robots final location to optimal consensus point

	Algorithm(1)	Algorithm(2)	Algorithm(3)
Robot(1) distance	108.90	297.82	137.83
Robot(2) distance	74.08	323.19	87.60
Robot(3) distance	90.47	309.10	55.09
Robot(4) distance	70.59	309.99	44.49

Figure 6.4: Four ground robots' initial, final locations and distances to optimal rendezvous point

6.1.3 Plots of Performance Functions

It can be seen from Fig. 6.8 - Fig. 6.10, the trajectory of performance function in every algorithm converges to optimal value within an acceptable error which satisfies the requirement of minimizing the performance function finally at the end of the experiment.

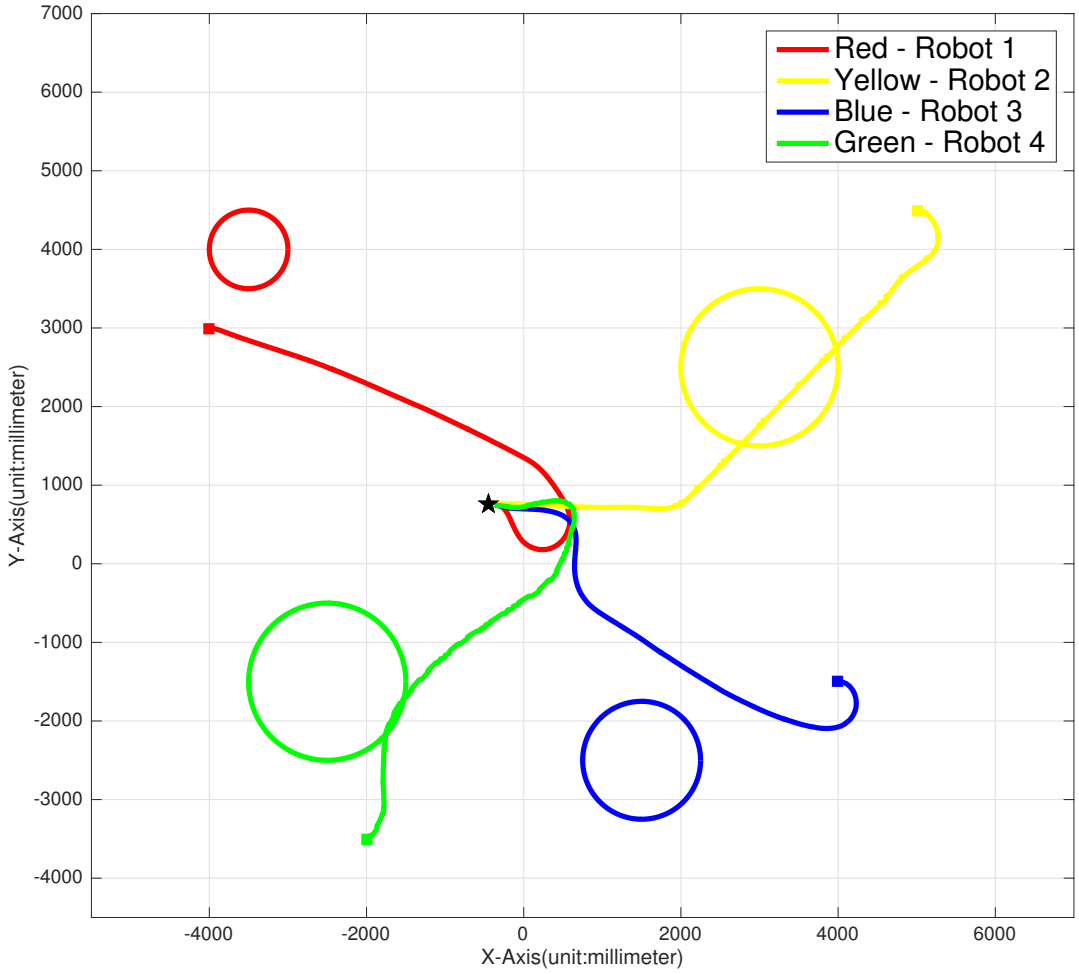

Figure 6.5: The experimental trajectory of robots when Algorithm 1 is used with $\alpha = 800$, $\gamma = 0.022$

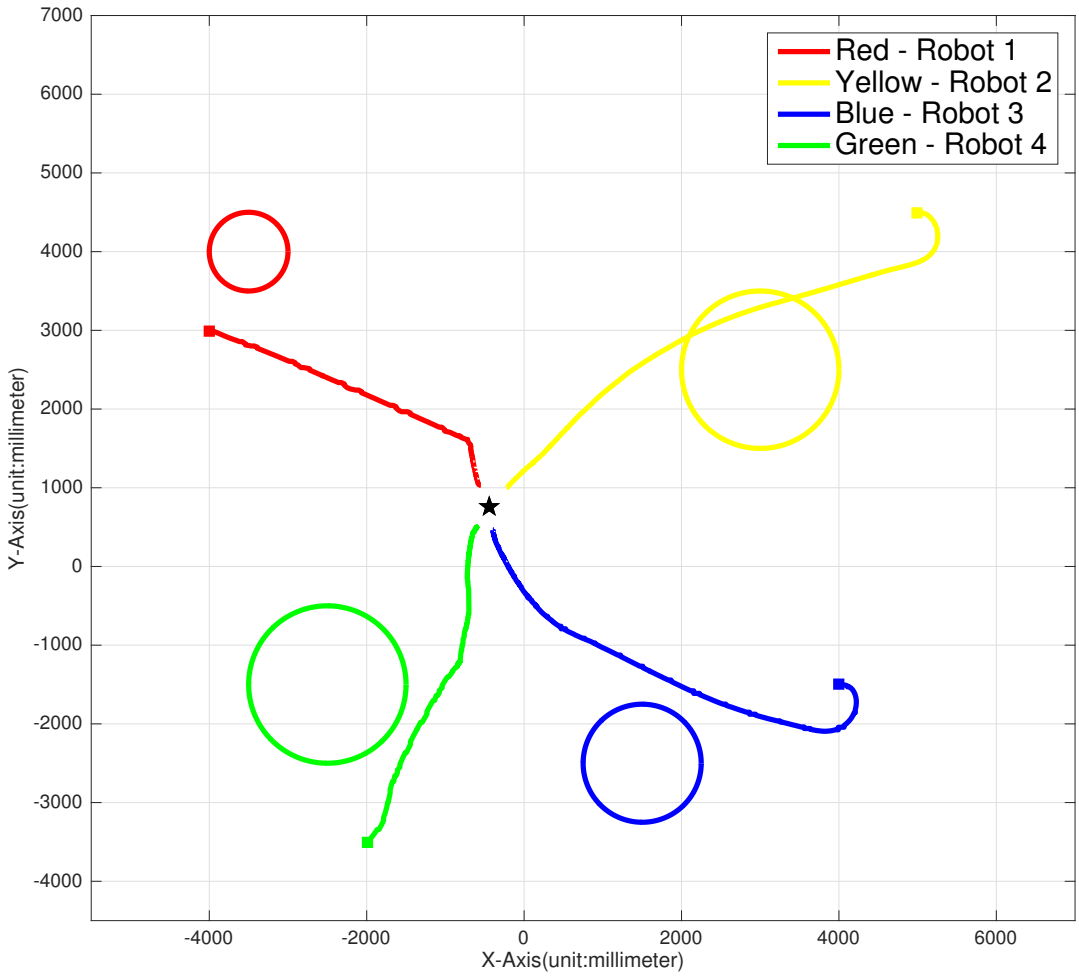

Figure 6.6: The experimental trajectory of robots when Algorithm 2 is used

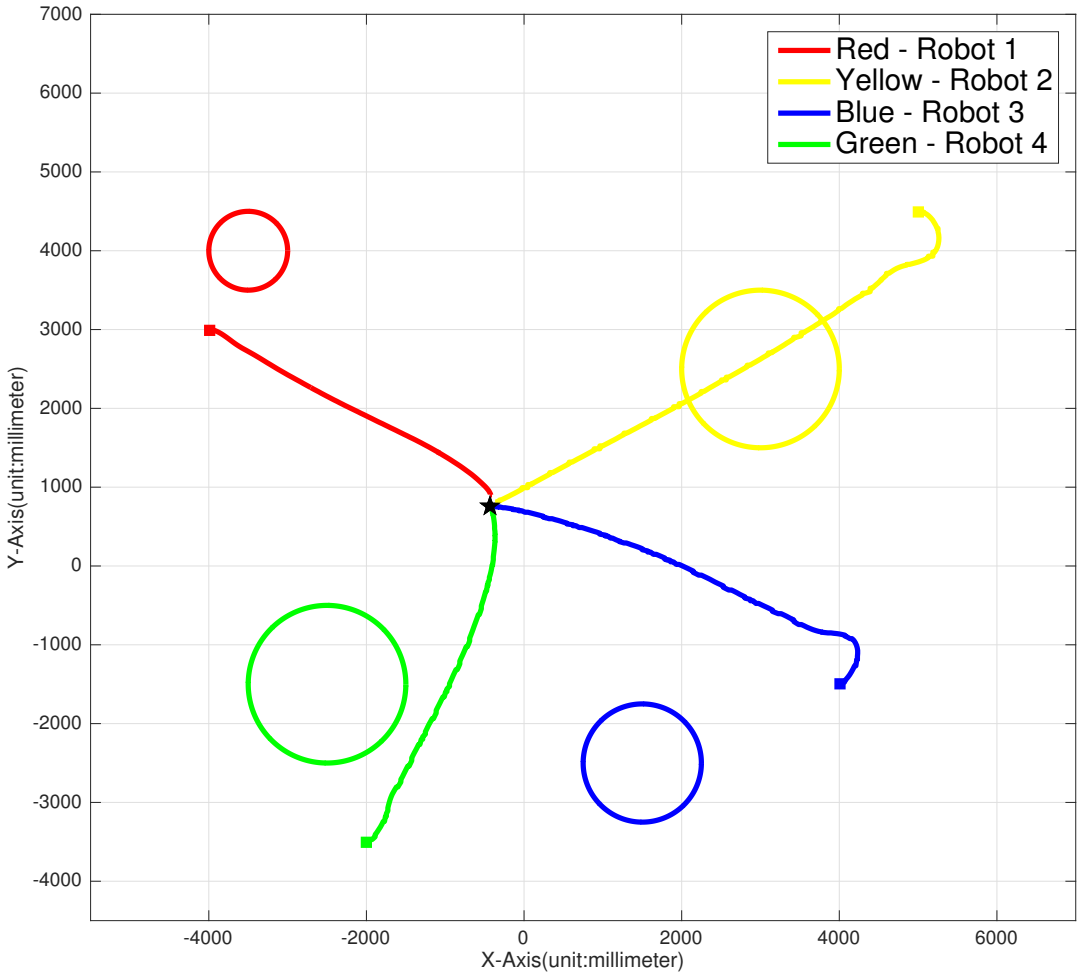

Figure 6.7: The experimental trajectory of robots when Algorithm 3 is used with $\alpha_1 = 2000$, $\alpha_2 = 200$

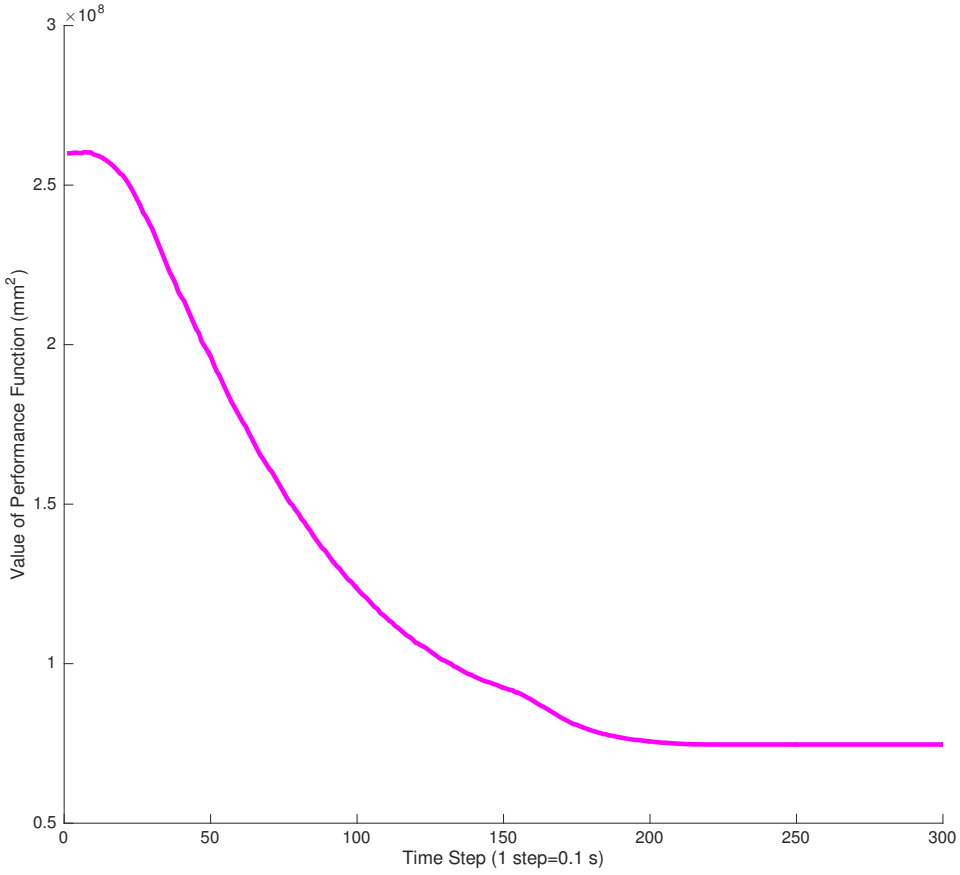

Figure 6.8: Performance function of Algorithm 1 with $\alpha = 800$, $\gamma = 0.022$

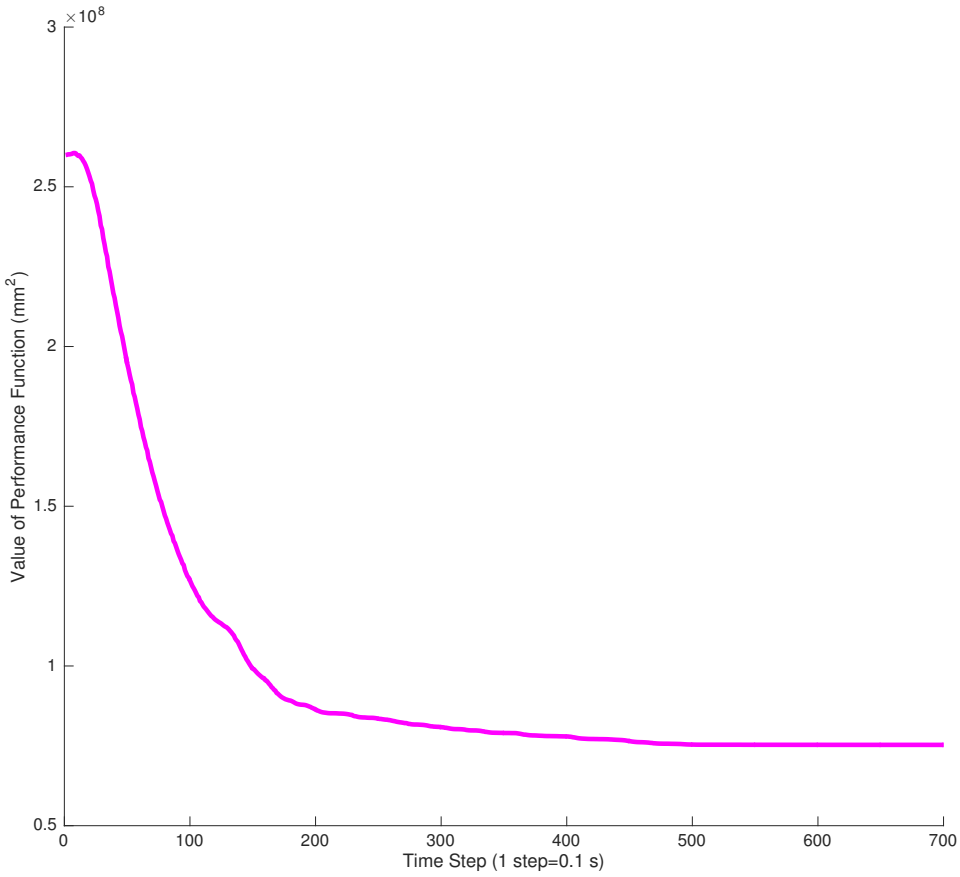

Figure 6.9: Performance function of Algorithm 2

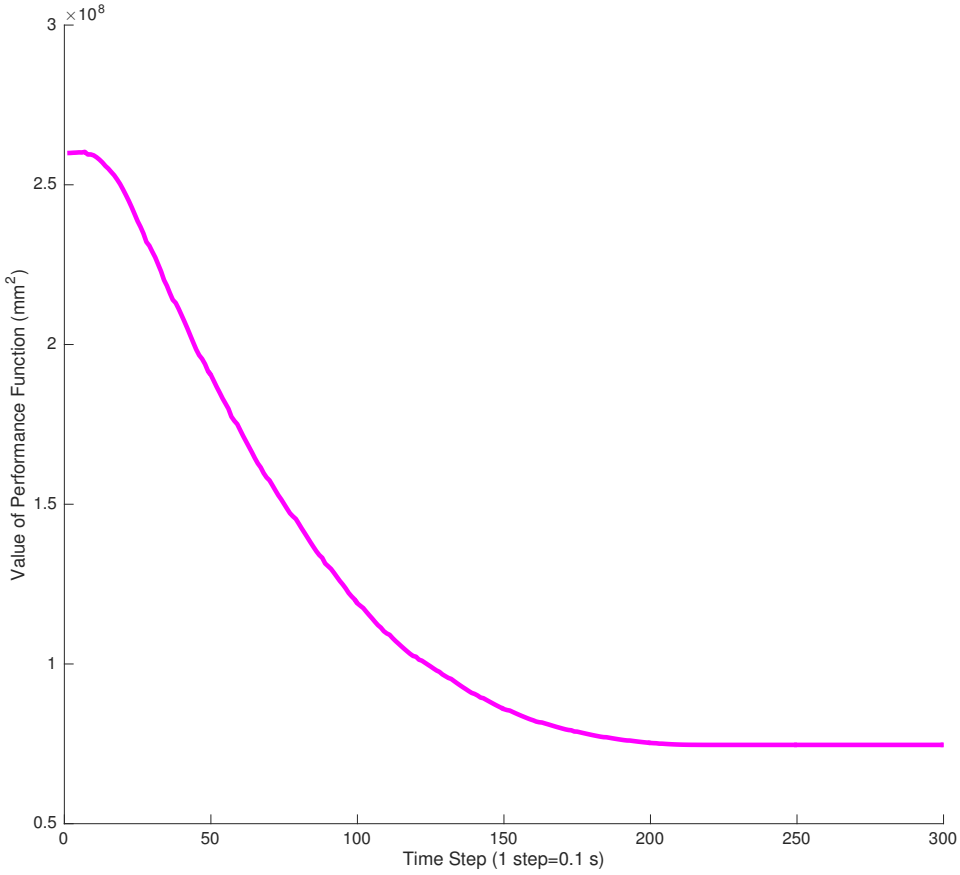

Figure 6.10: Performance function of Algorithm 3 with $\alpha_1 = 2000$, $\alpha_2 = 200$

6.2 Collision Avoidance

In order to avoid the collisions between robots in physical experiments, I introduced offset vectors to robots. Every robot was scaled 1m $\times$ 1m out its theoretical location by the offset vector as shown in Fig. 6.11. The offset vectors will not influence the execution and correctness of the algorithms. And finally, all robots will form a 1 m $\times$ 1 m square shape with center located at the optimal point if the algorithms work correctly.

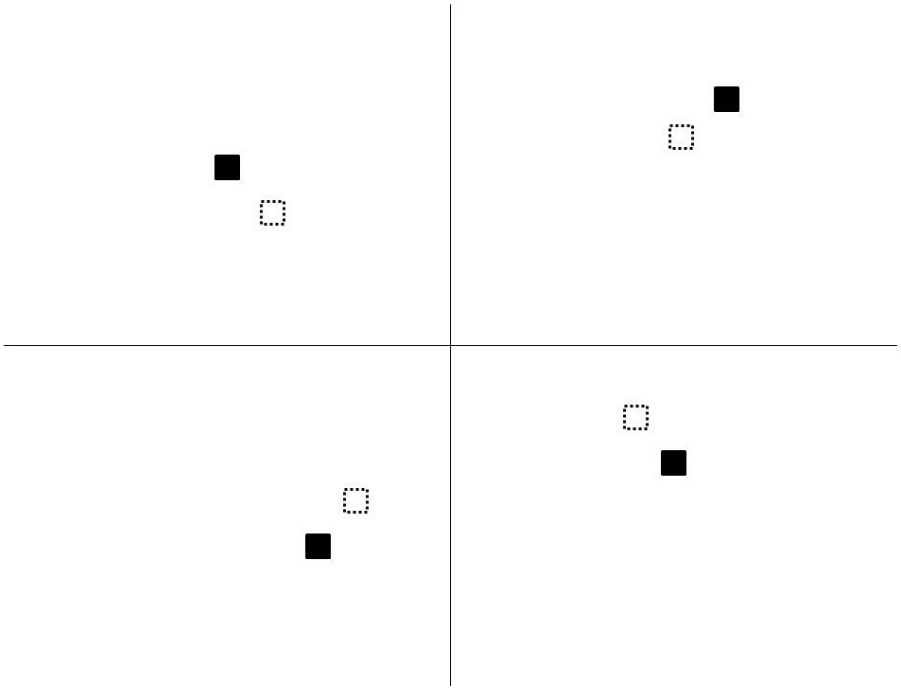

Figure 6.11: Offset vectors introduced to avoid collisions

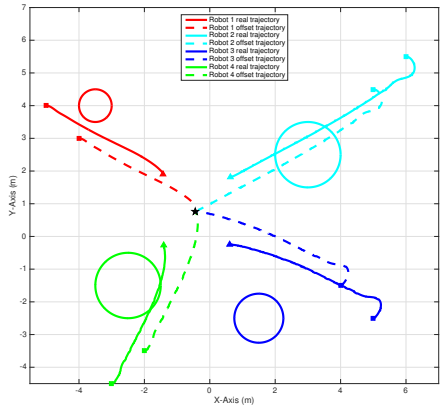

Figure 6.12:

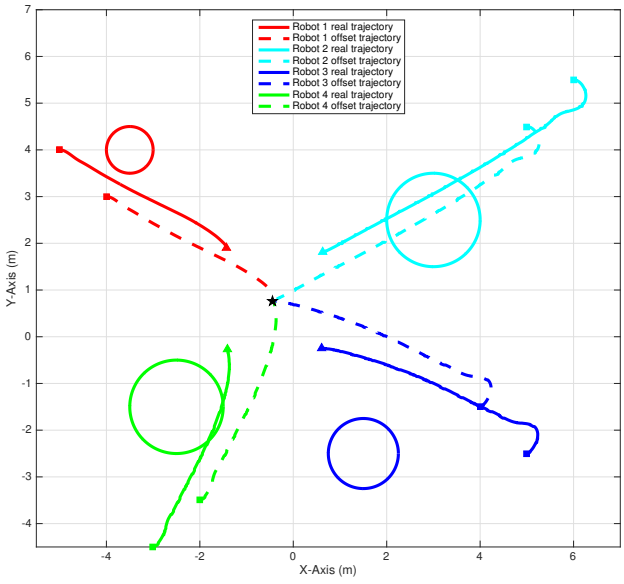

Figure 6.13:

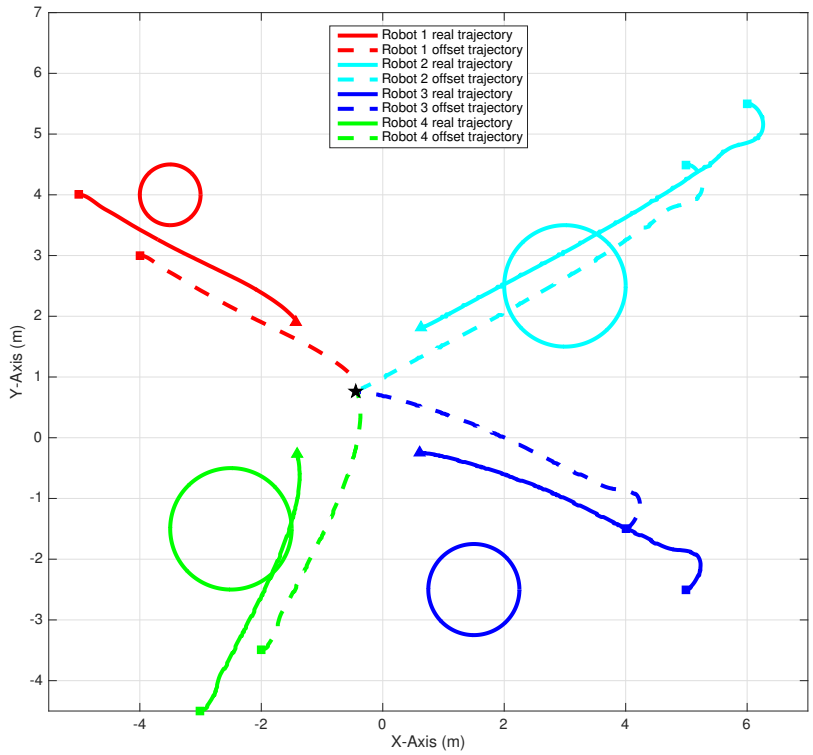

Figure 6.14:

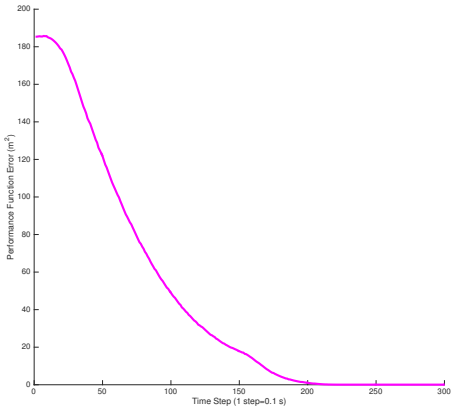

Figure 6.15:

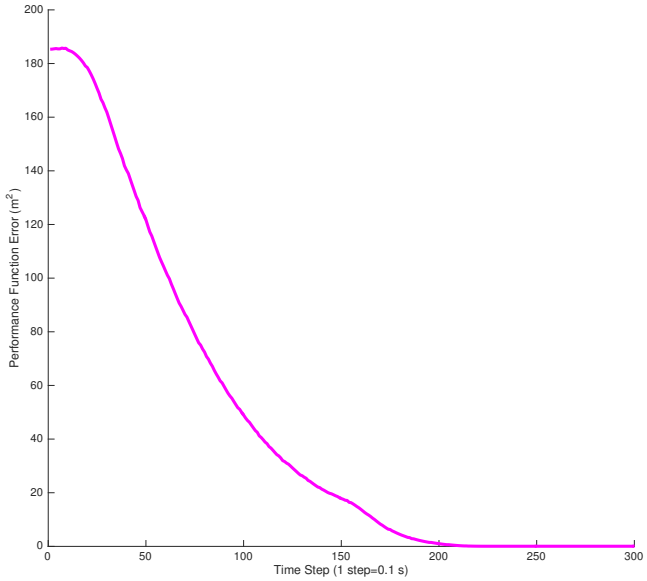
Performance Function Error (m^2)
Time Step (1 step=0.1 s)
0
20
40
60
80
100
120
140
160
180
200
0
50
100
150
200
250
300

Figure 6.16:

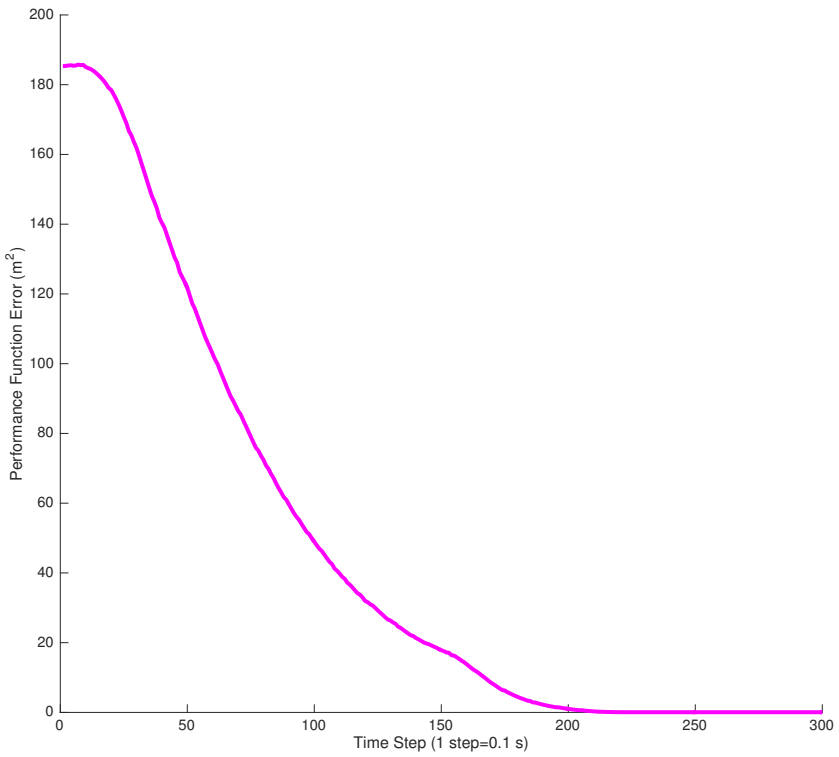
Performance Function Error (m^2)
Time Step (1 step=0.1 s)
0
20
40
60
80
100
120
140
160
180
200
0
50
100
150
200
250
300

Figure 6.17:

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis focuses on the implementation of multi-agent rendezvous problem where the rendezvous location has the total squared shortest distance to certain regions. We first introduced the problem and corresponding algorithms. Then we presented implementation of server and client programs, multi-robot platform, algorithms simulation results and experimental results on ground robots. Experimental results on the multi-robot platform showed the correctness and effectiveness of the algorithms which are consistent to the theoretical results in the Matlab simulations. Although there exist some errors, they might be caused from the discretization of the algorithms and the introduction of the offset vectors. Note that the errors are not very large and they are acceptable when no high accuracy is required.

7.2 Future Work

There are two aspects in this thesis which could be improved in the future. One aspect is to introduce time varying convex regions. The optimal point will keep changing according to the location and shape of convex regions. Another aspect is kinematic model of robot should be considered in detail. In this experiment, we didn't study robot's kinematic model, and we only regarded robot as a particle without considering its linear and angular velocity.

Bibliography

[1] https://en.wikipedia.org/wiki/Sign_function,.

[2] http://www.nathanobert.com/aria/figures/Robot_Communication_Options.png,.

[3] http://robots.mobilerobots.com/wiki/File:MobileSim.png,.

[4] S. Kar and J. M. F. Moura. Distributed consensus algorithms in sensor networks with imperfect communication: Link failures and channel noise. *IEEE Transactions on Signal Processing*, 57(1):355–369, Jan 2009.

[5] Shihua Li, Haibo Du, and Xiangze Lin. Finite-time consensus algorithm for multi-agent systems with double-integrator dynamics. *Automatica*, 47(8):1706 – 1712, 2011.

[6] T. Li, M. Fu, L. Xie, and J. F. Zhang. Distributed consensus with limited communication data rate. *IEEE Transactions on Automatic Control*, 56(2):279–292, Feb 2011.

[7] P. Lin and W. Ren. Distributed shortest distance consensus problem in multi-agent systems. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 4696–4701, Dec 2012.

[8] Wei Ren and R. W. Beard. Consensus seeking in multiagent systems under dynamically changing interaction topologies. *IEEE Transactions on Automatic Control*, 50(5):655–661, May 2005.

[9] Wei Ren, R. W. Beard, and E. M. Atkins. A survey of consensus problems in multi-agent coordination. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 1859–1864 vol. 3, June 2005.

[10] Wei Ren, Randal W Beard, et al. Consensus seeking in multiagent systems under dynamically changing interaction topologies. *IEEE Transactions on automatic control*, 50(5):655–661, 2005.

[11] G. Shi, K. H. Johansson, and Y. Hong. Multi-agent systems reaching optimal consensus with directed communication graphs. In *Proceedings of the 2011 American Control Conference*, pages 5456–5461, June 2011.

[12] Jianan Wang and Ming Xin. Optimal consensus algorithm integrated with obstacle avoidance. *International Journal of Systems Science*, 44(1):166–177, 2013.

[13] Wenwu Yu, Guanrong Chen, and Ming Cao. Some necessary and sufficient conditions for second-order consensus in multi-agent dynamical systems. *Automatica*, 46(6):1089 – 1095, 2010.