VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Bin Feng, Yuan Gao (in alphabetical order)

Development of Strategy Software and Algorithm Simulators for Multi-Agent System in Dynamic Environments

Technology and Communication

2013

PREFACE

We have implemented the applications at the Telecommunication and Information Technology Department in the Vaasa University of Applied Sciences from December 2011 to December 2012.

The thesis is co-written by Yuan Gao and Bin Feng. Yuan Gao is in charge of Chapter 2 Structure of System, Chapter 3 Simulation Software, Chapter 4 Strategy Software, Chapter 9 Expectation of Research Direction. Bin Feng is in charge of Chapter 5 Vision System and Kalman Filter, Chapter 6 Safety Navigation, Chapter 7 Motion Planning, Chapter 8 Overview of To-do Lists in Botnia SSL Team. The rest of the chapters are written together by us.

Firstly, we would like to express our greatest gratitude to our thesis supervisor, Dr. Yang Liu, for his splendid advises. Ever since, he has supported us not only by delivering a research and development assistantship over almost four years, but also academically and emotionally through multiple milestones to finish this thesis. He helped us come up with the thesis topic and guided us over almost a year of development. Whenever we came across difficulties he gave us the moral assistance and the independence we needed to move on.

We are also very grateful to Dr. Chao Gao, Mr. Matila Jukka, Dr. Moghadampour Ghodrat, Mr. Chavez Santiago, Mr. Ahvonen Jani, Mr. Virtanen Antti, Mr. Hahtokari Tapani, Dr. Reino Virrankoski, Mr. Tobias Glocker who together assisted us by providing constructive comments and suggestions, all of which have helped improve the content and quality of this thesis. Their deep thoughts and suggestions have made a strong salutary impact to the whole research and development process.

We also would like to credit all members of Telecommunication and Information Technology Department at Vaasa University of Applied Sciences for providing us decent, cosy, development-focused environment during our four years study time.

I, Yuan Gao, am grateful to my beloved girlfriend, Chuyi Yang, for her support, understanding, comprehension, and patience all throughout my time of studying

and producing this thesis. Simultaneously, I thank my friends for their caring when I am in a situation of misfortune. I also would like to thank my parents for their love, care, great sacrifice during all these years.

I, Bin Feng, thank my fellow lab-mates in the RoboCup robotics lab for the inspiring discussions, for all those days and nights devoted to those challenging work, and for all the joys we had during the past four years. I also thank my family for giving me a full support in my study and life.

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Degree Program of Information Technology

ABSTRACT

Author	Bin Feng, Yuan Gao (in alphabetical order)
Title	Implementations on RoboCup Algorithms
Year	2013
Language	English
Pages	194
Name of Supervisor	Yang Liu

In the field of mobile robots, many interesting and important algorithms are employed in the process of implementing a robust, collision-free multi-robot collaboration movement.

The main objective of this thesis is to introduce strategy software of RoboCup multi-robot system, and several related algorithm simulators that we made during the development process. Another objective is to introduce several crucial algorithm implementations for the system, for instance, Kalman filter, safety navigation algorithm and path-planning algorithm.

This thesis reanalyses, compares against research results in the history and today, and then proposes a solution with verification and correction. Along with the theory, we construct the theoretical framework of the strategic algorithm system. And then we tested the experimental effect with several simulation programs. The implementation results have been verified to be success by multiple tests. Finally, this thesis also suggests the to-do list and expectations of research directions.

The main result of the thesis contains the implementation of strategy software, algorithm simulators and the implementation of several crucial algorithms.

All the thesis source codes can be found at: https://svn.puv.fi/Botnia2011/

Key words: Agent-control software, algorithm simulator, algorithm complexity, algorithm efficiency, path-planning, navigation, Robocup.

CONTENTS

LIST OF ABBREVIATIONS

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
BRRT	Boarded Rapidly-Exploring Random Tree
CRC	Cyclic Redundancy Check
DECT	Digital Enhanced Cordless Telecommunication
DHCP	Dynamic Host Configuration Protocol
DRRT	Dynamic Rapidly-Exploring Random Tree
ERRT	Extended Rapidly-Exploring Random Tree
FPGA	Field-programmable Gate Array
JSON	JavaScript Object Notation
PRM	Probabilistic roadmaps
QSS	Qt Style Sheet
RPM	Revolutions per Minute
RRT	Rapidly-Exploring Random Tree
SLIP	Serial Line Internet (or, Interface) Protocol
SLL	Small Size Robot League
SPI	Serial Peripheral Interface Bus
UDP	User Datagram Protocol
XML	Extensible Markup Language

LIST OF FIGURES AND TABLES

1 INTRODUCTION

1.1 Purpose of Thesis

This thesis introduces the RoboCup small size league (SSL) implementation to the readers. The main work that we had finished for SSL project is strategy software and several corresponding simulators. In the following sections, we will introduce the structure of the system and explain particularly the structure of the strategy software and corresponding algorithm simulators.

1.2 Overview Structure of Thesis

This thesis can be divided into 11 Chapters. The first chapter Introduction mainly introduces the background of RoboCup and Botnia SSL team. The second chapter Structure of System explained several modules in the system, including hardware, embedded software, vision software, etc. The third chapter Simulation Software illustrated several simulation software. The fourth chapter Strategy Software analysed each module in the system and explained the functionalities of each of them. The fifth chapter focus on vision system and Kalman Filter, which is a powerful algorithm implemented in our system. The sixth chapter implemented the safety navigation algorithm in the system to prevent collision. The seventh chapter is about motion planning, it highlights a novel design and implementation of a RRT-based algorithm. The eighth chapter gives an overview of future to-do lists in our team. The ninth chapter suggested an expectation of future research directions. The tenth chapter gave a summary of the whole thesis. The eleventh chapter is for appendix, which included the completed codes for the previous mentioned algorithms.

1.3 Background of RoboCup

In 1993, Professor Alan K. Mack Worth from Department of Computer Science at the University of British Columbia in Canada brought his innovative idea that robot soccer competition would be a suitable research platform for developing science and technology in the field of robotics and artificial intelligence.

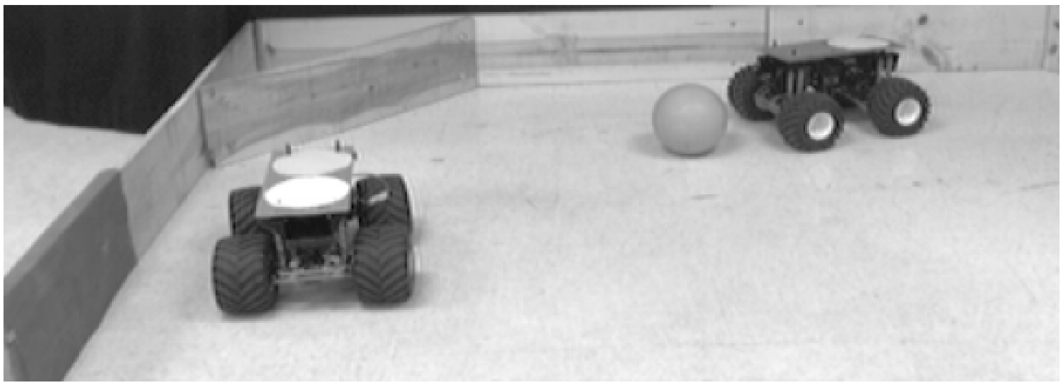


Figure **1.** Two soccer players compete in the Dynamo project [1]

Today, RoboCup (Robot World Cup) [2], an international soccer robot competition, whose goal focuses on providing a platform for developing innovative solutions in the field of intelligent robots, has become one of the largest soccer robot competitions in the world together with FIRA Cup.

RoboCup competition is held under a dynamic adversarial environment with a system of autonomous multi-robot and multi-agent coordination. There are several different types of leagues available in RoboCup competition, including Humanoid, Middle-Size, Simulation, Small-Size, and Standard-Platform.

Humanoid League (HL) focuses on human-like soccer robots confrontation. Key skills involved in the Humanoid League include balance keeping, walking and running, kicking balls, visual signal receiving and processing, self-localization, dynamic path finding, collision avoiding, etc.

Middle Size League (MSL) consists of 6 soccer robot players per team, and the soccer ball used in the competition is the same as the regular size FIFA ball. Soccer robots rely on on-board sensors to communicate with each other wirelessly.

Simulation League (SL) has two sub-leagues in 2D and 3D. Both are focusing on developing simulators for soccer robot competition without hardware support.

New artificial intelligence algorithms and team strategies can be tested in a simulation environment before applying them to real robots.

Small Size League (SSL) is known for its highly dynamic environment in which all robots are supervised by an overhead camera. All vision information captured from the field is collected by a centralized off-field computer running with a vision system (SSL-Vision) [3]. By analysing and processing the raw vision information, the computer can control and coordinate soccer robots in filed by sending and receiving signals. Building a successful SSL team requires multidisciplinary knowledge of fields including robotics, computer science, electrical engineering, mechanical engineering, etc.

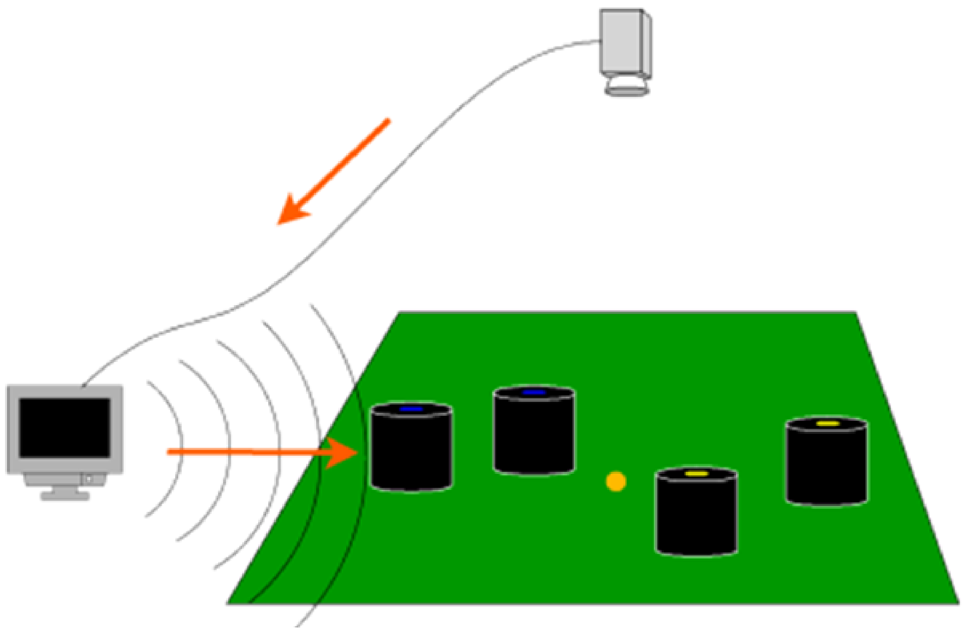


Figure **2.** RoboCup SSL Dataflow [2]

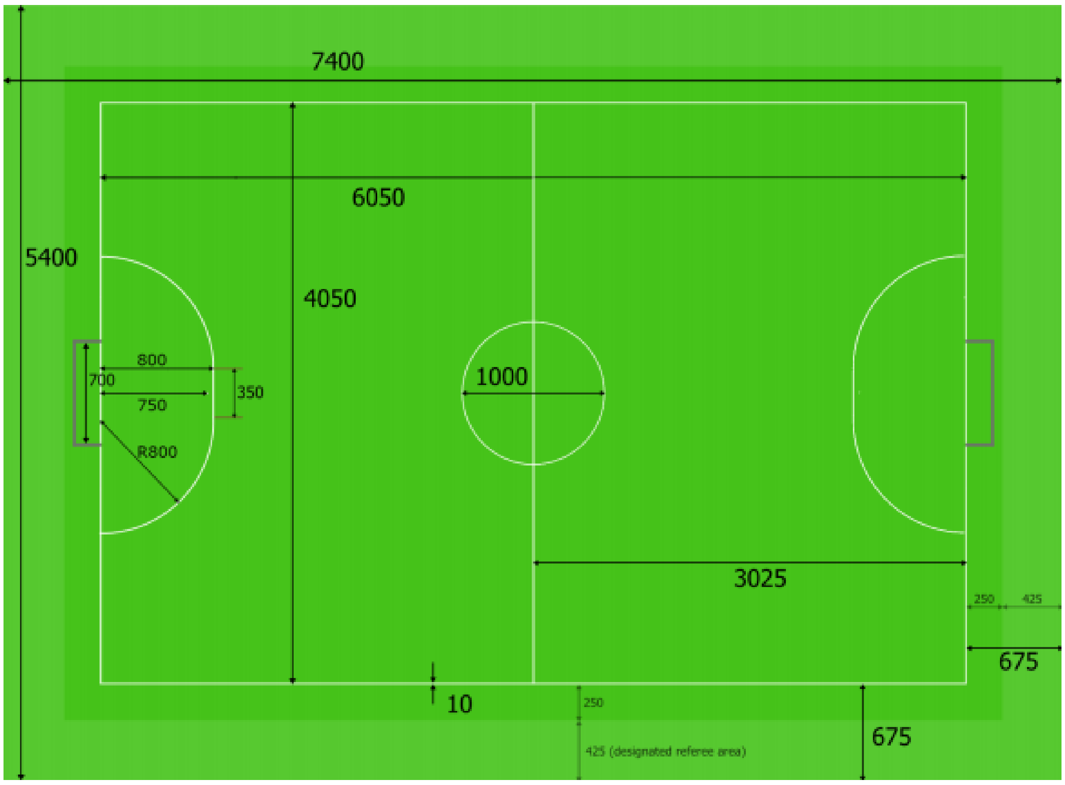


Figure **3.** Field Dimension of RoboCup SSL 2012 [8]

Standard Platform League (SPL) requires soccer robots be able self-localized and locating ball since the vision system is forbidden. In SPL, all teams have same standard robots hardware, participants would be required to focus on software strategy development.

1.4 Background of Botnia SSL Team

Starting from 2006, Botnia SSL Team has been in the RoboCup world championship and ranked top 10 in Small Size League competition. Currently, our team is the only qualified RoboCup SSL team in the whole Nordic region.

Up to now, there have been three major updates (SR4E, SR5, SR6) for hardware part of soccer robots in our team. The latest version SR6 of hardware has a sophisticated design and an excellent performance.

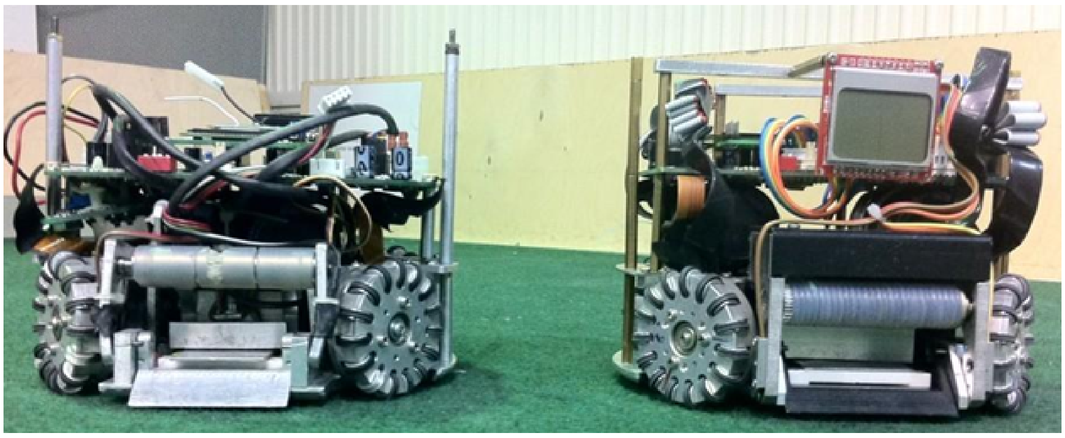
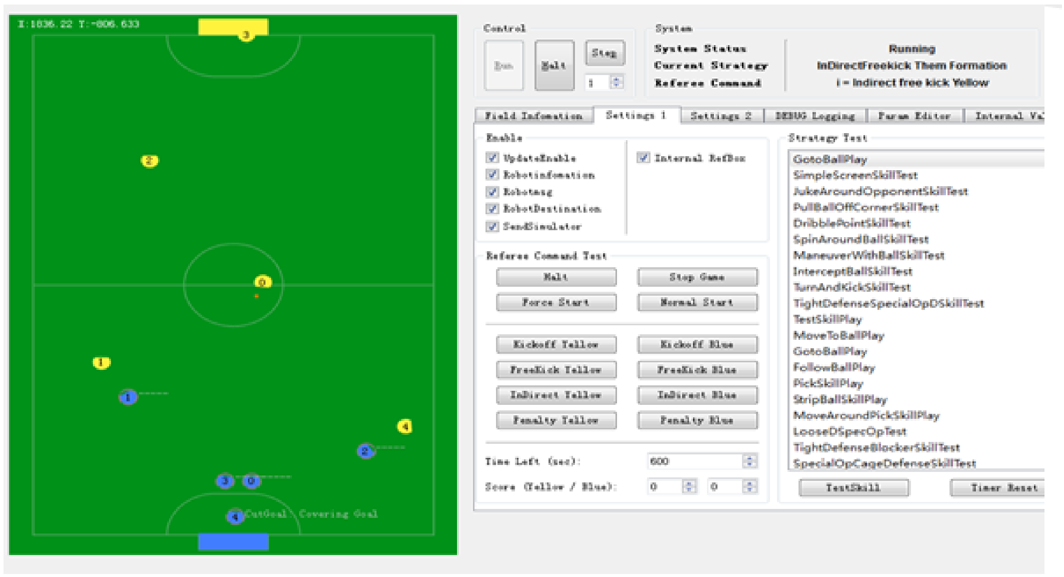


Figure **4.** Botnia Robot SR4 and SR6 [4]

Besides a sophisticated hardware, our team also has a complete strategy software system. The windows-based software system is used for analysing and processing incoming vision information and send precise commands to infield robots based on the strategy system. In order to have a better real-time control and performance, currently we are rewriting the codes to build a new Linux-based strategy software system.



System Status: Running
Current Strategy: InDirectFreekick Them Formation
Referee Command: i = Indirect free kick Yellow

☑ UpdateEnable
☑ Robotinfomation
☑ Robotmsg
☑ RobotDestination
☑ SendSimulator
☑ Internal RefBox

Time Left (sec): 600
Score (Yellow / Blue): 0 0

GotoBallPlay
SimpleScreenSkillTest
JukeAroundOpponentSkillTest
PullBallOffCornerSkillTest
DribblePointSkillTest
SpinAroundBallSkillTest
ManeuverWithBallSkillTest
InterceptBallSkillTest
TurnAndKickSkillTest
TightDefenseSpecialOpDSkillTest
TestSkillPlay
MoveToBallPlay
GotoBallPlay
FollowBallPlay
PickSkillPlay
StripBallSkillPlay
MoveAroundPickSkillPlay
LooseDSpecOpTest
TightDefenseBlockerSkillTest
SpecialOpCageDefenseSkillTest

Figure **5.** Second Generation of Windows-based Strategy System [4]

1.5 Motivation

The main motivation of going into this field is that the multi-robot system has been a great research platform that has been designed for any newcomers. It includes a complicated mechanism behind the iron blocks and the complicated circuits.

People can do research in robotics on the subject of navigation, logic programming, computer vision, path-planning, navigation and controller. In the international research forum, there are always quite a lot interesting subjects popping up including the project which combines the robotics with the evolution theorem [1], the project which researches on SLAM (Simultaneous localization and mapping) or virtual SLAM [2] and the project which detects the network boundaries [3].

Our project, however, may be limited to the platform that we are using. According to what we know about some research labs. The platforms that people use may vary depending on the different purpose. The typical ones are the iRobot, swarm robot and the one we are using - the multi-robot system of SSL.

Influenced by robots, the research topic is also limited to a small range. The tests are mainly aimed for the path-planning algorithm and navigation. As a consequence, we designed the framework of strategy software and the simulations for information transform, evolutionary algorithm, random walk and so on.

All these are the basis to achieve the final goal of designing a better software structure of the multi-robot. If what we have done for robots can move this area a little bit forward then we are already happy about it.

1.6 Terminology

In this thesis, if we introduce an abbreviation the first time, we would also give its full word together. In addition, we will use a grey highlighting to emphasize code that is particularly important in the present context.

We will use two kinds of code representation. They are code list and code example.

- Code listing is a complete program. As stated before, in most cases, they will provide a summary of the code. This summary will tell you the environment, compiler instruction, linking instruction, other description about the code.
- Code example is a piece of code which is isolated from other part of the program and cannot be run without changes. They are used to represent how the API (Application Programming Interface) and algorithms are used.

1.7 Summary

In this chapter, the background of RoboCup and Botnia SSL Team are introduced. RoboCup SSL is a fascinating competition which requires multi-disciplinary knowledge in robotics, computer science, electrical engineering, communication technology, mechanical engineering, and so on. Besides, RoboCup provides a highly-valuable platform for students to learn both in theoretical and practical fields.

In the next chapter, we will describe structures of this system.

2 STRUCTURE OF SYSTEM

2.1 Introduction

This chapter will give a general view on the structure of the system. It introduces the basic information and background knowledge of entering the technical details of the system. After this chapter, we will go through each part individuality.

2.2 Structure of System

The structure of the system is as follows. Here, we have instance diagram and logic diagram to describe the relationships between the hardware and software from different points of view:

- Instances diagram

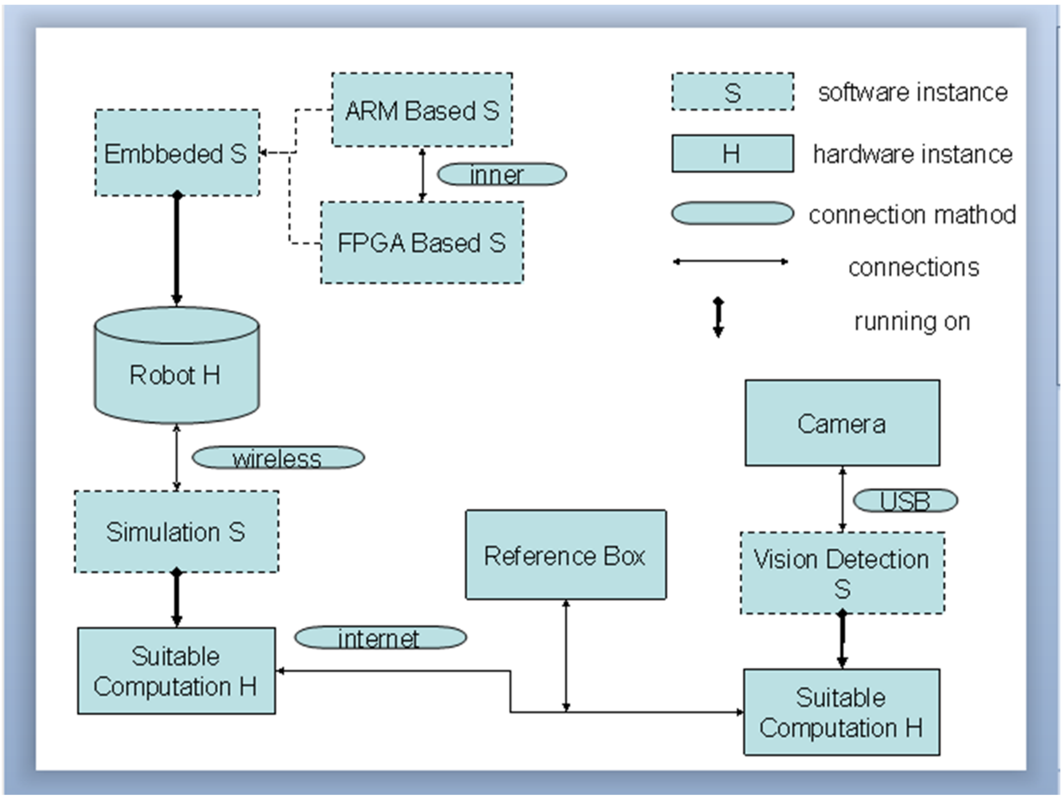


Figure **6.** the Structure of System

- logic diagram

From the previous diagram, we can make an abstract diagram to separate each functional section.

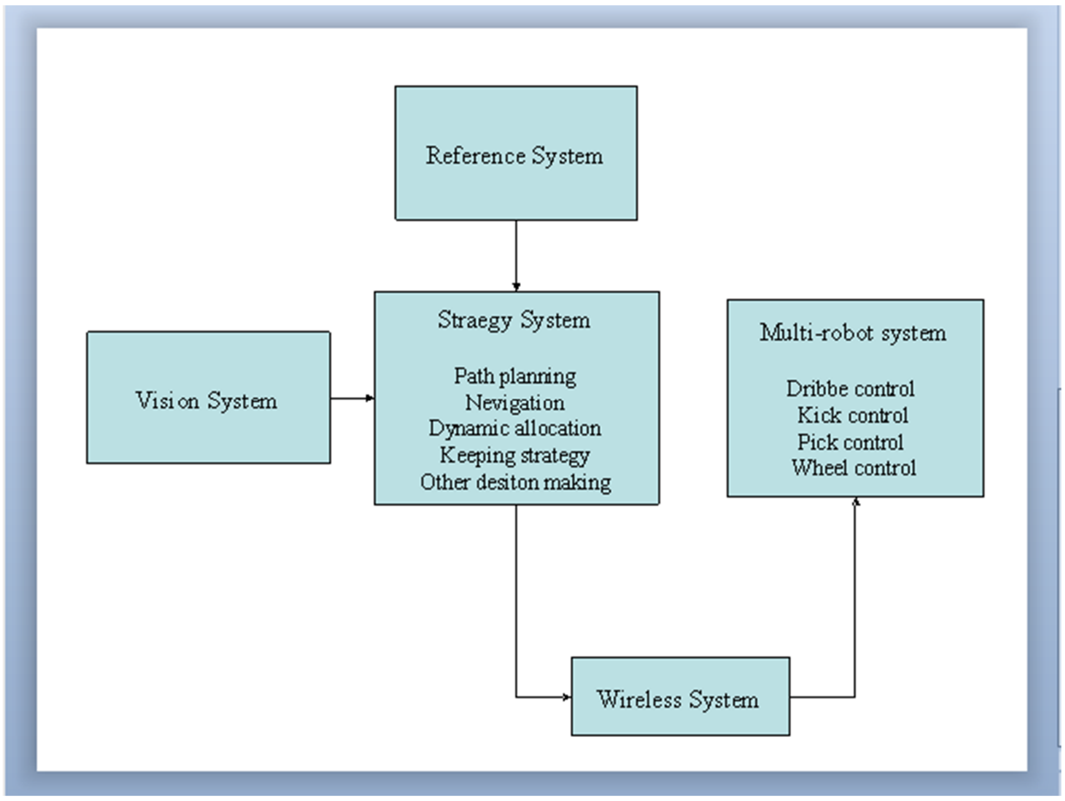


Figure **7.** Logic Structure of System [1]

From the logic structure of the system, we identify there are five main parts in it. Namely these five parts are wireless system, vision system, multi-robot system, strategy system and reference system.

1. Wireless System

The main function of the wireless system is to communicate with the strategy software and send the information to robots so that it can control and monitor robot properly. To achieve that, we need not only to implement modules in strategy software but also in the robot strategy software.

2. Vision System

The main function of the vision system is to collect, monitor the environmental information on the competition field and then provide the support information of position and direction to the strategy system. Now, in the competition, this system has been standardized.

3. Multi-robot System

The main function of the multi-robot system is to perform the command sent by the strategy system under different condition. In this process, it may be able to auto-adjust its parameters to suit the dynamic environment. It also needs to be able to send information back to let the strategy software to analyse it.

4. Strategy System

The main function of the strategy system includes evaluating according to the information on the competition field, to make outlined strategy of the multi-robot system, to determine the role of each robot in a team. Normally, each team needs one computer to deal with this job.

5. Reference System

The reference system should be taken care by two parts - both human and the reference box. In this case, the referee should cooperate with the reference box. RoboCup is a very tightly connected platform, and the reference box will not influence the processing of the competition.

The competitors can improve the vision system, communication system and multi-robot system to accomplish mission better. Among all of these systems, the strategy system is the dominant factor of relevant factors. As a consequence, the improvement on the strategy system can bring better result. The strategy system directly decides the rule of behaviour and it also can avoid the obstacles while robots are moving around.

The environment of the competition is as follows:

- Real-time

To ensure the real-time operation is the basic requirement for the multi-robot system win the competition. The condition of the football changes quickly and dramatically. The fast reaction of each individual robot is the key factor to make a goal in the game.

In the competition, the vision collects information of the field with a period as 33ms through the camera. Then wireless communication system sends the information to the strategy system to evaluate. As a consequence, the strategy software, communication sub-system and the multi-robot system need to have very fast speed and efficiency to make real-time decisions.

- Dynamic

The wheel of each robot in multi-robot system can continuously move with two degrees of freedom. In theory, the maximum speed can be 2 meters per second and change in velocity and direction can be extremely frequent. This quite possibly results robot to collide into each other. For avoiding collision and going to the goal, the detector should have fast reactive speed.

- Uncertain

This multi-robot system is made by several sub-systems. The vision sub-system is responsible for collecting vision information. It is a process of making a continuous system to become discrete. Admittedly, it is very hard to record all the information. Same part of the graph may be affected to have distortion by light, clothes, platform or even the dust in the air.

The communication is also possible to be influenced by the interferences. When small disturbance happens, many robots may run into big messes which then generate even more disturbances. Additionally, the multi-robot system may not operate accordingly due to lacking of the enough electricity. All of above can cause the environment of multi-robot system to be unstable.

- Non-linear

The environment of RoboCup competition and the system of multi-robot system have obvious non-linear property. This directly leads to a complicated description of the system. Simultaneously the strategy software is limited by the time and capacity of calculation. Therefore, obtaining the satisfactory solution of path-planning and navigation is also a big challenge. If we do not have a reasonable way of solving this problem, the computational time and the difficulty of controlling multi-robot system will be increased dramatically. This shows that, at the base of difficult points, which was previously brought by environmental properties, no matter the purpose is theoretically researching the problem or practically solving the problem, it would bring significant meaning to the decision making process and strategic control problem. Among all these problems, the core of the strategy software is path-planning problem and collision detection problem and these two points are the emphasis in the strategy software chapter.

From the viewpoint of AI, the environment can also be classified into following categories:

- Partially Observable

This system is partially observable. The issues like unbalanced light, reference system, dynamic environment are strong enough to cause the exact information about the item on the field sometimes cannot be fully obtained. For example, the colour of the ball is unidentifiable when the light reflected from the ball is blocked by a moving robot. [2]

- Stochastic

If the next state of the environment is completely determined by the current state and the action executed by the agent, then the environment is deterministic. Otherwise it is stochastic. [2]

- Sequential

In episodic environment, an agents experience is divided into atomic parts. One episode doesn't affect the next. For example: an agent assigned to find defective parts in an assembly line is in episodic environment. Sequential is the reverse of it. For example: a chess playing agent is in a sequential environment. [2]

- Dynamic

If the environment changes while the agent is operating, then it's dynamic, else static. If the performance of the agent changes the environment, then it's semi-dynamic. For example: taxi driving is dynamic, chess is semi-dynamic and crossword puzzle is static. [2]

- Continuous

The state is continuous. A chess game is of discrete states, while driving a taxi requires continuous actions, continuous steering over continuous time. [2]

- Multi-agent:

If the agent operates in an environment where the only agent is him, then it's single agent environment. Agent playing crossword puzzle is in the single agent system. A taxi driving agent is in Multi-agent environment. [2]

2.3 Hardware

This system includes four sections of hardware, namely vision hardware platform, strategy hardware platform, multi-robot hardware platform, wireless communication hardware platform.

The first one is vision hardware platform, which is also a PC. The software which controls the vision system is offered by the SLL organizer. People can download and compile it on a Linux-based computer.

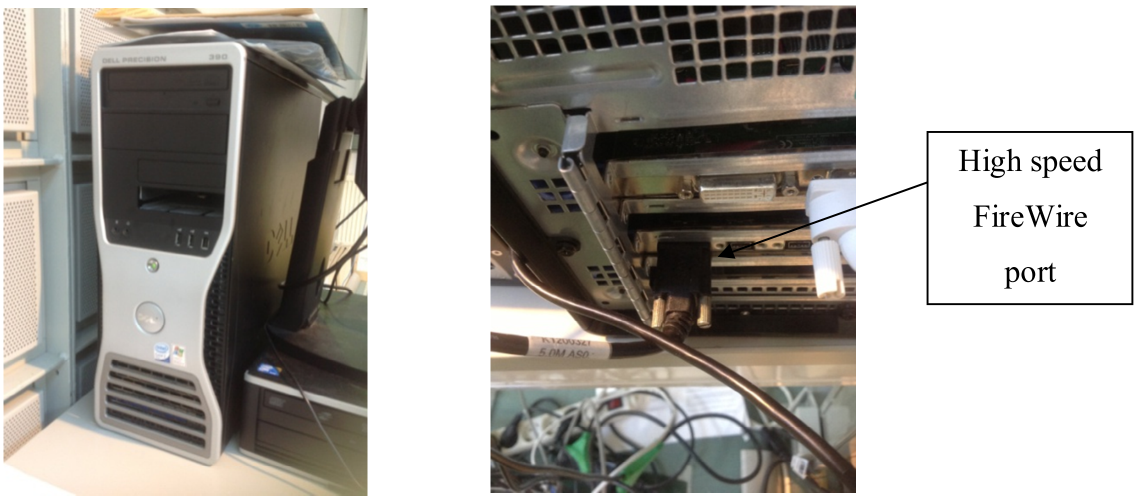


Figure **8.** the Front Side and Back Side of Vision PC

On the back side of the server, we can see a high speed camera connected to it through a high speed FireWire port. The protocol used for this connection is an IEEE standard called IEEE1394b. This connection is comparable with USB and always used together with USB. The reason why we chose this standard method as the communication method is because it is widely used for real-time information transfer.

Figure **9.** a High Speed Camera Connected to the Vision Server

The second one is a PC which runs strategy software. This PC is a standard PC which has a higher computational rate. For future development, we would like to ask the team to install a powerful GPU in it.

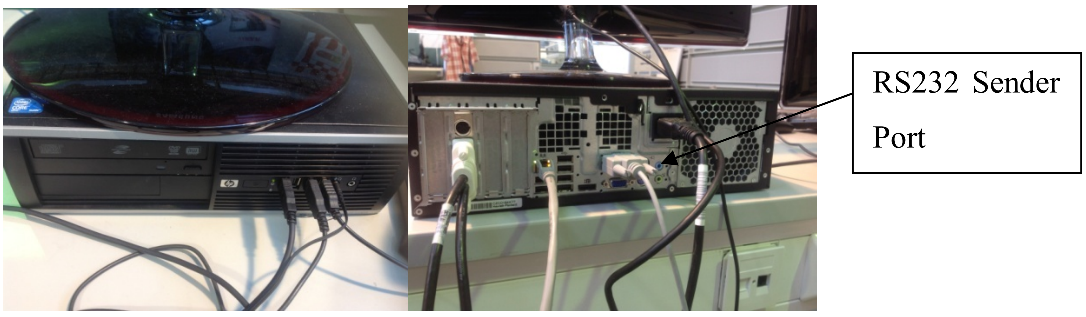


Figure **10.** the Front End and Back End of Strategy Software PC

The third part we would like to introduce is the multi-robot system. Each robot in the project contains different modules including wireless module, wheel module, dribbler module, etc.

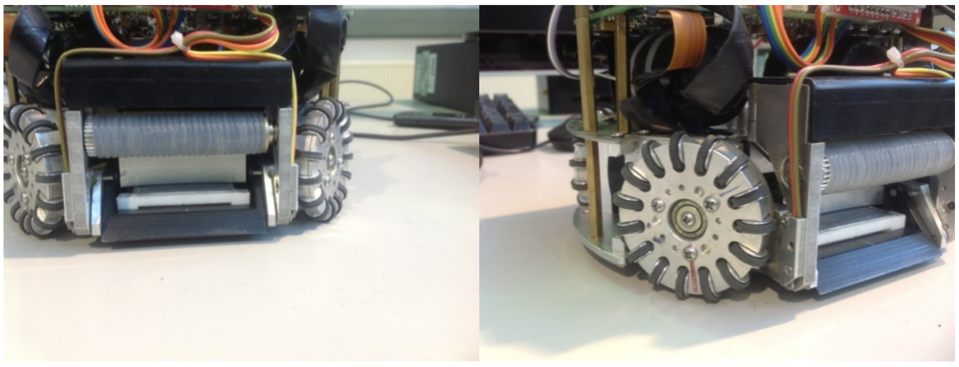


Figure **11.** the Dribbler and the Wheel of Robot

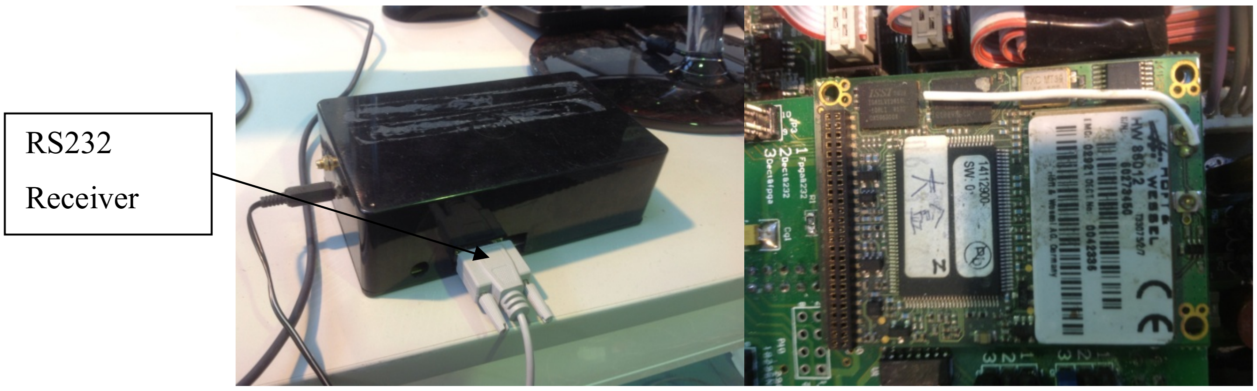


Figure **12.** Wireless Sender Connected with PC and Wireless Receiver on Robot

2.4 Embedded Software

Embedded systems contain processing cores that are either microcontrollers or digital signal processors (DSP).A processor is an important unit in the embedded system hardware. It is the heart of the embedded system. The key characteristic, however, is being dedicated to handle a particular task. Since the embedded system is dedicated to specific tasks, design engineers can optimize it to reduce the size and cost of the product and increase the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale. Physically, embedded systems range from portable devices such as digital watches and MP3 players, to large stationary installations like traffic lights and factory controllers. Complexity varies from low, with a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure.

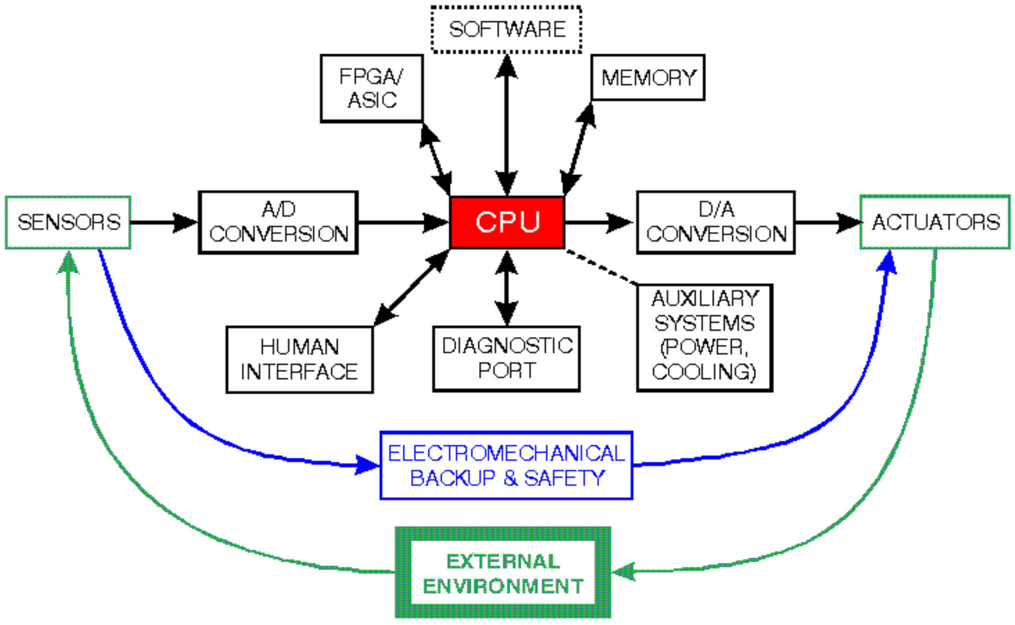


Figure **13.** Basic Embedded System Structure

2.5 Vision Software

In previous years, the RoboCup Small Size League rules have allowed every team to set up their own global vision system as a primary sensor. This option bears several organizational limitations and thus impairs the league's progress. Additionally, most teams have converged on very similar solutions, and have produced only few significant research results to this global vision problem over the last years. Hence the responsible committees decided to migrate to a shared vision system (including also sharing the vision hardware) for all teams from 2010. This system - named SSL-Vision - is currently developed by volunteers from participating teams. [3]

3 SIMULATION SOFTWARE

3.1 Introduction

Simulation software is a program used for the user to observe an operation through the simulation without actually performing the operation. Normally, it is based on a modelling process according to mathematical formulas or an algorithm.

During the stage of developing the strategy software, we encountered problems of making unit tests for algorithms. Therefore, we made the algorithm simulators independently for testing and simulating various algorithms. By using these algorithm simulators, we are able to visualize the data structure and computation process, and as a result, we can see the effect and benchmark of different algorithms.

Each of following sub chapter contains simulation software. The logic map would be like a zone as we will introduce them one by one in details.

Here is the list of small simulation software that we are going to talk about.

1. Random Walk Simulator For Path-planning project
2. Parameters Monitoring Web Server
3. Vision Simulation Software
4. Rapidly Random Exploring Tree Simulator
5. Path-planning Simulator

The first item in the list is the Random Walk Simulator that we made for the path-planning research. The following Parameters Monitoring Web Server is what we implemented for visualizing the data collected in the embedded robot software. The Vision Simulation Software is a possible exploration of using python for simulating vision software. Then the fourth software Rapidly Random Exploring Tree is a prototype for future development of this project. Finally the Path-planning simulator is a concrete C++ implementation that we used to test the time consumption of different algorithms.

3.2 Random Walk Simulator

3.2.1 Introduction

A random walk simulator was made to test the possibility of applying the random walk algorithm to the path-planning problem. When it executes, this random walk simulator asks key parameters from the user and then simulates the algorithms on graph. It offers the functionality of simulating either one-dimensional random walk or two-dimensional random walk.

3.2.2 Motivation

In 2011, Dr. Karaman presented a thesis discussing about his innovative RRT star algorithm of finding the optimal path for automatic system. This algorithm has been proven that this is very adaptive in many cases. [4]A system like RoboCup project, however, cannot use this algorithm as this algorithm takes a lot of computation resources. Although the time complexity and the space complexity of RRT and RRT star algorithms are both $O(n * \log(n))$. [5]The reason of latency lies in that the RRT star algorithm generates several paths for comparisons and this really dragged the speed down. [4]

A typical random walk is as follows:

To define this walk formally, take independent random variablesZ_1, Z_2 ..., where each variable is either 1 or -1, with a 50% probability for either value, and $S_0 = 0$and $S_n = \sum_0^{i=n} Z_n$. The set$\{S_n\}$ and the series is called the simple random walk on $\mathbb{Z}$.

If we apply a random walk on RRT star algorithms, when RRT star algorithm converges very slowly, I believe the modified RRT start algorithm can bring a better performance.

3.2.3 User Interface

1. User Interface

The user interface is designed for the convenient test of the algorithm. As a consequence, the UI can be simply divided into three parts. The first part is the action panel which is on the top of the whole view, the second part is the tab widget which is in the middle and the third party preference is on the left of the user interface. It specifies the configurations of the simulation.

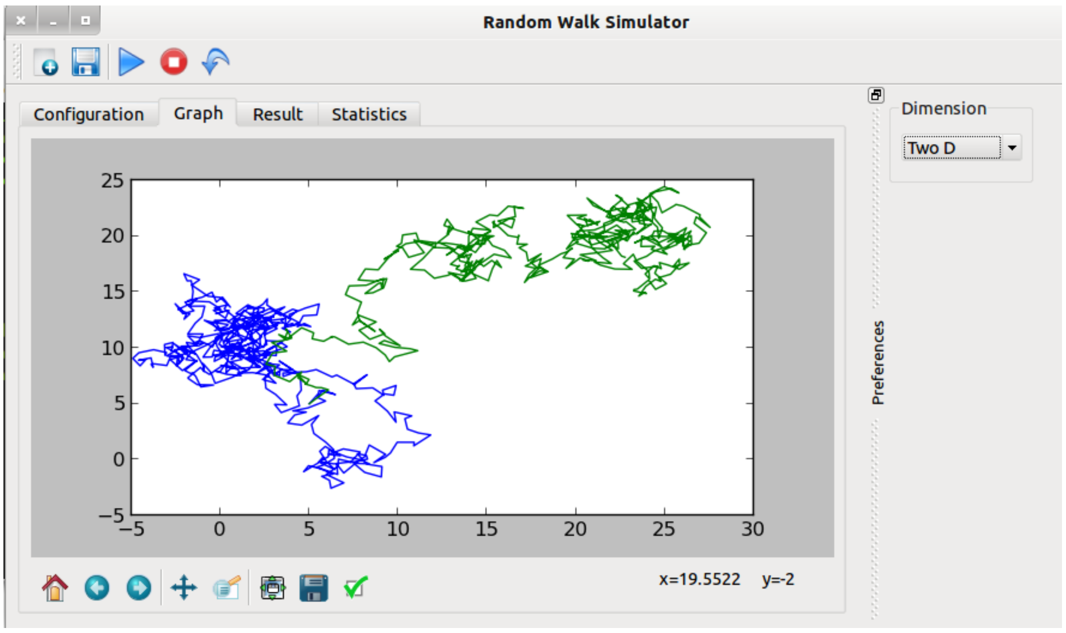
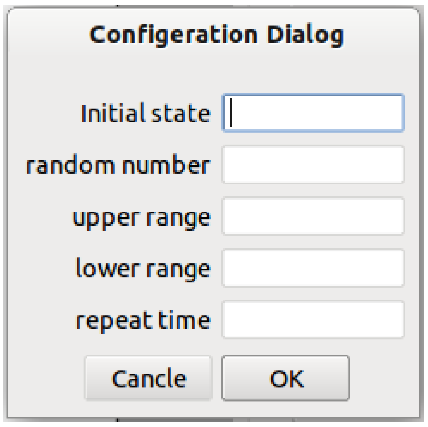


Figure **14.** User Interface of Random Walk Simulator

- Action Panel

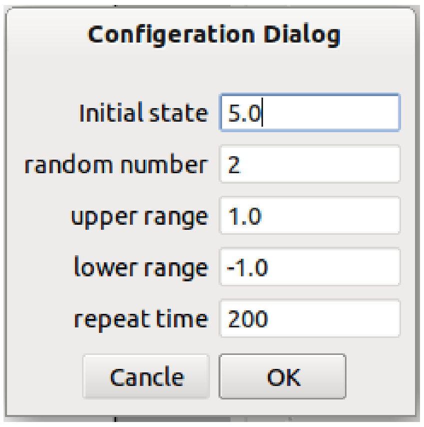
The entrance of the user interface is as presented above. There are five actions in the action panel. The first one is to make a new configuration, when it is clicked, there will appear another setting window to adjust the key parameters of the simulation. The second action is to edit the configuration, if a new configuration has been made, the "Edit" action will allow adjusting the existing parameters otherwise the "Edit" action will call the same setting window as "New" action but with all the values stated as None.

The parameters needed are initial state, random walk number, upper limit range, lower limit range and repeat time.



Initial state	
random number	
upper range	
lower range	
repeat time	

Figure **15.** Activated Configuration Dialogue by New Action



Initial state	5.0
random number	2
upper range	1.0
lower range	-1.0
repeat time	200

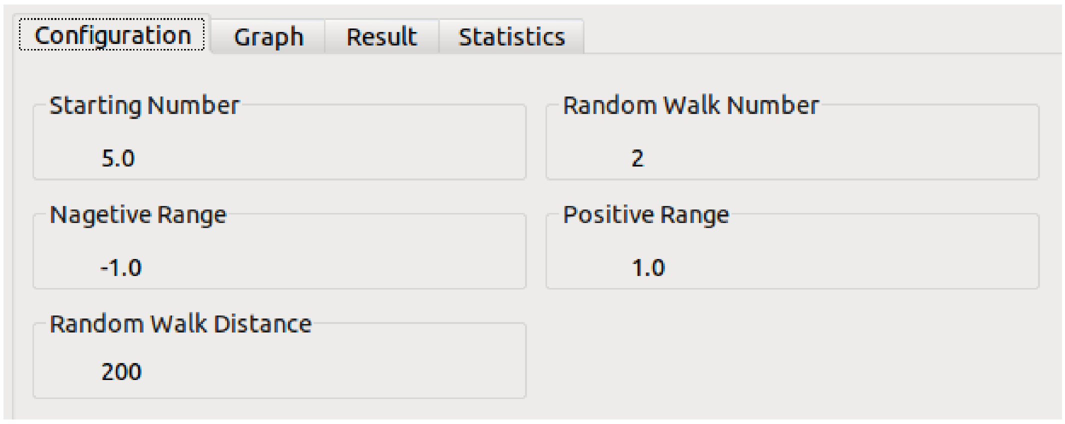
Figure **16.** Activated Configuration Dialogue by Edit Action

The third action “Run” is to run simulations, the fourth action “Stop” is to stop the running, if needed, and the last action “Clear” is applied to clearing the graph.

- Tab Widget

The tab widget is a combination of different tabs which are used to show the information of the simulation from a different point of view.

The first “Configuration” tab is a container which stores the configuration of the simulation. If you change the parameters in action, the values here will also be changed.

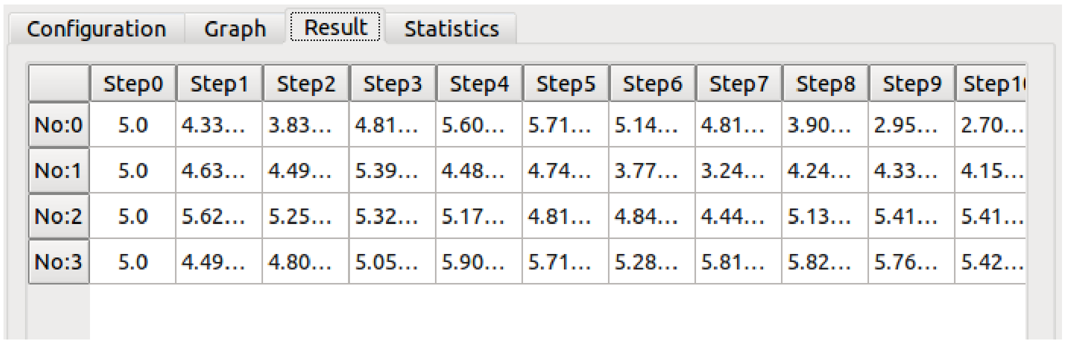


Starting Number: 5.0	Random Walk Number: 2
Nagetive Range: -1.0	Positive Range: 1.0
Random Walk Distance: 200	

Figure **17.** Configuration Tab

The second "Graph" tab is used to give a virtual representation of the random walk. When the simulation runs, the graph will plot the simulation step by step. If it is one dimensional plotting, then the x value of the simulation would be from zero to the maximum hamming distance.

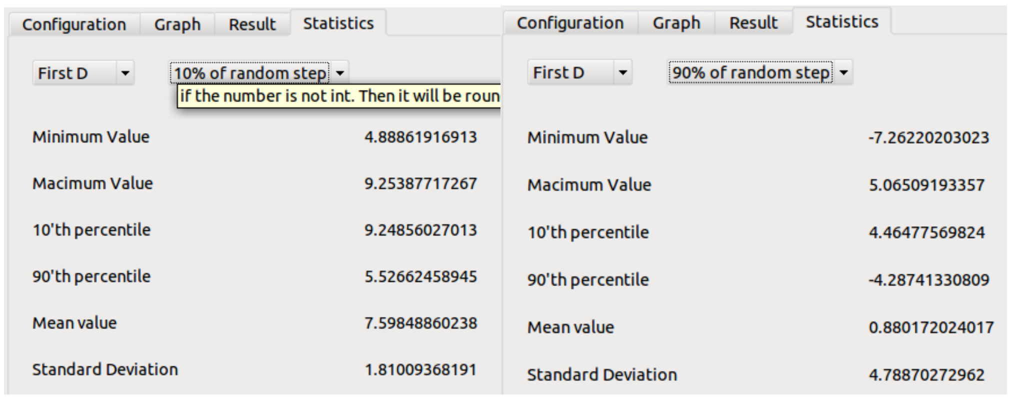
The third "Result" tab is a table which stores all the information of the simulation; we can check the result for each walker's each step. In this way, we would see the how it changes over time. If it is the one-dimensional simulation, the numbers of rows are the same as the walkers and if it is a two-dimensional walk simulation, it shows twice the number of the walkers as in total there are two times the number of walkers.



	Step0	Step1	Step2	Step3	Step4	Step5	Step6	Step7	Step8	Step9	Step1
No:0	5.0	4.33...	3.83...	4.81...	5.60...	5.71...	5.14...	4.81...	3.90...	2.95...	2.70...
No:1	5.0	4.63...	4.49...	5.39...	4.48...	4.74...	3.77...	3.24...	4.24...	4.33...	4.15...
No:2	5.0	5.62...	5.25...	5.32...	5.17...	4.81...	4.84...	4.44...	5.13...	5.41...	5.41...
No:3	5.0	4.49...	4.80...	5.05...	5.90...	5.71...	5.28...	5.81...	5.82...	5.76...	5.42...

Figure **18.** Two Dimensional Walk for Two Walkers.

The fourth “Statistics” tab is a tool to gather statistical information. It lists the minimum value, maximum value, 10th percentile value, 90th percentile value, mean value and the standard deviation for different stages of simulation and different walkers.



Minimum Value	4.88861916913
Macimum Value	9.25387717267
10'th percentile	9.24856027013
90'th percentile	5.52662458945
Mean value	7.59848860238
Standard Deviation	1.81009368191

Minimum Value	-7.26220203023
Macimum Value	5.06509193357
10'th percentile	4.46477569824
90'th percentile	-4.28741330809
Mean value	0.880172024017
Standard Deviation	4.78870272962

Figure **19.** Statistic Information of 10th and 90th Total Random Steps.

- Preferences

The preferences contain the settings for this algorithm simulator. In the current situation, it only has a selection of dimension.

3.2.4 Use Case Diagram

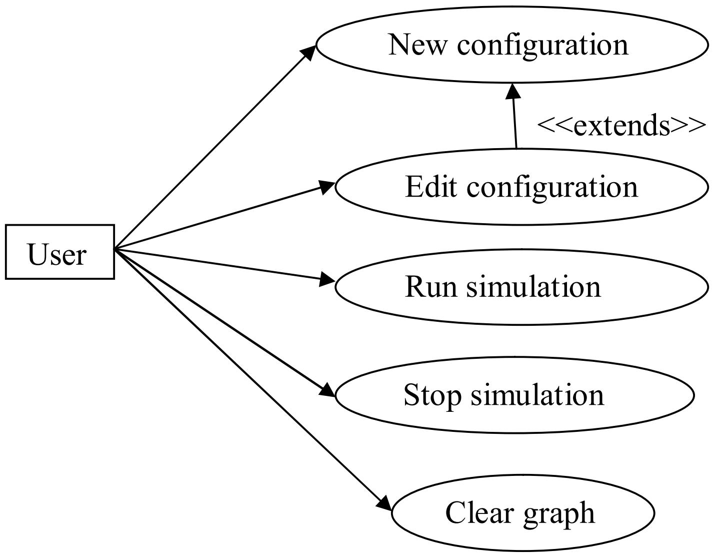


Figure **20.** the User Case Diagram for Random Walk Simulation.

3.2.5 Result of Testing

This section shows some testing of the random walk simulator with specific parameters.

The following two groups of graphs show the result of simulations under different situations. The x-axis and the y-axis do not have any unit as they only indicate how big steps each random walker has walked. The first group of graphs shows the undirected random walk and the second group shows the directed random walk. Both groups contain the one-dimensional and two- dimensional situations.

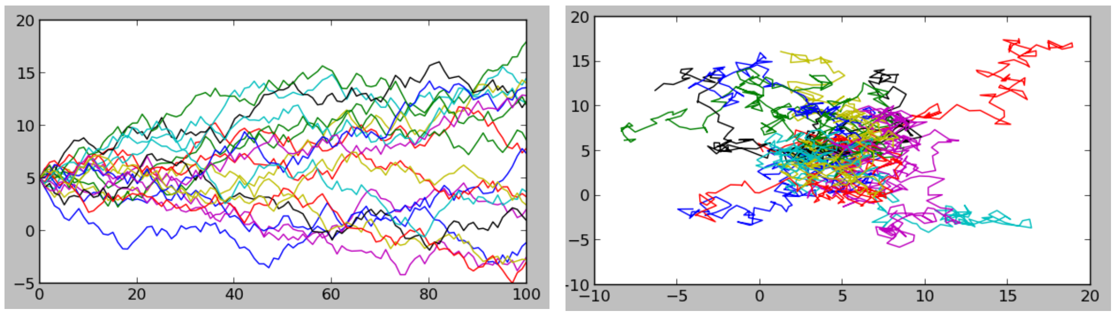
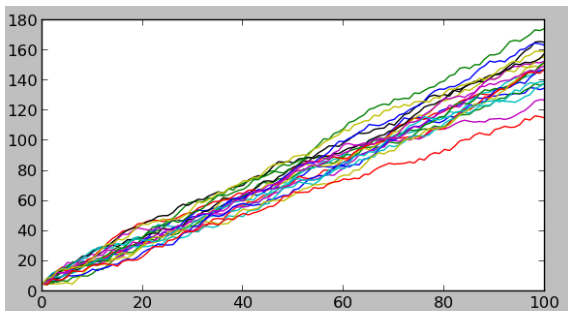


Figure **21.** 1D and 2D Undirected Random Walk

(X axies and Y axies are indicating the steps)

The undirected random walk basically means walking with a fair coin. During the process of random walk, the possibility of going to any direction is uniformly distributed. As a result, with a reasonable number of walks in a stage, the random walk can access almost any direction.



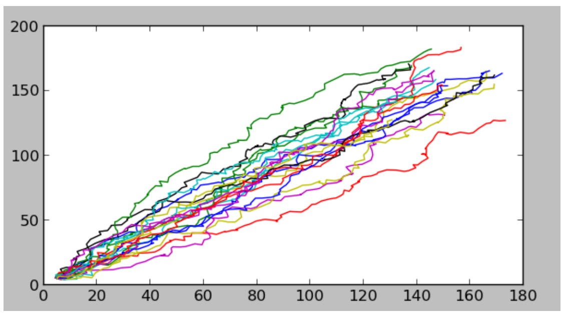


Figure **22.** 1D and 2D Directed Random Walk

(X axies and Y axies are indicating the steps)

The directed random walk is different from the undirected random walk.

From the view of stochastic process, directed random walk is influenced by a not-evenly-distributed chance. According to the formula provided by $E(S_n) = \sum_{j=1}^{n} E(Z_n)$, where $E(S_n)$means the expected value, the random walk, in this case, will be directed to a certain place.

From the view of field theory, the directed random walk can be also applied with an artificial potential field. These artificial potential fields will continually influence the random walk by generating a force on each step, and this force is generated according to the distance from goal to current position.

3.3 Parameters Monitoring Web Server

3.3.1 Introduction

During the development of the system, we also wanted to build the chain of information from the FPGA to a web server so that we can monitor the parameters in the embedded software.

3.3.2 Technical Introduction

- AJAX

AJAX (Asynchronous JavaScript and XML) is used for exchanging information on dynamical pages. AJAX allows web pages to be updated asynchronously by sending requests which contain a small amount of data to the server. The basic form is by using JavaScript functions and HTTP methods.

- FPGA

FPGA (Field-programmable Gate Array) is an integrated circuit designed as general hardware which can be configured latter. The language of making configuration is named as hardware description language (HDL).

- SPI

Serial Peripheral Interface Bus (SPI) is a synchronous serial data link standard. It was first announced by Motorola which operates in full duplex mode. The SPI connection is normally used under the situation of communicating between two different hardware.

- Highcharts

Highcharts is a JavaScript library designed for fast implementation of graphs on the web pages. It provides rich APIs to generate an interactive, dynamic chart.

- JSP

Java Server Pages (JSP) is a technology which helps the software developer to create dynamically generated web pages. The system is based on XML, HTML etc. The back-end is taken care by Java programming language.

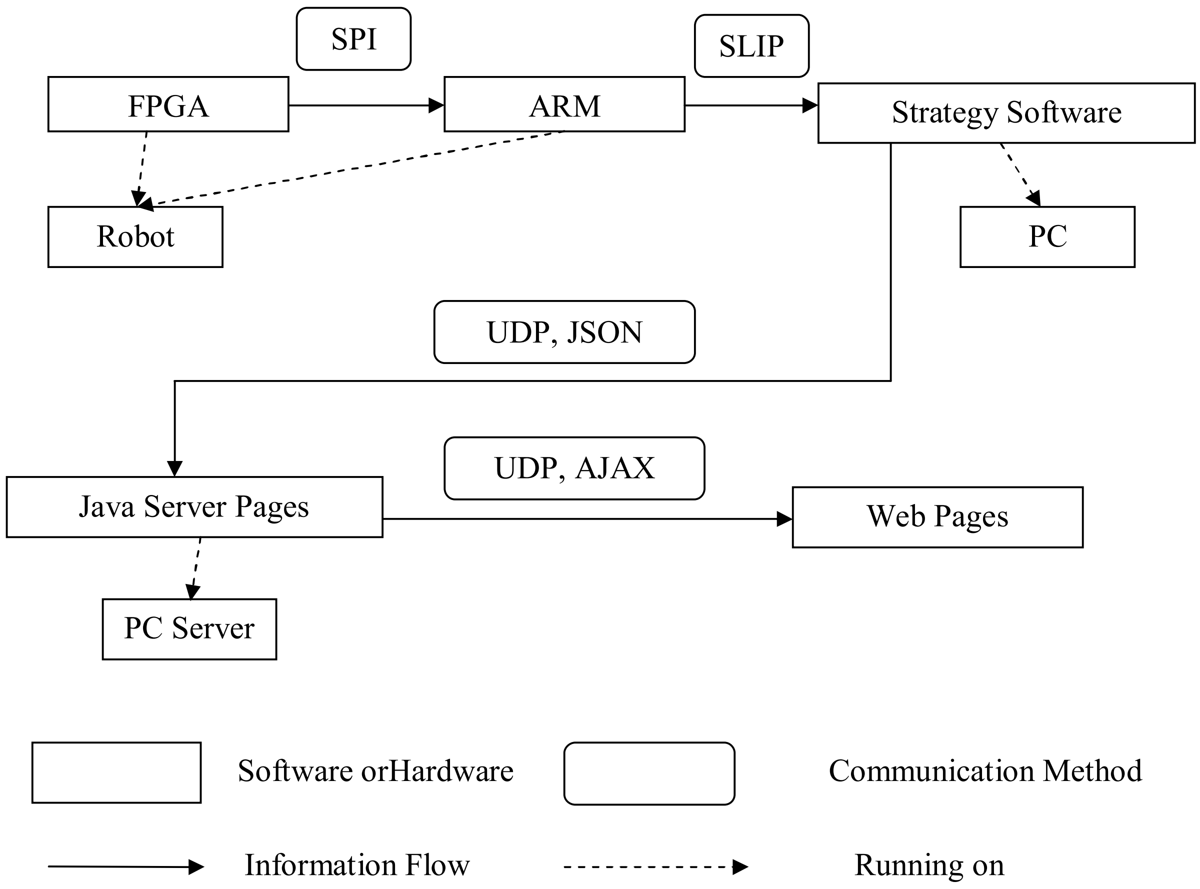
3.3.3 Motivations

There are two main motivations of making this web server service.

The first is that as we noticed that the Matlib's 2012b version can automatically generate code for embedded software based on the mathematical model, we would like to use a monitoring service to test correctness of the mathematical models on motor, etc.

The second reason is that in some case, we need to know what problem caused the system to fail. According to the experiences, the failure can be caused by the flaws lying in voltage control or current control functions of embedded software. Therefore, a tracking mechanism is essential for locating the cause of different failures.

3.3.4 System Structure



The system's structure is as represented above. Firstly we collect the information from FPGA. The information is mainly related to RPM (Revolutions per Minute) of each wheel. After this, the information goes from the FPGA to ARM through the SPI channel. Then the web_data_module.cpp of strategy software will take care of receiving from ARM and send to a JSP server on another computer through UPD broadcasting. The information at this stage is in the form of Google protobuf. As soon as the information has been sent to the JSP server, the serve stores the information in an instance and then wait it to be forwarded by quest from browser.

3.3.5 User Interface

The user interface is designed for flexible usage of checking information from the multi-robot system. We used highcharts, which is a JavaScript library, to develop the interface. According to the number of parameters, we show the number of charts on the graph. For each parameter, the appearances are identical to each other. For any chart, the appearance is as follows:

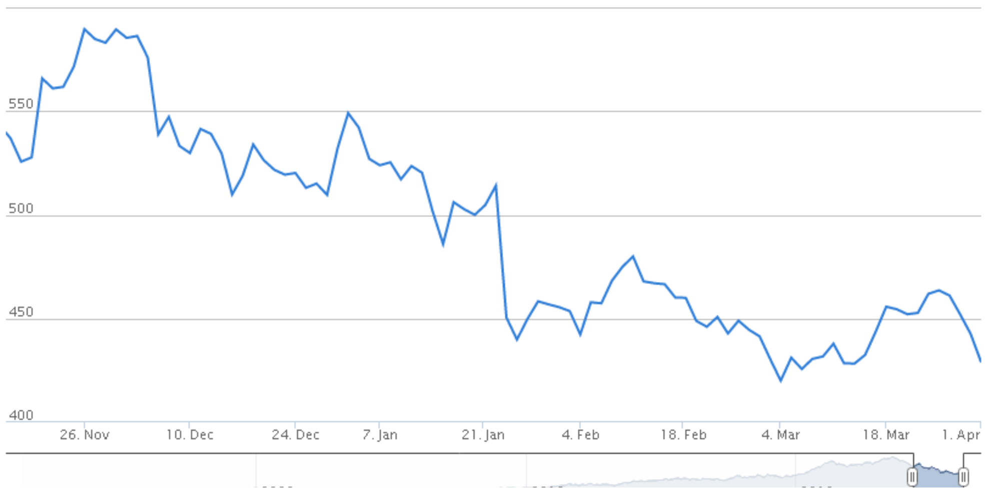


Figure **23.** Monitoring Graph

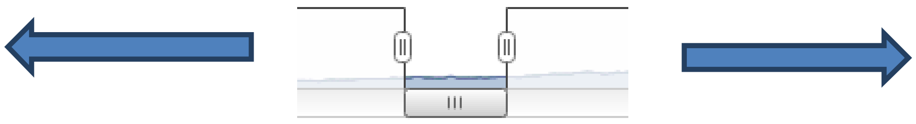


Figure **24.** Slider of Monitoring Graph Monitoring

The upper side is a normal graph with interactive functions. The user can point the cursor to the any point on the graph and it will show the information of the point on the web page. The lower side is a drag window. The user can specify the range that they would like to use for checking the status of the information

3.4 Vision Simulation Software (Python)

3.4.1 Introduction

The vision simulation software was initially designed for replacing the vision system. The vision software is able to receive the information from the strategy software and represent vividly on its simulation area. Moreover it can also allow users to instantly modify the information of robots and send the status of robot to strategy software. After we found the way of porting the simulation from windows platform to Linux platform, this project has been suspended. Nevertheless, it is still a reasonable platform for some other future implementation.

3.4.2 User Interface

The interface of vision simulation software is designed simply for representing the robot on the graph.

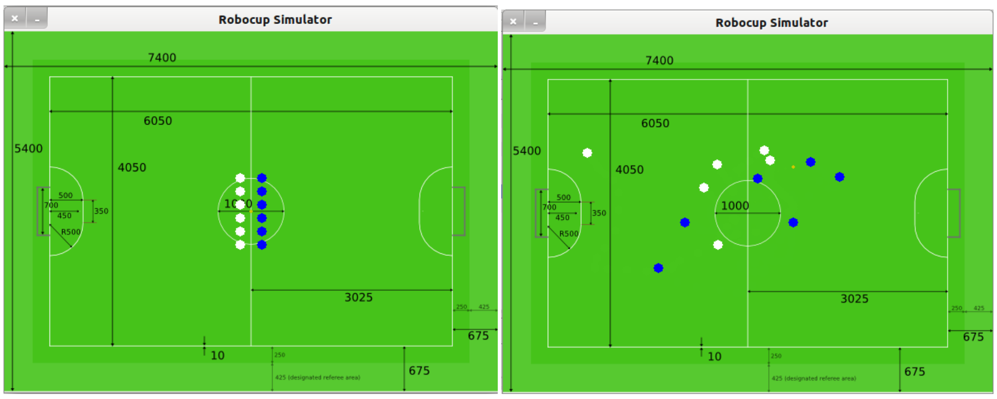


Figure **25.** User Interface of Vision Simulator

The simulator is similar to the C++ version of vision simulator, and the only difference is that the broadcasting function has not been implemented.

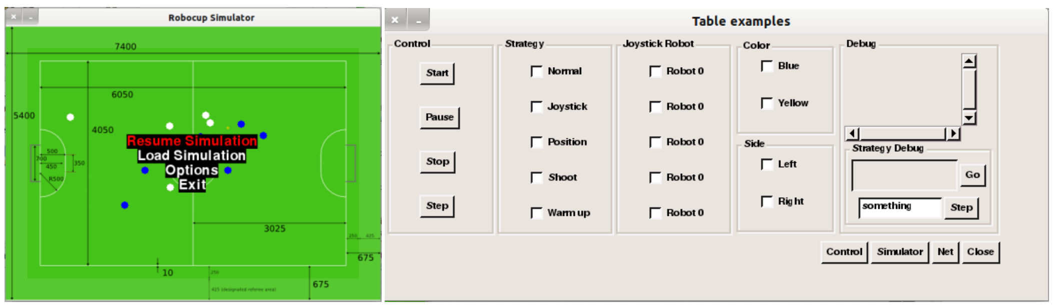


Figure **26.** Setting Panel and Options

When the setting panel is called, the software will be four options in the middle of the screen. They are resume simulation, load simulation, options and exit respectively. Among all of these selections, the “option” will show another setting window to ask the user to specify the environment of the simulation.

3.4.3 Use Case Diagram

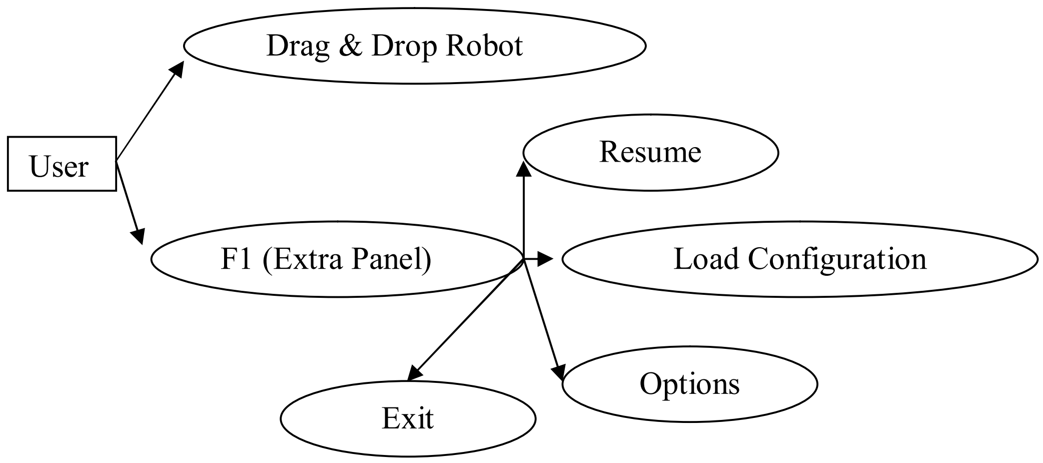


Figure **27.** the User Case Diagram for Vision Simulation Software

3.5 Rapidly Random Exploring Tree Simulator (Python)

3.5.1 Introduction

This simulation software is used to virtualize the process of path-planning. At the moment, it had only implemented the RRT algorithms

3.5.2 Technical Introduction

- Pygame

Pygame is a set of Python modules designed for writing games. Pygame adds functionality on top of the excellent SDL library. This allows you to create fully featured games and multimedia programs in the python language. Pygame is highly portable and runs on nearly every platform and operating system. Pygame itself has been downloaded millions of times, and has had millions of visits to its website. [6]

- NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. [7]

- SciPy

SciPy (pronounced "Sigh Pie") is open-source software for mathematics, science, and engineering. It is also the name of a very popular conference on scientific programming with Python. The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. The SciPy library is built to work with NumPy arrays, and provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization. Together, they run on all popular operating systems, are quick to install, and are free of charge. NumPy and SciPy are easy to use, but powerful enough to be depended upon by some of the world's leading scientists and engineers. [8]

3.5.3 Advantages and Disadvantages

Comparing with the C++ version that we built before, this python version is faster on the development. The implementation has no more than 100 lines of code for one algorithm. Therefore it is very suitable for fast prototyping. When you have an interesting idea, you can immediately build a system to test.

It also has some disadvantages. The most noticeable is that it cannot be implemented for a situation which needs fast running speed. The reason lies in that the APIs in python are sufficient, but they are not transparent to the users. In this case, it is very hard to arrange a test by fixing other factors which may influence the speed.

3.5.4 The Test on RRT

The following graph shows the how the simulator simulates the process of RRT algorithm.

The maximum point on the graph has been set to 5000. The following pictures have been recorded at four different moments.

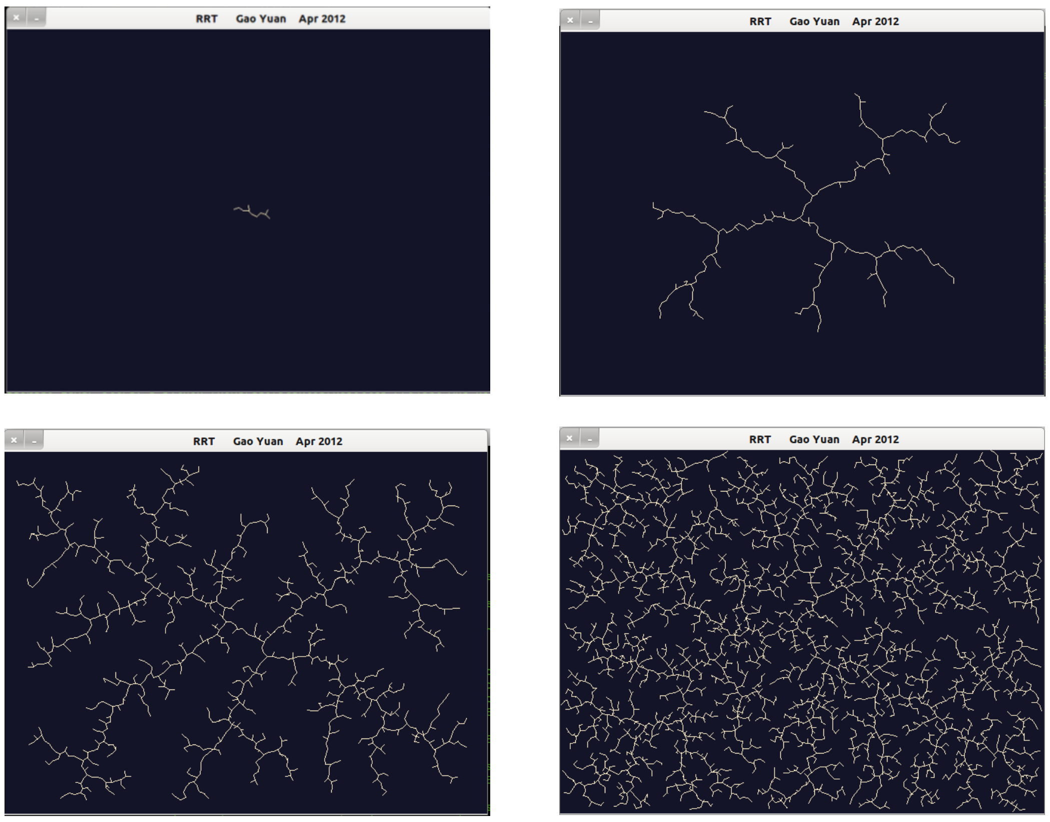


Figure **28.** Four Different Moments of RRT Expanding Process

3.6 Path-planning Simulator

3.6.1 Introduction

This simulation software is used to virtualize the process of path-planning. Comparing with the python RRT simulator, this C++ simulator is more suitable to the serious testing purpose. As we use the standard libraries of Qt to develop this software, the same data structure and implementation mechanisms in Qt can enable us to give detailed analysis of the different algorithms.

3.6.2 Technical Introduction

- Qt Creator

Qt Creator is a cross-platform integrated development environment (IDE) tailored to the needs of Qt developers. It provides:

- C++ and JavaScript code editor
- Integrated UI designer
- Project and build management tools
- gdb and CDB debuggers
- Support for version control
- Simulator for mobile UIs
- Support for desktop and mobile targets

[9]

3.6.3 Implemented Algorithms

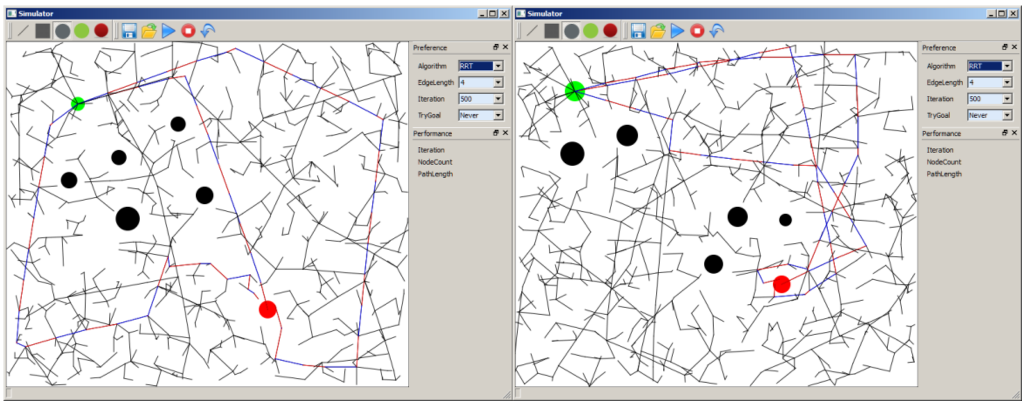
There are two algorithms have been implemented. The first one is definitely the most important algorithm RRT. Then we have implemented the RRT start algorithm developed by two MIT professors - Sertac Karam and Emilio Frazzoli.

We actually have a list of algorithms to be hopefully implemented later. These algorithms can be very suitable for studying purpose in the field of sampling-based path-planning algorithms.

RRT	ERRT
BRRT	DRRT
RRT plus	RRT start
A *	D *
PRM	PRM *

Figure **29.** Possible Set of Algorithms to be implemented

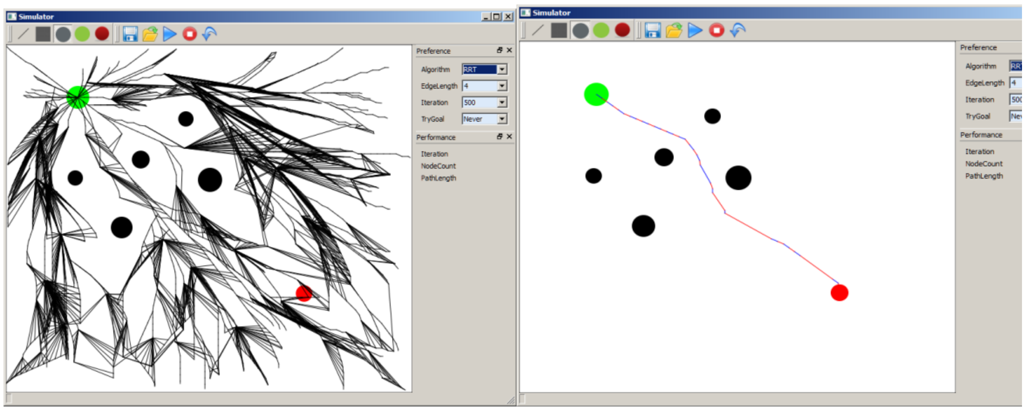
3.6.4 User Interface



Preference
Algorithm RRT
EdgeLength 4
Iteration 500
TryGoal Never
Performance
Iteration
NodeCount
PathLength

Preference
Algorithm RRT
EdgeLength 4
Iteration 500
TryGoal Never
Performance
Iteration
NodeCount
PathLength

Figure **30.** the Unser Interface of RRT Testing



Preference
Algorithm RRT
EdgeLength 4
Iteration 500
TryGoal Never
Performance
Iteration
NodeCount
PathLength

Preference
Algorithm
EdgeLength 4
Iteration
TryGoal
Performance
Iteration
NodeCount
PathLength

Figure **31.** the User Interface of RRT star Testing

4 STRATEGY SOFTWARE

4.1 Introduction of Software

4.1.1 General Introduction

Strategy software is AI software. It offers a set of information management mechanisms, utilities and services to help a multi-robot system to automatically self-organize.

Structure of strategy software was designed for smooth and stable computation on Linux platform. Based on the previous researches, the Linux platform can provide more accurate time functions. This property of the Linux platform is essential for a system that is sensitive to communicational environment.

In our robot soccer project, the strategy software needs to communicate with a high speed camera. This condition requires the strategy software to make a decision within 1/60 second. The instability of the strategy software, which we previously built on windows, leaded us to search for a solution to ensure that the time error is limited to a small value.

During the period of building this version of software, we have to admit that the work of the previous developer actually influenced us a lot. After checking several papers and talking to some experts, we decided to develop the new version of strategy software based on technical tools that we know.

4.1.2 Main Motivations

There are three main motivations for designing this software.

The first motivations what previously mentioned - timing issues. This motivation is probably the most important reason.

The second one is to test system can be as similar as the human brain’s structure. In fact, the strategy software was developed according to the structure of how human process the information in an intellective way. When we wanted to design a structure to show how I can reduce the coupling of each module, the way of how human intelligence functions gave me a hint. After I considered the connection between the requirements of the program and natural intelligence, we decided to organize the software separately as in the process of human thinking.

The third one is to build a platform which can really test our algorithms on. The performance of planning algorithm, navigation algorithm and controller are essential to the evaluation of the strategy software. Although the most of the algorithms have been de facto since almost they were first invented, there are still several problems to notice when applying them to multi-robot system.

4.1.3 Development Process

In the 1960s, the software was first called as art-craft by Knuth. Nowadays, we cannot simply consider designing software as a person's heroism. In fact, we need to establish an engineering way crafting a complicated software patterns.

According to the classic methodology of software engineering, there are several ways to control the process of software development. A typical OOP (object oriented programming) way is stated as follows:

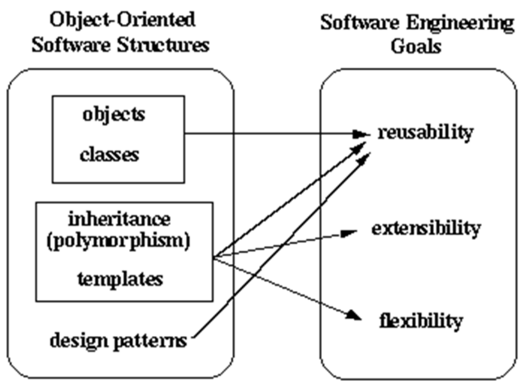

Object-Oriented Software Structures
Software Engineering Goals
objects
classes
inheritance (polymorphism)
templates
design patterns
reusability
extensibility
flexibility

Figure **32.** the Development Process of Object-oriented Programming [10]

Although we have followed the proper development process, the real situation is different. The fact turned out iterative and incremental development is more suitable than the traditional process (waterfall model) as the most of the time we are not clear about the whole requirements.

At every stage, we had to face the fact that we need to design a small part of the software and analyses of the needs. And then after that we had to focus on implementing the project and communication. This whole process helped us to go through the project quickly and effectively.

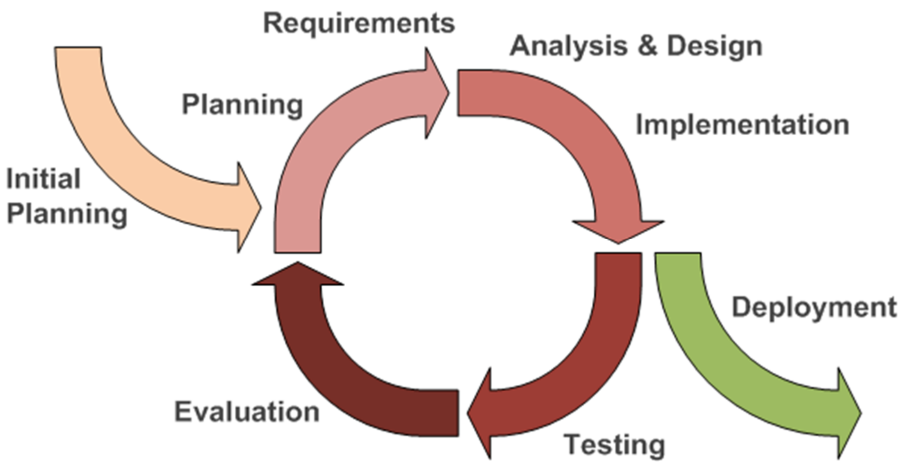


Figure **33.** the Iterative Development Process [11]

A typical example is that we formed a small team to develop a communication channel function in the software. And then after that we worked as tight as we can, communicated as frequently as we can and optimized the APIs (application programming interface) as we can. Finally the outcome software is quite sharp and the time of development was not long at all.

As a consequence, I would like to suggest the people who work or will be working on the project to try this method. Especially this method is strongly emphasized by the software engineering community under the nowadays' condition.

4.1.4 System Structure Design

1. File Hierarchical Structure

As previously mentioned, the file structure follows the pattern of the nature intelligence.

From the neurobiology, cerebrum contains four main areas frontal, temporal, occipital, and parietal which are roughly recognized as language, memory, and voice and vision functional centre.

The design of the software, to some extent, follows the structure of natural intelligence. The following is the relationship between the file clusters of the software:

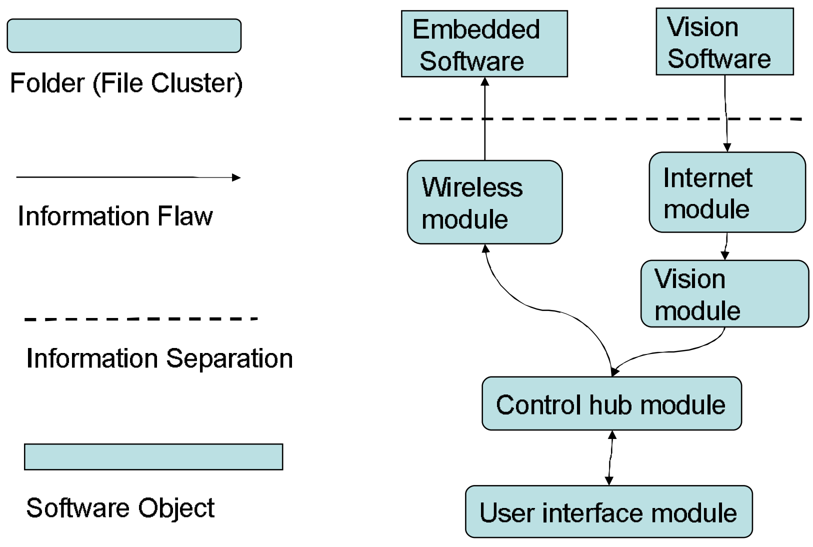


Figure **34.** Modules of Strategy Software

4.1.5 Logic Structure Design

In this system there are several server-client models to consider about. No matter whether there is a callback function offered by the OS (operating system), it is better to use concurrent structure to deal with the message receiving and sending. Although the concurrent programming includes process, I/O (input/output) multiplexing and threading, however, in this thesis, the concurrent programming here particularly means threading.

Strongly influenced by a philosophy that brain's different parts processing information interactively, the software, on one hand, is designed differently from the design of the file structure, on another hand, the functions are still limited to their physical locations. (It is just like the brain, isn't it?).

The relationships between the threads are shown below:

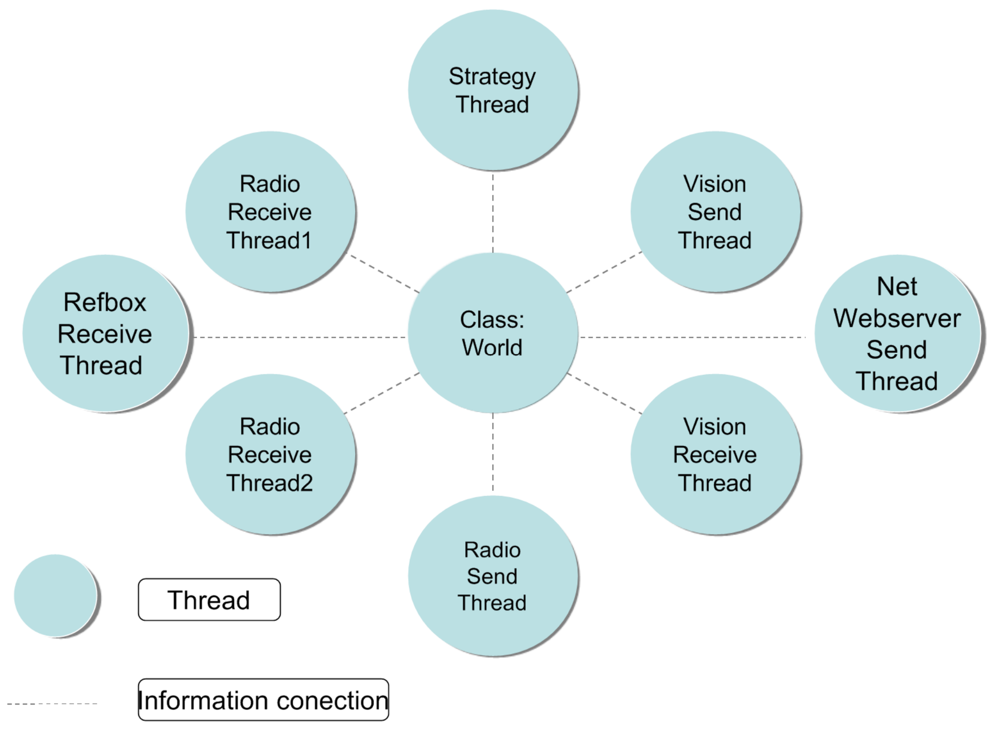


Figure **35.** Logic Structure of Strategy Software

The code example is shown below:

```
// various threading in header file
StrategyThread* strategy_thread_;
VisionSendThread* vision_send_thread_;
VisionReceiveThread* vision_receive_thread_;
RadioSendThread* radio_send_thread_;
RadioReceiveThread* radio_receive_thread0_;
RadioReceiveThread* radio_receive_thread1_;
RefboxReceiveThread* refbox_receive_thread_;
NetWebserverSendThread* net_webserver_send_thread_;
```

Figure **36.** Code Example of Thread Implementation

As shown in the previous graph, a concurrent programming method is applied to the development of the software.

- Information Structure Design

The previous section describes the structure picture of the software. This section introduces the general information flow between modules. Latter we will talk about the information protocol and data structure.
The whole system has two physical media. One of the media is the air and another one is cable. These two channels formed communication infrastructure of the strategy software. And after that, we established protocols based on existing technology and then specified them according to our need.

4.1.6 Guide to Thesis

The previous section describes the structure of the software from different points of view including the structure of file orientation, the structure of the logic orientation and the information structure. The following sections will first introduce the system based on following orders:

1. Communication Data Type
2. Interface Module
3. Wireless Module
4. Internet Module
5. Vision Module
6. Control Hub Module

4.2 Communication Data Type

4.2.1 Introduction

The information is needed for communication. No matter it is for sending calculated information or for monitoring, it can always be proven to be subtle in the system as they can ensure that the time of transformation is limited to a small range.

4.2.2 Technical Introduction

- JSON

JSON (JavaScript Object Notation) is a text-based open standard for human-readable information exchange. It uses several data structures in JavaScript language to represent relationships between different object. But in reality, it is suitable for many languages. Different language has different libraries to parse the structure and almost all APIs are efficient and developer-friendly. Comparing with XML (Extensible Markup Language), it is faster and more readable. In C++, the parser library we used QJson and in Java, we used Gson.

JSON's basic types are:

Number: (double precision floating-point format in JavaScript, generally depends on implementation)

String: (double-quoted Unicode, with backslash escaping)

Boolean: (true or false)

Array: (an ordered sequence of values, comma-separated and enclosed in square brackets; the values do not need to be of the same type)

Object: (an unordered collection of key: value pairs with the ':' character separating the key and the value, comma-separated and enclosed in curly braces; the keys must be strings and should be distinct from each other)

Null: (empty)

[12]

- QJson

QJson is a Qt-based library that maps JSON data to QVariant objects.

JSON arrays will be mapped to QVariantList instances, while JSON's objects will be mapped to QVariantMap. [13]

- Gson

Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of. [14]

- Google Protocol Buffer

GoogleProtocol Buffer (In short, Google protobuf) is a standard introduced by Google. As a binary coded communication method, it is much faster than any readable format. But it is also a little bit difficult to understand. The RPC (remote procedure call) communications between different agents are critical and as a result of many researches and selections, we found it is very convenient to let several software agents (vision software, strategy software and reference box software) to interchange information.

The Google protobuf stores information in a .proto file and it uses a separate compile to change the .proto file into a specified language. (E.g. C++, Java and Python)

When it runs, the special compiler provides the setters and getters for the data specified in .proto file. When you import the auto-generated file which includes API, you can use the setters and getters accordingly.

- SLIP

SLIP (serial line inter protocol) is protocol first used to transfer information through the serial port using internet protocol. In this project, we use it as a secure and error-free way of transforming information from the robot.

There are four bytes which indicating different meanings of information in SLIP. A typical configuration could be END, ESC, ESC_END and ESC_ESC. And normally the values of these four kinds of bytes are 0300, 0333, 0334 and 0335 accordingly. [15]

According to the rule, we should first send an initial END character to flush out any data that may have accumulated in the receiver due to line noise. Then for each byte in the packet, send the appropriate character sequence, if it's the same code as an END character, we send a special two character code so as not to make the receiver think we sent an END, if it's the same code as an ESC character, we send a special two character code so as not to make the receiver think we sent an ESC. Otherwise, we just send the character. At the end, we send END to tell the receiver that we're done sending the packet. For the improvement, we add a CRC check after all the bytes.

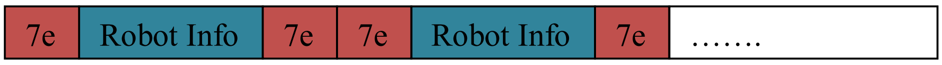
The structure of SLIP in C is listed below:

```
#define END 0300/*c0*/
/*indicates byte stuffing*/
#define ESC 0333/*0xdb*/
/*ESC ESC_END means END data byte*/
#define ESC_END 0334/*0xdc*/
/*ESC ESC_ESC means ESC data byte*/
```

```
#define ESC_ESC 0335/*0xdd*/
```

- Transparent Protocol

The transparent protocol is the first protocol that we used to communicate with the robot. The data structure is comparatively simple and easy to understand. The encapsulation of the package starts with hex number 0x7e and also ends with the same number. Between these two numbers, the information of robot is contained.



The other part of this information structure is robot info, and this part is implemented by using a struct in C/C++ language. The following graphs show the detail of information contained in the codes.

Code Example:

```
typedef struct
{
int index;
int x_velocity;
int y_velocity;
float total_velocity;
int kick;
int dribble;
int chipkick;
int rotate_velocity;
   RobotType robot_type;
}RobotParamters;
```

The total length of the package is determined by the number of the robots in a team. When a robot receives the information wirelessly, it will filter out the information which does not correspond to its robot ID. After that process, the embedded software on the robot will pick each piece of information out from the data package.

- DHCP

Dynamic Host Cogeneration Protocol (DHCP) is a network protocol which is used to configure the device on a network. This protocol is used in the project as the second version protocol for sending the information to robot including the instant velocity, instant direction and instant.

- UDP

User Datagram Protocol (UDP) is widely used unidirectional transmission protocol. It is a core function of Internet Protocol (IP) suit, and it is a very useful protocol if we want to save the time for transmission. In fact, as in our project, many time-critical applications use the UDP for providing fast broadcasting service.

- DECT

Digital Enhanced Cordless Telecommunications (DECT) is a standard used in digital telecommunication system. Unlike the GSM standards, it does not specify any internal aspects of the fixed network itself. Connectivity to the fixed network (which may be of many different kinds) is done through a base station or "Radio Fixed Part" to terminate the radio link, and a gateway to connect calls to the fixed network.

4.2.3 Introduction to Channels

The channels include the air and the cable. Both ways are essential to this system. Wireless is chosen because the Multi-robot system needs to communicate with the robot and the cable is used to receive information from the vision software which is sensitive and time critical.

The relationships between the media and protocol structures are listed below:

4.2.4 Wireless Channel

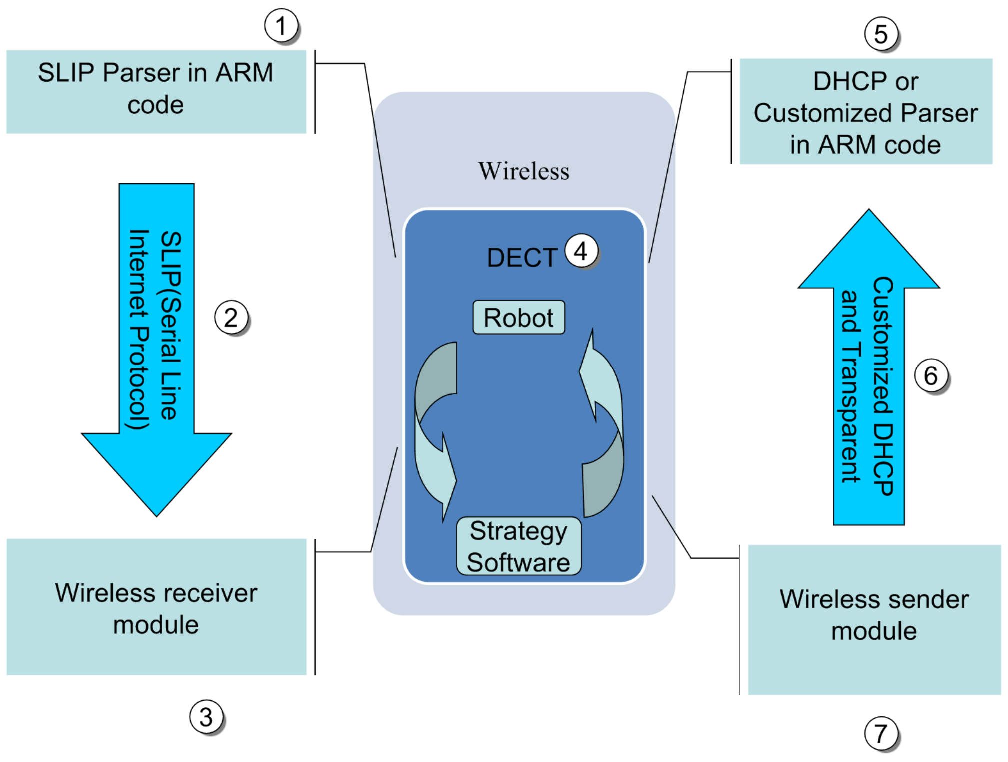


Figure **37.** the Communication Structure of Wireless Channel

The structure of wireless is designed for fast communication between robots and the server. The sent information is the command for the robots. Here we introduce the graph term by term.

The first, second and third terms, which has marks ①, ② and ③ on the graph respectively, represent the connection from robot to strategy software using SLIP (Serial Line Internet Protocol).

The SLIP protocol is used here to ensure the transformation to be efficient and almost error free. The original idea was to transfer the files according to the internet protocol through serial lines. For the information of original specification, please see the appendix.

By considering the hardware configurations, we followed the idea of encapsulation of the SLIP protocol in this project.

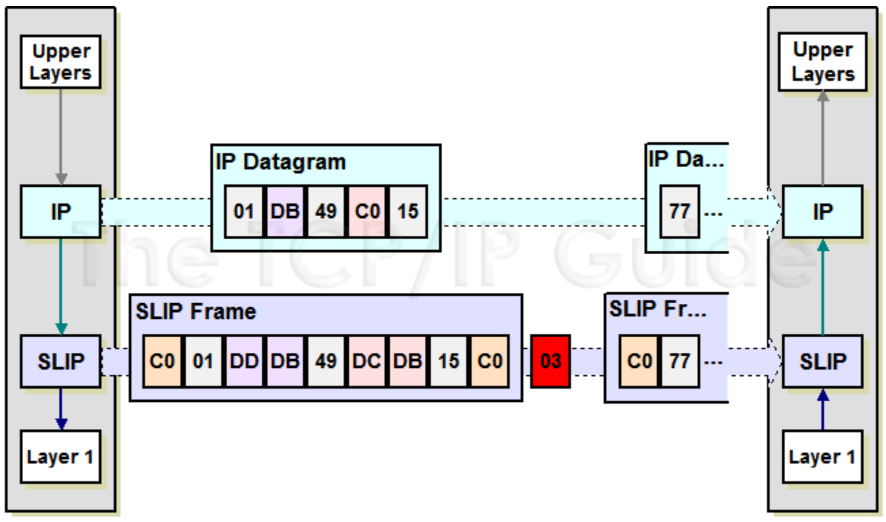


Figure **38.** Structure of SLIP Structure [16]

Both parsers in the ARM code and in strategy software use automata to read through the information package and encode the information into the command using specified bytes with a CRC (cyclic redundancy check). As a consequence, we can determine which information is sent from a certain robot.

The fourth part is the showing the physical layer of the information transformation. For convenience, we used DECT (digital enhanced cordless telecommunications) structure to find a solution.

DECT is firstly originated in Europe. And now it has become a universal standard replacing the earlier standards, such as 900 MHz CT1 and CT2. The detailed description can be found in the appendix.

The fifth, sixth, seventh part shows two kinds of protocol we used to implement the connection from the computer to the robot. During the development process, there were two protocols considered.

The simpler one is transparent protocol. In this protocol, we assemble all the information as a serial command line. After sending through the serial port, when all robots received the command, each robot selects the particular part of the information that it needs and, at the same time, filters all redundant information. The detailed article about this protocol can be found in the appendix.

Another one is the DHCP (dynamic host configuration protocol). According to the description of the standards, it is originally used to configure devices which are connected to a network. We noticed the coding scheme is pretty suitable for our project. We implemented this protocol in the strategy software as well. The detailed article about this protocol can be found in the appendix.

4.2.5 Internet Channel

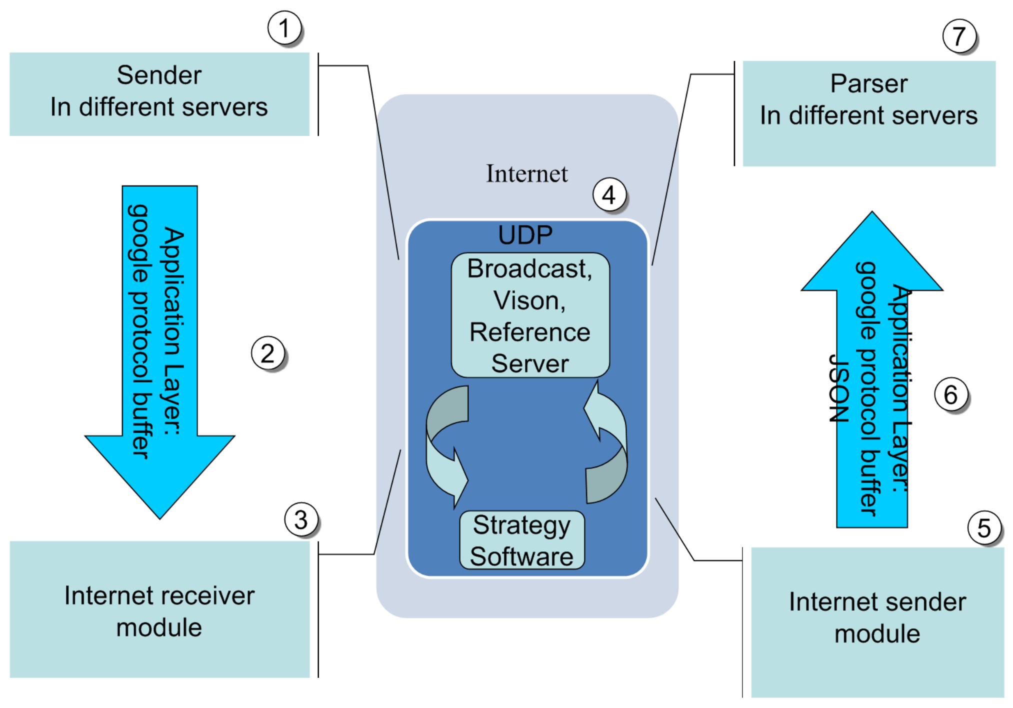


Figure **39.** the Communication Structure of Wireless Channel

The connection between the strategy software and broadcast server, the strategy software and vision software, the strategy software and the reference server are much more difficult than the wireless connection.

The first reason is that the internet connection has more layers than the wireless connection layers. This means the developers should pay more attentions on the protocols than algorithms. The second reason is that although the structures from physical layer to the transport layer are the same, the protocols in application layers are different. We will describe all of them individually in following context.

The label 5,6 and 7 represent the connection from strategy software to broadcast server is built up using UDP protocol. After receiving information from the robots, a thread NetWebserverSendThread will forward the information to a JSP server.

As a consequence, we will be able to monitor the statistics not only on the web page but also on the mobile phone.

In the connection, the information was first collected by the ARM embedded software. The structure is as follows:

Code example

```
struct {
unsigned short cell_voltage
unsigned short capacitor_voltage
unsigned short kicking_voltage
unsigned short  current_level[5];
unsigned int number_of_package
}robot_running_info
```

Figure **40.** the Code Example of Json Structure

Then the protocol of the second side of the application layer is JSON. But the information that needs to be transformed contains following information in Java:

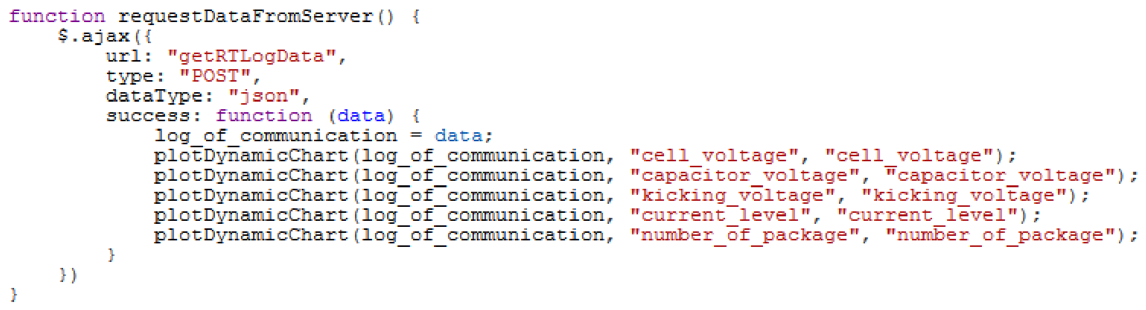
Code example

```
DOUBLE cell_voltage
DOUBLE capacitor_voltage
DOUBLE kicking_voltage
DOUBLE  current_level
DOUBLE number_of_package
```

Figure **41.** the Code Example of Data Type in Java

Based on the definition of JSON structure, we can transform previous parameters into following JSON code:

Code example



```
function requestDataFromServer() {
    $.ajax({
        url: "getRTLogData",
        type: "POST",
        dataType: "json",
        success: function (data) {
            log_of_communication = data;
            plotDynamicChart(log_of_communication, "cell_voltage", "cell_voltage");
            plotDynamicChart(log_of_communication, "capacitor_voltage", "capacitor_voltage");
            plotDynamicChart(log_of_communication, "kicking_voltage", "kicking_voltage");
            plotDynamicChart(log_of_communication, "current_level", "current_level");
            plotDynamicChart(log_of_communication, "number_of_package", "number_of_package");
        }
    })
}
```

Figure **42.** the Code Example of AJAX JavaScript Function

The label 1, 2 and 3 represent the information flaw from the vision server and reference box to the strategy software. Although the UDP is the transport layer protocol, the application layer uses Googleprotobuf as the container to store the information. The basic idea is to assure the efficiency of the communication.

In the.proto file, the Google protobuf specifies the information sent from the vision system to the strategy software. It is assembled from parts; the first part is SSL_GeometryFieldSize which specifies the size of the field, the second part SSL_GeometryCameraCalibration is used for storing information of the calibration parameters. The third part called SSL_GeometryData is used to encapsulate the previous two parts into one for convenience of transforming. The parameters contained in the data are very self-illustrative. The detailed structure is listed below:

Code example

```
message SSL_GeometryFieldSize {
required int32 line_width = 1;
required int32 field_length = 2;
required int32 field_width = 3;
required int32 boundary_width = 4;
required int32 referee_width = 5;
required int32 goal_width = 6;
required int32 goal_depth = 7;
required int32 goal_wall_width = 8;
required int32 center_circle_radius = 9;
required int32 defense_radius = 10;
required int32 defense_stretch = 11;
required int32 free_kick_from_defense_dist = 12;
required int32 penalty_spot_from_field_line_dist = 13;
required int32 penalty_line_from_spot_dist = 14;
}

message SSL_GeometryCameraCalibration {
required uint32 camera_id     = 1;
required float focal_length = 2;
required float principal_point_x = 3;
required float principal_point_y = 4;
required float distortion = 5;
required float q0 = 6;
required float q1 = 7;
required float q2 = 8;
required float q3 = 9;
required float tx = 10;
required float ty = 11;
required float tz = 12;
optional float derived_camera_world_tx = 13;
optional float derived_camera_world_ty = 14;
optional float derived_camera_world_tz = 15;
}

message SSL_GeometryData {
required SSL_GeometryFieldSize field = 1;
repeated SSL_GeometryCameraCalibration calib = 2;
}
```

Figure **43.** the Code Example of Vision Information Exchange Structure

4.3 Interface Module

4.3.1 Introduction

The appearance of the user interface was designed to have better simulation and control interaction. Besides the menu, there are two parts in the window. The first part is the control panel which has buttons and group boxes for each selection.

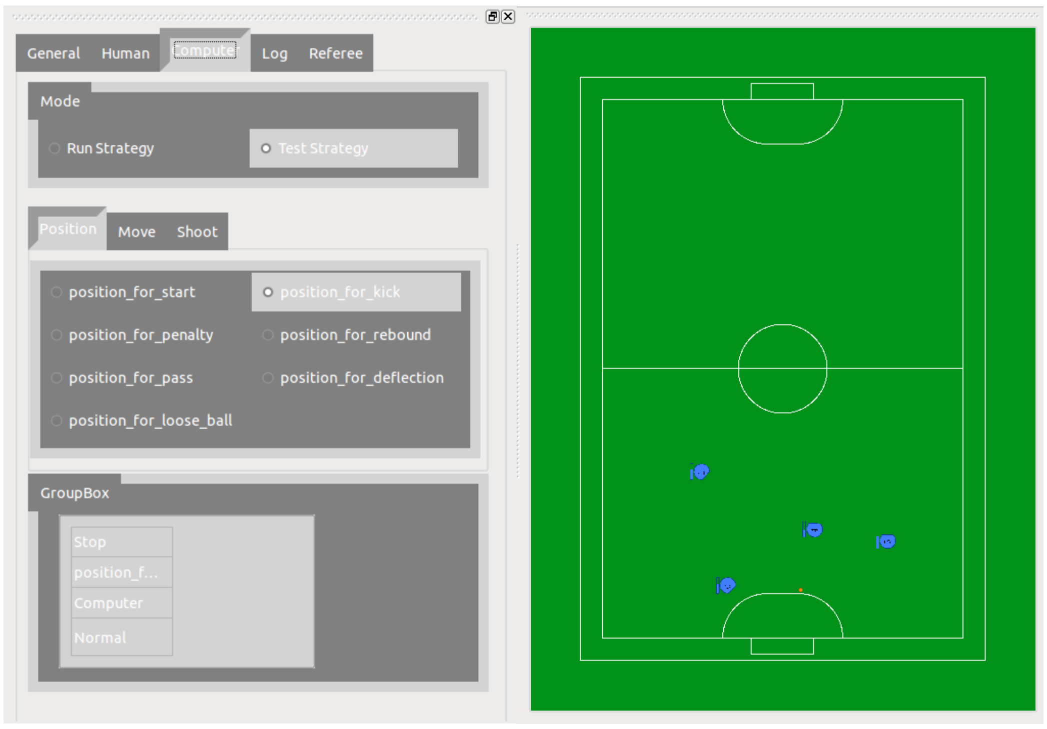


Figure **44.** the User Interface of Strategy Software

The control panel is separated into five parts. From the graph we can see that they are general, human, computer, log and referee. The first section “general” is taking care of the selection of game status, control source, colour selection, side selection and robot selection. The second section “human” specifies the robot human can control. The third section “computer” specifies the mode and strategy of the strategy software. The fourth section specifies the log information of the strategy software. Finally the last section specifies the information sent from the referee.

The simulated field is a place in which robots and balls can be represented in the window. When the strategy software is running, the information sent from the vision software will be analysed and visualized in this field. The robots will have different colour as either red or blue and the ball will be yellow. All the objects in the field can be selected.

By combining these two main areas, the strategy software can control and have a feedback view of the system. This ensures the system to have strong robustness.

Technical introduction:

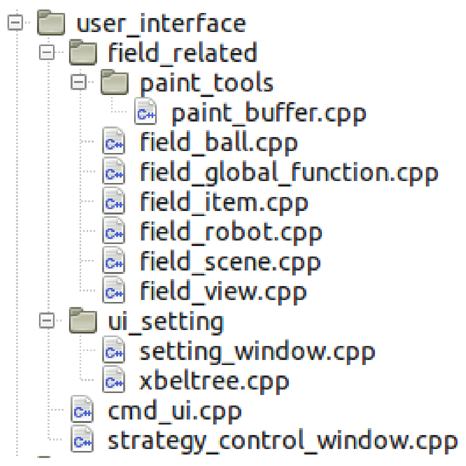
- Qt Designer

The user interface was designed using the Qt designer. Qt designer is Qt's tool for designing and building graphical interface. It contains a full tool chain of developing interfaces under Qt's environment.

- Qt Style Sheet

Qt Style Sheets (QSS) are a powerful mechanism that allows people to customize the appearance of widgets, in addition to what is already possible by subclassing QStyle. The concepts, terminology, and syntax of Qt Style Sheets are heavily inspired by HTML Cascading Style Sheets (CSS) but adapted to the world of widgets.

4.3.2 File Structure



```
user_interface
    field_related
        paint_tools
            paint_buffer.cpp
        field_ball.cpp
        field_global_function.cpp
        field_item.cpp
        field_robot.cpp
        field_scene.cpp
        field_view.cpp
    ui_setting
        setting_window.cpp
        xbeltree.cpp
    cmd_ui.cpp
    strategy_control_window.cpp
```

Figure **45.** the File Structure of User Interface Module

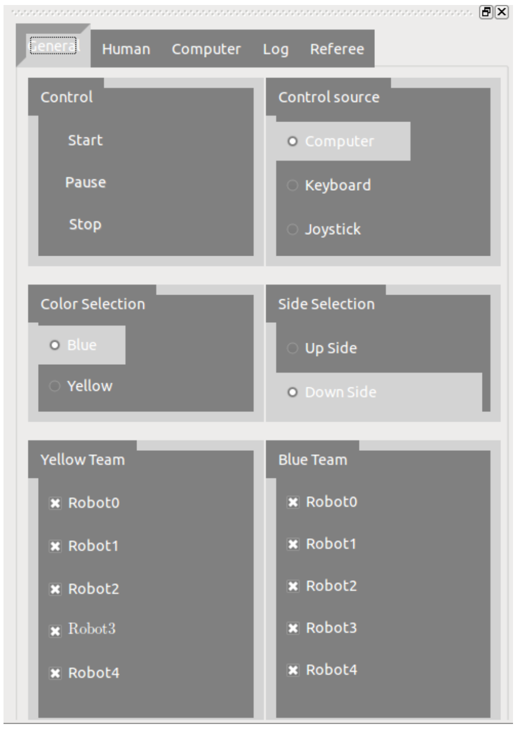
Among all folders are contained in the user interface module. The field_related folder contains specification for all items on the field. The ui_setting folder contains the setting window of the strategy software. In the latter sections, each folder will be introduced in detail.

4.3.3 Appearance Design

The appearance has been decided for functions of control the system. As previously introduced, the whole window has been separated into two main sections. The control panel has five sub tabs and simulation panel has its own component. The function of each part will be introduced in the following content.

- General

This section is designed for general control of the system. By including four group boxes, the most convenient actions are on this page. The first control box includes start, pause and stop. With these functions, the strategy software can decide the state of the system. The second box specifies the control source of the system, and it can be computer (Artificial intelligence), Keyboard and Joystick. In the third group box, the user can specify which side he or she wants to control. If blue is selected, the strategy software is going to control the blue team and vice versa. In the fourth box, the side of the configuration is specified. If upside is selected, robots in the field will try to attack the upper side goal and vice versa. The last two group boxes are used for determining which robots are needed for competition. The strategy software will try to control all checked robots in these group boxes with colour specified above.



General | Human | Computer | Log | Referee

Control: Start, Pause, Stop

Control source: [✓ Computer] Keyboard Joystick

Color Selection: [✓ Blue] Yellow

Side Selection: Up Side [✓ Down Side]

Yellow Team: ✓ Robot0, ✓ Robot1, ✓ Robot2, ✓ Robot3, ✓ Robot4

Blue Team: ✓ Robot0, ✓ Robot1, ✓ Robot2, ✓ Robot3, ✓ Robot4

Figure **46.** General Tab of Strategy Software

- Human

This section specifies that how should human control the robot. The implemented functions include keyboard control, joystick control and kinect control. At this moment, it only has the function of specifying which particular robot the user would like to control.

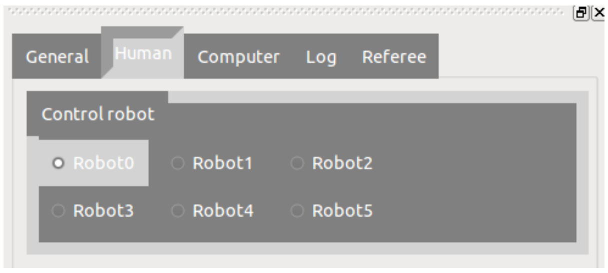


Figure **47.** Human Tab of Strategy Software

- Computer

This section specifies which strategy the software would use for the controlling multi-robot system. The first group box in the demo asks the user to select which mode the user would like the strategy to run. Run strategy mode means the user would like strategy software to run as in a competition and the test strategy mode means the user would like to test one particular skill, tactic or play for debugging purpose. The second group box contains different tests categorized in different sections. The last group box shows the current status of the game field so that the user can get the basic information.

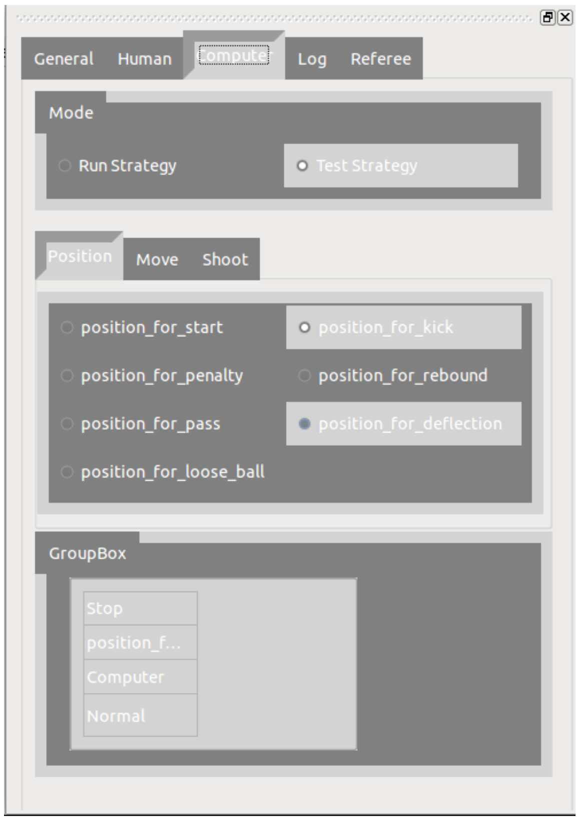


Figure **47.** Computer Tab of Strategy Software

- Log

The log field contains the output of the strategy software. The first group box contains the record of the goals of each team and also the time left for the game. The second group box contains the output of whole software, and normally the information is mainly related to debugging.

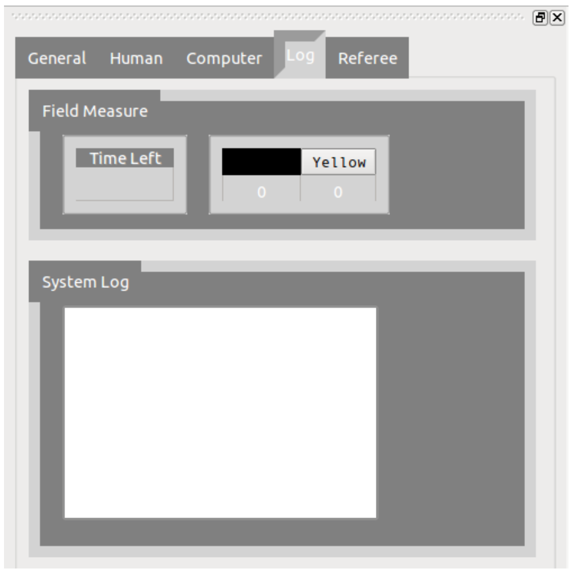
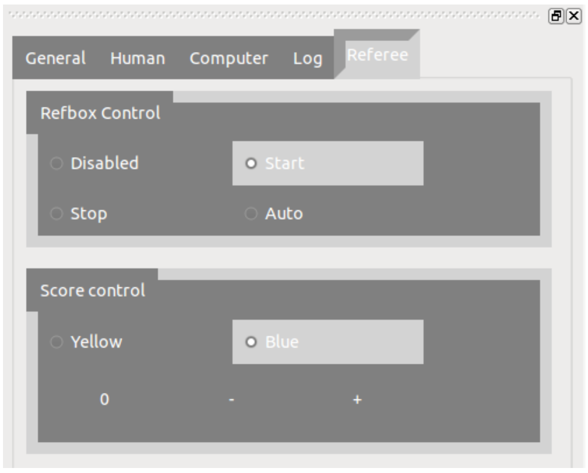


Figure **48.** Log Tab of Strategy Software

The log section specifies the information on the field. No matter it is about the score or about the remaining time, the information will be represented in the table.

- Referee

The section shows the status of the referee box and also sometimes used for simulating the function of reference box. The first group box contains four different selections namely disable, start, stop and auto. If the auto is selected, the strategy software box will act according to the command sent by the reference box otherwise it will act according to what is selected on the panel.



Refbox Control

Disabled [✓ Start]

Stop Auto

Score control

Yellow [✓ Blue]

0 - +

Figure **49.**Referee Tab of Strategy Software

4.3.4 User Interface Style

The original user interface is designed in Qt designer; and the style is old-fashioned comparing with our purpose. As a result, we used QSS to redesign the appearance of the software. The detailed QSS code is in the appendix.

- Simulation Panel

The simulation panel is a place where we store and visualizes the information on the field. The whole panel was drawn by the drawing tools provided by the Qt tool chain.

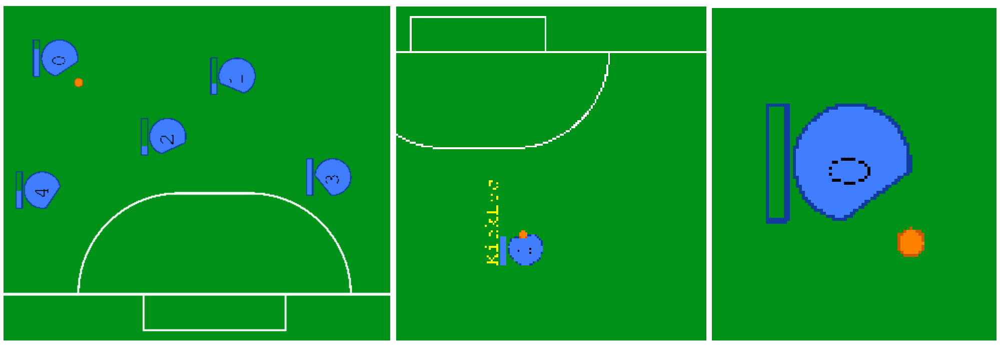
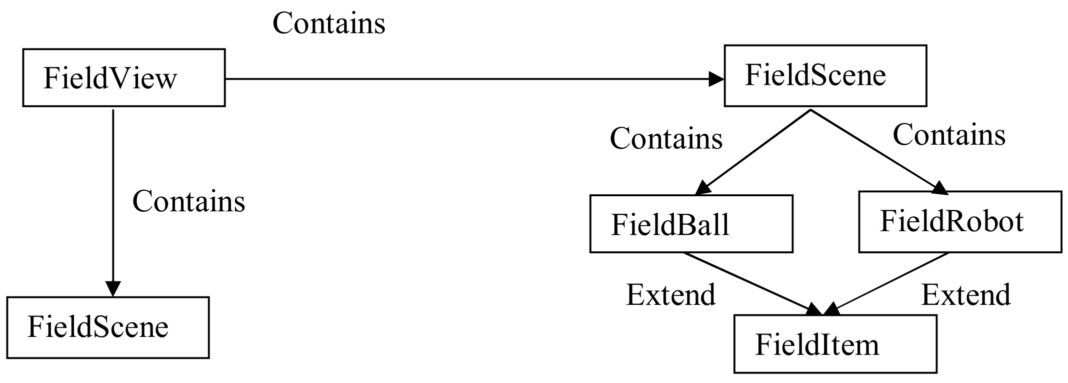


Figure **48.** the Simulated Robots on Simulation Panel

In the folder called field_related, there are several files which are used to represent the information of the field. Among all these files field_global_function.cpp defines the whole configuration of the simulation panel. The two basicfilesarefield_robot.cpp and ball.cpp. These files contain a robot class, a ball class and their different properties including the shapes, confidence bar etc. These two classes both extend a class called FieldItem defined in field_item.cpp. Besides what I mentioned, there also exist two files named as field_scene.cpp, field_view.cpp. Each file defines a class, which has a similar name to the file's name, to represent a possible instance. In fact, the FieldView class defined in field_view.cpp contains FieldScene class defined in field_scene.cpp. To illustrate the idea, the visualized relationships of all classes are represented in following graph:



4.3.5 Logic Structure

The logic of the structure was implemented in a file called strategy_control_window.cpp. In the constructor of class, the class sets the parameters of the whole project. The following code shows the content in the constructor:

Code example:

```
TimerInitialization(); // initialize the timer
SetupWindows(); // allocate the memories and establishUI .
SetupMode(); // set either thread mode or sequential mode of software
SetupThread(); // setup and initialize each thread
SetupWindowsComponent(); // set default status of windows component
SetupGraphicsView(); // setup the status of graphic simulator
SetupGUIConnection(); // setup the signal-slot connection
SetupAutoSelection(); // setup the debug connection
SetupWindowProperties(); // setup the properties of widow e.g. title
StartTimer(); // start the timer
```

Figure **49.** the Code Example of Setup Step

In the previous example code, the contractor establishes the logic structures step by step. The name of each step is very demonstrative.

4.3.6 Control and Animation

The connection between the animation and the control panel is QT's signal/slot mechanism. Different actions change the state of the field. For example, the radio buttons of selecting the robot influence the number of the robots in the field.

Each action you make on the control panel will firstly be captured by the program and then stored in the world class. Then the simulation panel, which is in the main thread, will continuously acquire the world class for information. As soon as it receives the information, it puts the animated effect onto the objects in the field including the direction, confidence of belief, the number of robots etc.

4.4 Wireless Module

4.4.1 Introduction

The wireless module was implemented for communication with multi-robot system. For communicational efficiency, different protocols were all included in this module together. There are two reasons for doing this, the first reason is all the wireless protocols need to use the RS232 port for sending signals to the wireless hardware, the second reason is that if we offer a standardized API to the main class, and then it becomes easier to manage the resources. Nevertheless, the resources need to be controlled by different threads, and only by designing a simple mechanism, the deadlock avoidance can be achieved.

4.4.2 File Structure

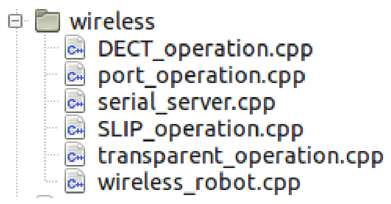

```
wireless
    DECT_operation.cpp
    port_operation.cpp
    serial_server.cpp
    SLIP_operation.cpp
    transparent_operation.cpp
    wireless_robot.cpp
```

Figure **50.** File Structure of Wireless Module

From the picture, we can see the wireless module is quite clear as it does not contain any sub folders.

4.4.3 Logic Structure

The main file which provides most of the APIs is serial_server.cpp. It contains a class called SerialServer, and this class includes all header files of other protocols. Among all files, the port_operation.cpp manages the distribution of serial port, the SLIP_operation.cpp defines the operation related SLIP protocol, the DECT_operation.cpp defines the protocol related to DECT, and transparent_operation.cpp determines the use of transparent protocol.

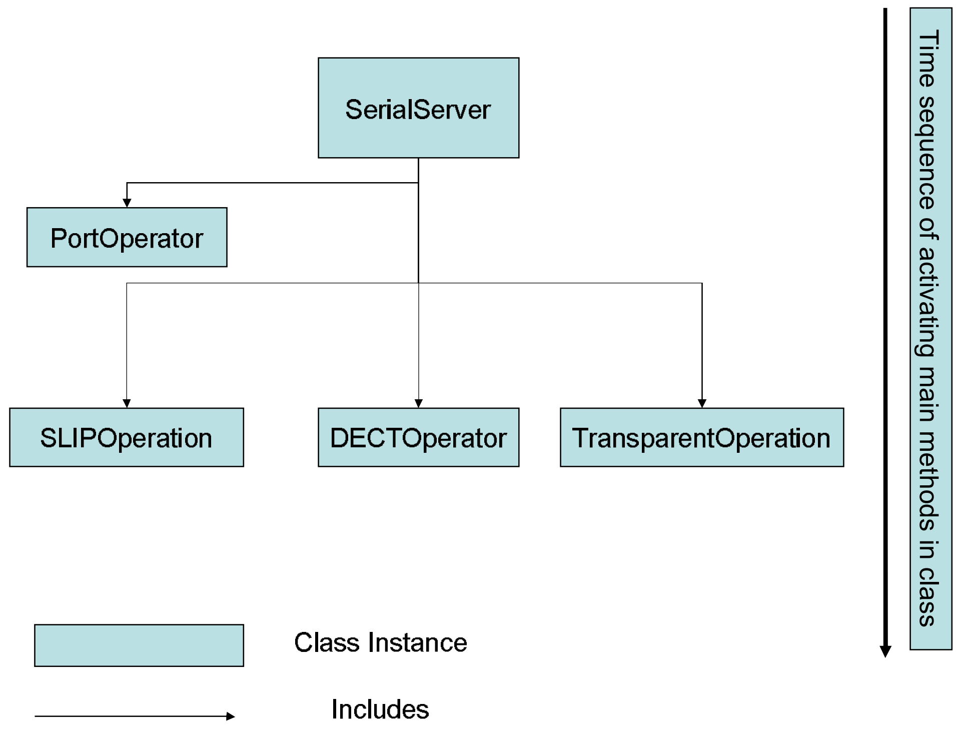


Figure **51.** Structure of Wireless Module

As we can see from the graph, although every individual protocol class has the ability to control the serial port, only the SerialServer can really activate the hardware. In the constructor of SerialServer, the method firstly opens the port and chooses different protocol accordingly. When the main program wants to switch from one protocol to another one, it destroys the instance of the previous class and then make a new instance of the second class.

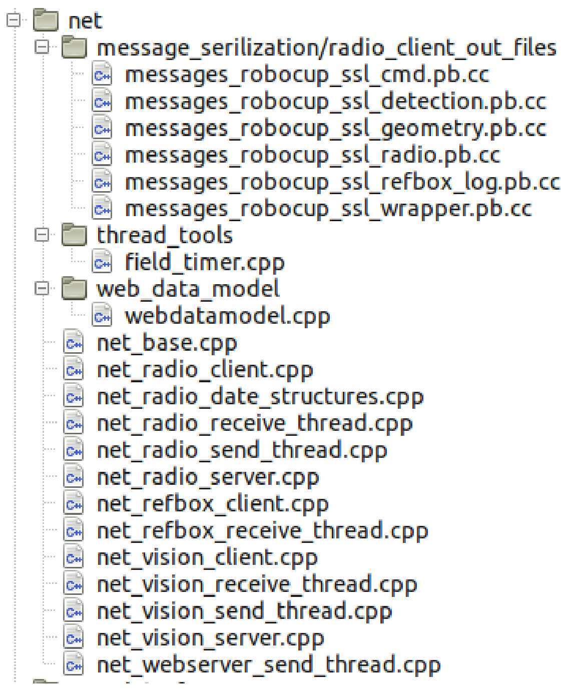
4.5 Internet Module

4.5.1 Introduction

As introduced in the section of the information flow structure, the internet module takes care of the communication channel. The main goals of this module are to establish the stable and fast connection with simulation software, vision software, reference box and the monitoring server. In this case, we have to consider both receiver program and sender program for different connection scheme.

4.5.2 File Structure

The file structure of internet module is as follows:



```
net
  message_serilization/radio_client_out_files
    messages_robocup_ssl_cmd.pb.cc
    messages_robocup_ssl_detection.pb.cc
    messages_robocup_ssl_geometry.pb.cc
    messages_robocup_ssl_radio.pb.cc
    messages_robocup_ssl_refbox_log.pb.cc
    messages_robocup_ssl_wrapper.pb.cc
  thread_tools
    field_timer.cpp
  web_data_model
    webdatamodel.cpp
  net_base.cpp
  net_radio_client.cpp
  net_radio_date_structures.cpp
  net_radio_receive_thread.cpp
  net_radio_send_thread.cpp
  net_radio_server.cpp
  net_refbox_client.cpp
  net_refbox_receive_thread.cpp
  net_vision_client.cpp
  net_vision_receive_thread.cpp
  net_vision_send_thread.cpp
  net_vision_server.cpp
  net_webserver_send_thread.cpp
```

Figure **52.** File Structure of Internet Module

Among all the files and directories, the message_serialization folder contains all the files which are used to change the information from text format to binary format. Nevertheless, most of files are automatically generated by Google protobuf. In the radio_client_out_files folder, we include one specific version of Google protobuf to build this software and also a bash script which can automatically install specific version of Google protobuf on a target computer.

The thread_tools folder contains data structures of timer which is critical to the simulation process of the field. The file field_timer.cpp in this folder defines several global methods to control the flow of the timer.

The web_data_model folder contains the files contributed to the communication between web monitoring system and strategy software. For example, in this class, methods for easier usage of JSON protocol are included.

The rest of files are mainly related to the communication with simulation software, vision software and reference box. Followed by the standard of establishing a communication under Qt environment, all of them have similar implementation, which is making a subclass of QThread. No matter whether it is a receiver program or a sender program, in the run loop (the main loop of a thread), it tries to acquire or receive information from the operating system.

4.5.3 Information Process Method

The receiver and sender program for each protocol are both using pulling method to either receive from or send to a fixed IP address. This method is less efficient, however the Linux operating system does not provide an efficient callback method to pass the information to our application. As a consequence, we inherited the Qt's thread class to continuously monitor a certain port on computer. In this way, the information can be also found and transferred.

4.5.4 Logic Structure

The logic structure of internet module is illustrative. Each file in the class establishes a thread and exchange with the world class.

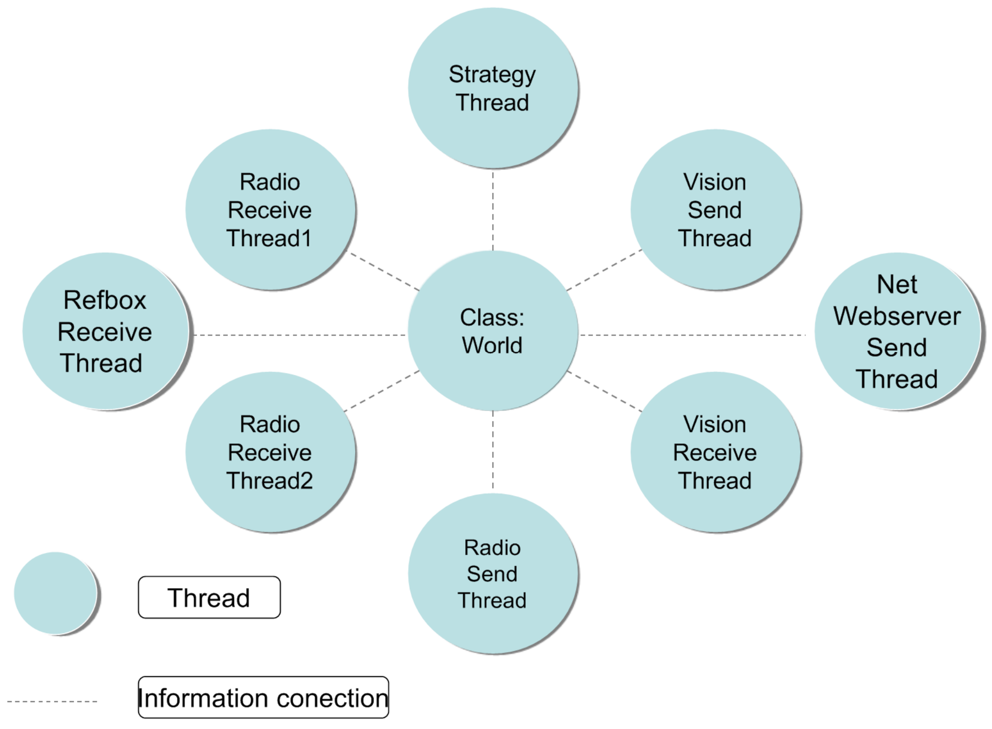


Figure **53.** Logic Structure of Internet Module

4.6 Control Hub Module

4.6.1 Introduction

The control hub module is the most significant module in this strategy software. It monitors the environment of the field, takes all information to make a logical deduction, and makes decision for each action of each robot within a very short time interval. It is also the essential core of making a sequence of actions to enable the robot to move by itself. Besides controlling a robot automatically, this module also specifies several ways for human to control the robot.

Technical Introduction:

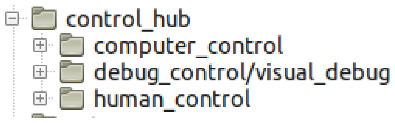
- Freenect

The freenect is C++ library designed for the free control of Kinect. It has several APIs to activate the Kinect and adjust the functionality of Kinect.

- TurboC

TurboC is a C library. With the APIs it provides, people can develop graphical applications quickly. It is developed by Ncurses and Xlibs library under Unix environment. It is compatible with the GUN gcc project.

4.6.2 File Structure

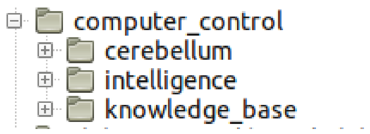


```
control_hub
  computer_control
  debug_control/visual_debug
  human_control
```

Figure **54.** Folder Structure of Control Hub Module

The first level of control_hub module contains three parts. To begin with, the most important part in the folder is computer_control. In this folder, we included all the files which are related to artificial intelligence according to the hierarchical structure of strategy. The second folder is for debugging process. We make unit test and implement new algorithms in this folder. The third folder includes the files for human to control. In a way, we also can say files in this folder implemented the interface of human-computer interaction.

4.6.2.1 Computer Control



```
computer_control
  cerebellum
  intelligence
  knowledge_base
```

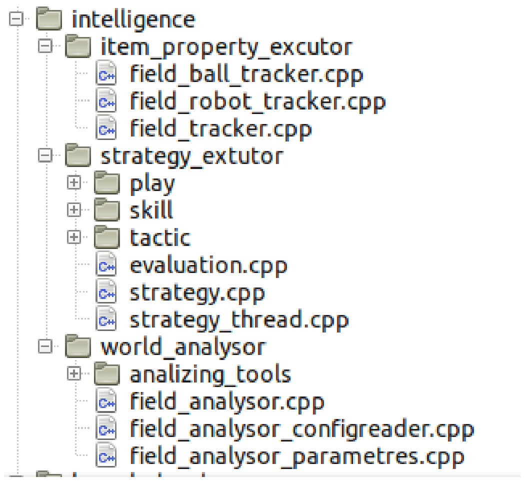
Figure **57.** Structure of Computer Control

The structure of computer control is heavily inspired by the functionality of the brain. There are three parts shown as three folders in the graph. The cerebellum is the Latin name for small brain which enables people to adjust the balance when moving around. The second part intelligence and the third part knowledge_base together represent the cerebrum, which is 'the big brain' in English.

1. Cerebellum

In current version, the cerebellum is not implemented fully. Its function was implemented in the intelligence folder. The future plan, however, is to place the algorithms related to intelligent avoidance to the cerebellum.

2. Intelligence

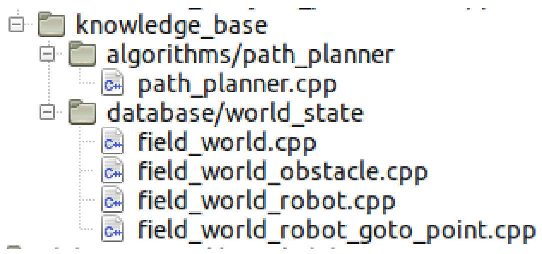


```
intelligence
  item_property_excutor
    field_ball_tracker.cpp
    field_robot_tracker.cpp
    field_tracker.cpp
  strategy_extutor
    play
    skill
    tactic
    evaluation.cpp
    strategy.cpp
    strategy_thread.cpp
  world_analysor
    analizing_tools
    field_analysor.cpp
    field_analysor_configreader.cpp
    field_analysor_parametres.cpp
```

Figure **58.** File Structure of Intelligence

The intelligence folder contains three folders. They are item_property_executor, strategy_executor and world_analyzer individually. The item_property_executor folder is responsible for tracking and analysing the information from the vision and then assigns all this information to an instance called world. The "world" is a class which stores all the information. It will be introduced in detail later. The folder strategy_executor is the most advanced part in this software. It makes the strategy for the multi-robot system. It is also the main part of the strategy thread. The world_analyzer is another import part of interacting with the "world" class. According to our plan, it takes care of the interaction between the strategy software and human. When human changes the property of the field, not only by using the control panel but also by editing the configuration file, this part will automatically assign all the properties to "world" class.

3. Knowledge_base



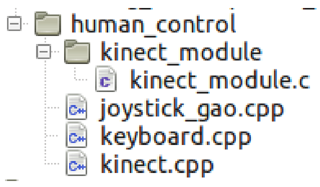
```
knowledge_base
  algorithms/path_planner
    path_planner.cpp
  database/world_state
    field_world.cpp
    field_world_obstacle.cpp
    field_world_robot.cpp
    field_world_robot_goto_point.cpp
```

Figure **59.** File Structure of knowledge_base

The knowledge_base folder is a place where we store all the information of field. The concrete methods are to use a class called world to describe the state of the field. This class includes the setters and getters of all the parameters, the obstacle detection algorithm and the timer settings.

The concept of this folder is an abstraction of memory of human, and in the folder, the strategy software stores the information obtained from the environment.

4.6.2.2 Human Control

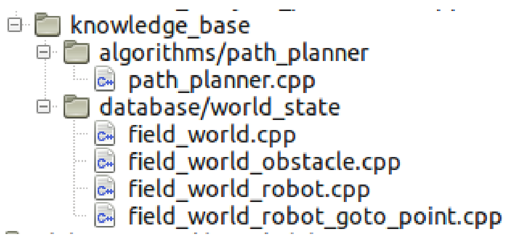


```
human_control
  kinect_module
    kinect_module.c
  joystick_gao.cpp
  keyboard.cpp
  kinect.cpp
```

Figure **55.** File Structure of Human Control

Three kinds of method were implemented in the human control folder. We can use joystick, keyboard or the kinect to control the individual robot.

4.6.2.3 Knowledge Base



```
knowledge_base
    algorithms/path_planner
        path_planner.cpp
    database/world_state
        field_world.cpp
        field_world_obstacle.cpp
        field_world_robot.cpp
        field_world_robot_goto_point.cpp
```

Figure **56.** File Structure of knowledge_base

The knowledge base is a place to store all the information. The concept is inherited from expert system. But this part did not really contain the inference system or pre-programmed knowledge base structure. It can be an implementation in the future.

Besides the world class which includes the information of the field, this knowledge base also contains the basic idea of obstacle. By adopting this concept of obstacle, the world can treat robots, walls or even ball as obstacles under different situation.

4.6.3 Logic Structure

4.6.3.1 Glance

The logic structure is more difficult than it seems. The following graphs show the basic ideas of control hub module.

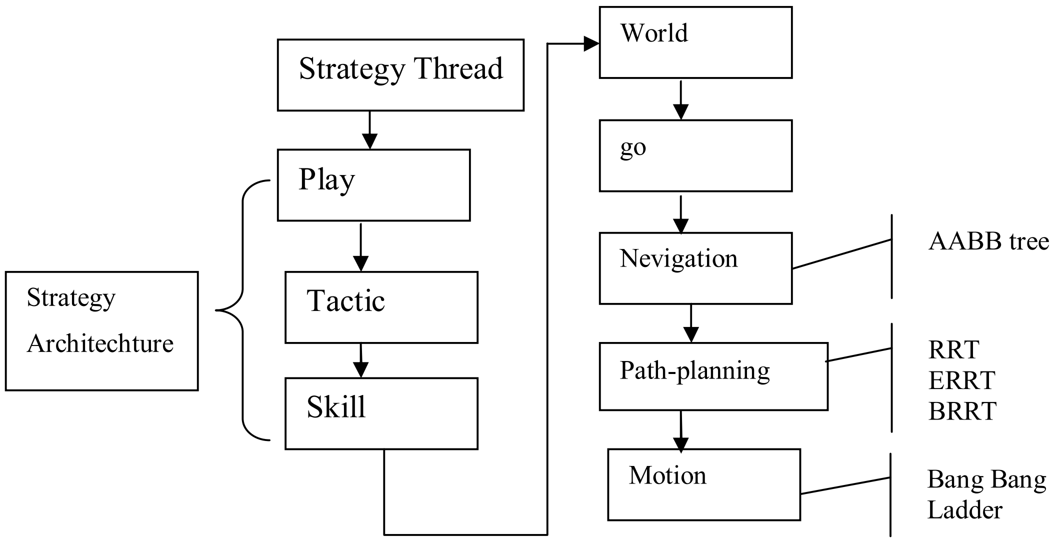


Figure **57.** Logic Structure of Strategy Implementation

The progress is similar to the description in the graph. When the constructor of main window firstly activated the strategy thread, the strategy thread will begin to try to detect which play, tactic and skill belong to current settings. After going through the strategic structure, it will focus on several basic issues.

The first issue is to navigate between several obstacles. The next thing to consider about is how we can find a valid path to go to our goal, and the last thing to consider is to control the motion of the robot so that it can move according to the plan. All these algorithms will be introduced in details later.

Here we will introduce the strategy structure first.

4.6.3.2 First Cluster-Strategy Architecture

The STP (skill, tactic, play) architecture was firstly mentioned by a paper made by Carnegie Mellon University. The author, James Bruce was the researcher in the RoboCup team. According to what he described in the paper:

"The key component of STP is the division between single robot behaviour and team behaviour. In short, team behaviour results from executing a coordinated sequence of single robot behaviours for each team member."

As a consequence, this way of designing strategy would satisfy this complicated, dynamic and adversarial environment.

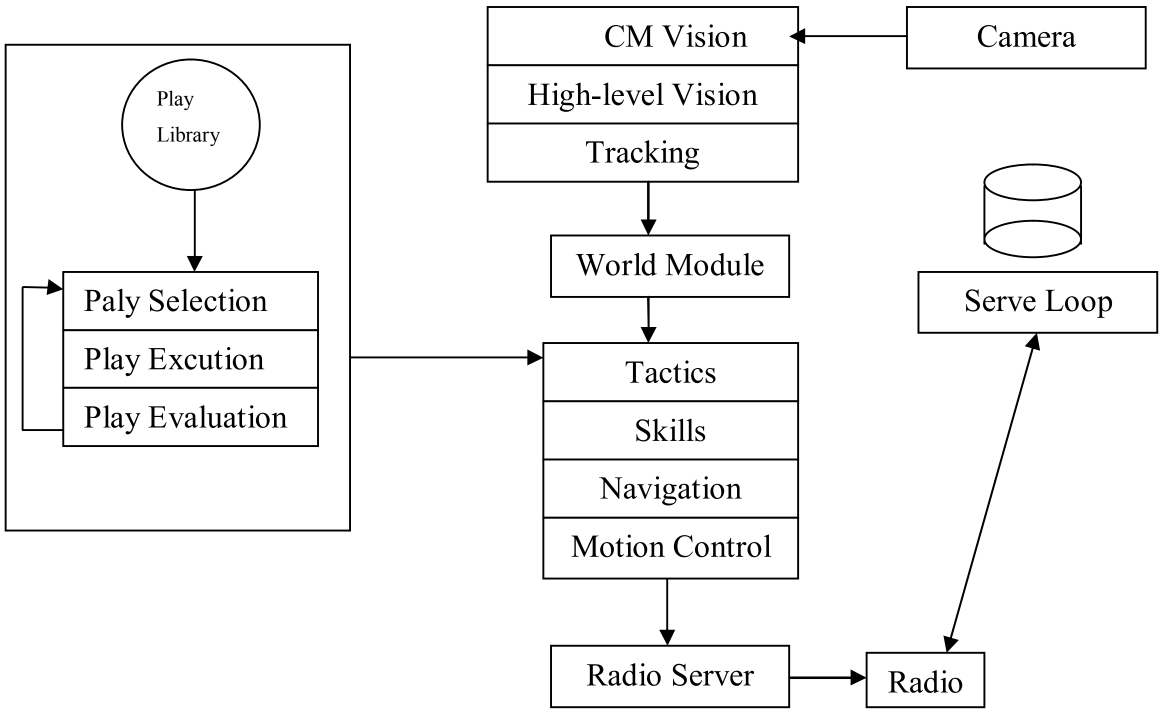


Figure **58.** Structure of First Cluster [17]

4.6.3.3 Second Cluster-From Skill to Motion

The skill defines sequences of actions to make sure that the robot can move according to what has been planned. Each skill class contains the necessary methods including load configuration, navigate, call the planner and control motion.

In each action, the strategy software would input the information received and analysed from the vision system and then get the processed information from different class.

For now, because the functionalities between skill and tactic are not so clear, the skill is implemented as a tactic. The structure of calling from skill to robot is as follows:

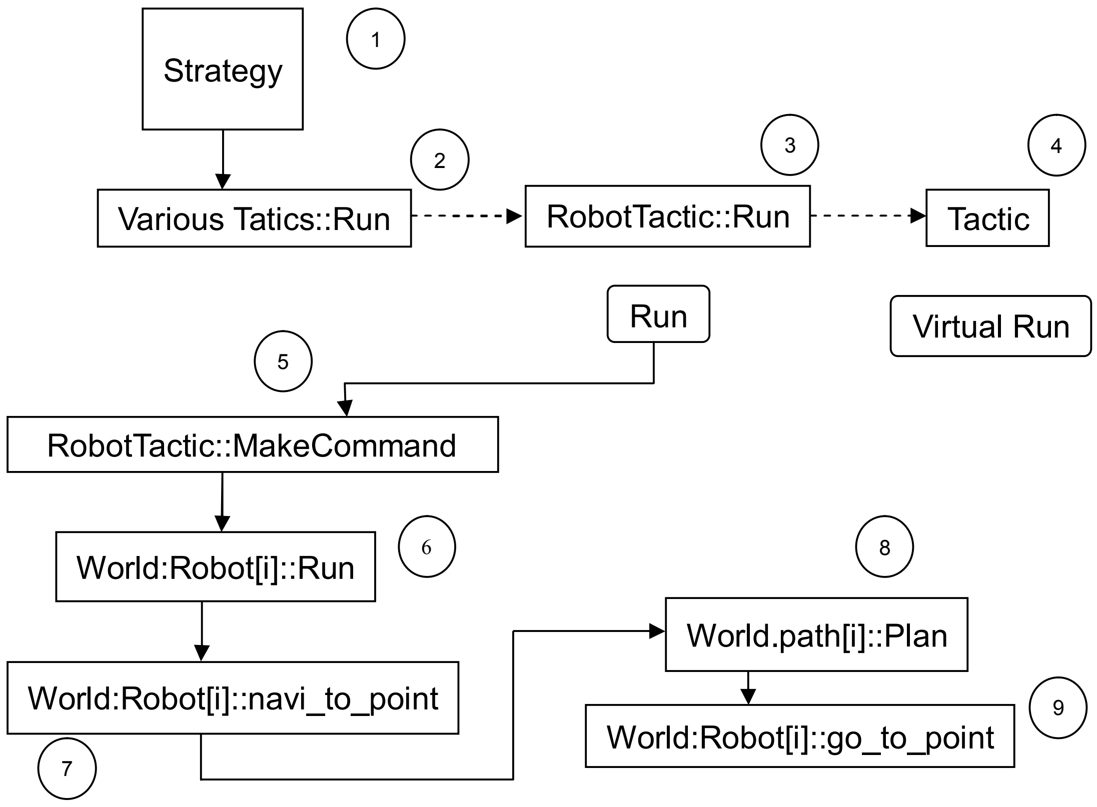


Figure **59.** Structure of First Cluster

The previous graph illustrates how the strategy threads call from the play to skill then to the robot. The first element in the graph represents the strategy thread. It first gets information from the user interface and when the world state has been set to active, it will try to call the run method in specific tactic class. If this tactic has overwritten the run method in the base tactic class, then the strategy thread will call the method directly, otherwise, it will still to call the common method in the RobotTatic class. After entering the process of dealing with the information, the RobotTactic will make a selection based on the whether we should warm up or compete in the field. Then the class which represents the robot in the field will take the responsibility to make intelligent cooperation, aka navigation. The navigation algorithm will give a trajectory as the feedback and based on this trajectory and the information about obstacles, the planner will form a sequence of points to represent the path. Finally, the go_to_point method will send all the information through the internet.

4.6.4 Problem Definition

This section defines the problems of motion planning and cooperative safety as addressed in this thesis. The major notation used for planning and safety are given, along with the definitions of general terms for this whole strategy software development. It also pointed out the complex factors for the algorithms mentioned in the following content.

4.6.4.1 Navigation

Nathaniel Bowditsh firstly defines the navigation as a field of study that focuses on the process of monitoring and controlling the movement of a craft from one place to another. In order to form a specification for the navigation, a more formal definition is adopted.

In fact, for any mobile device, or multi-mobile system, the main purpose is to avoid dangerous situations, for example, the collision, the hot temperature and the deep hole.

5 VISION SYSTEM AND KALMAN FILTER

5.1 Introduction of SSL Vision System

In Chapter 1.1, we have briefly introduced the background of RoboCup SSL competition. It can be seen that RoboCup SSL field is supervised by a camera which is placed on top of the field.

All the in-field soccer robots can be individually distinguished by different colour patterns present on top of each robot.

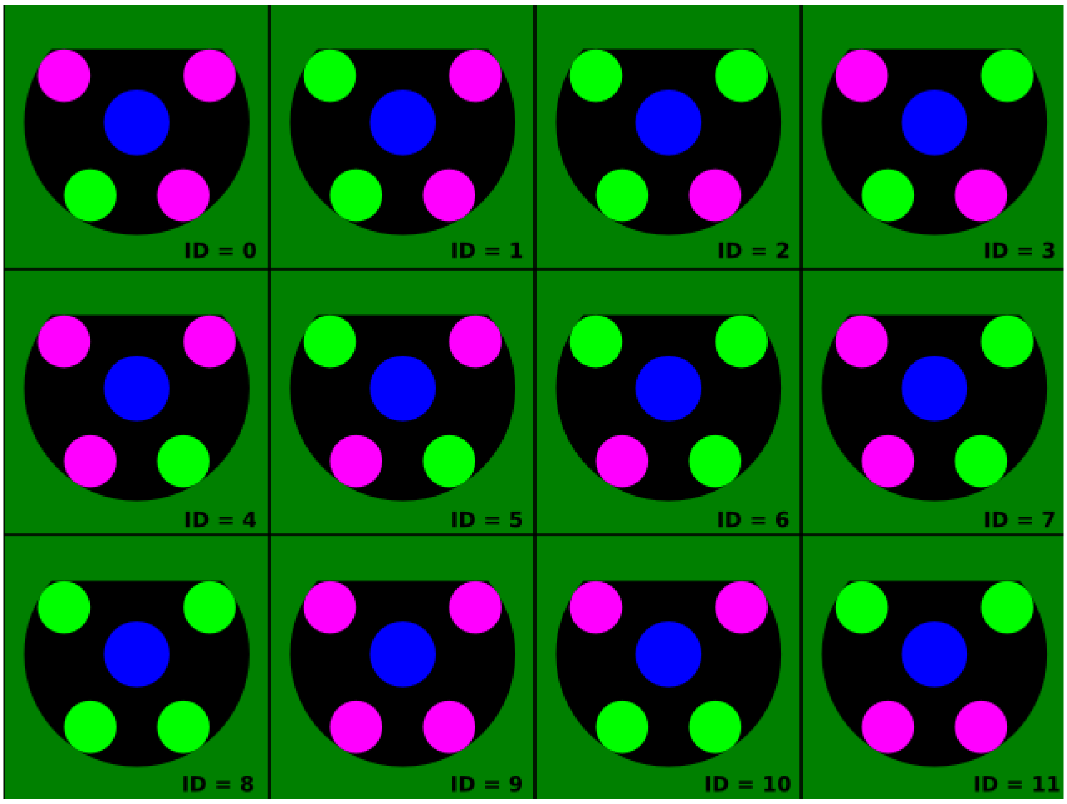
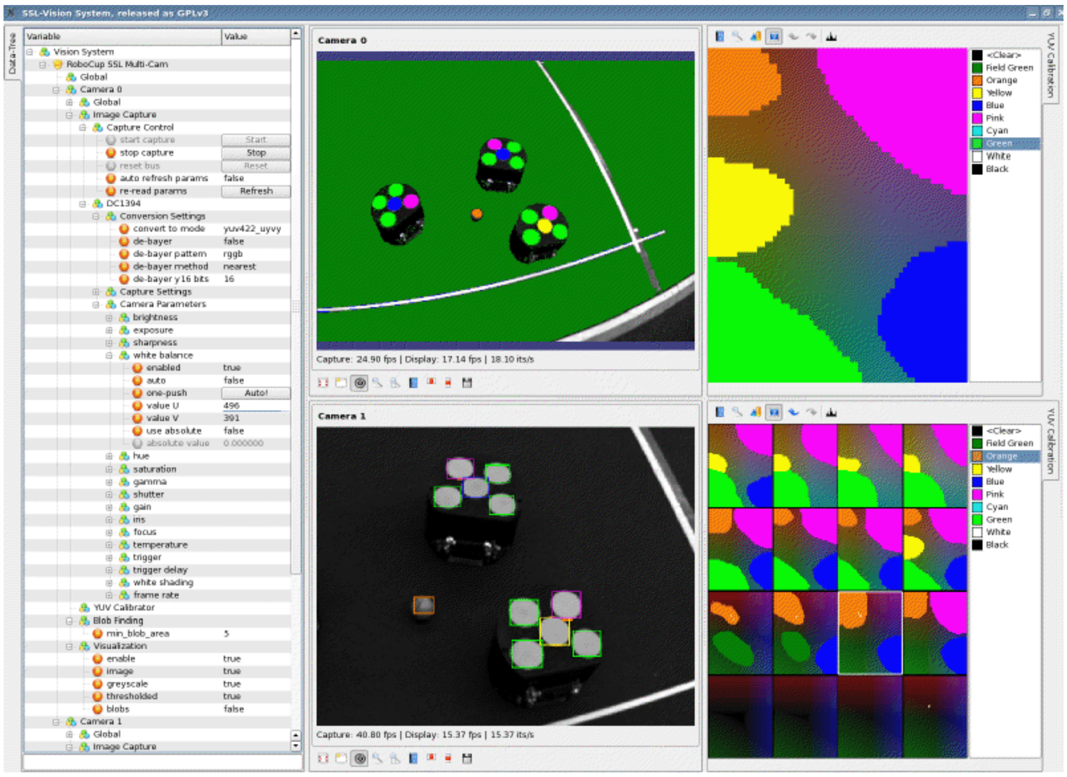


Figure **60.** The Standard Colour Assignments for RoboCup SSL 2012 [18] [19]

The Figure 10 shows the standard colour assignments for RoboCup SSL 2012. Among those 12 patterns, ID 0 to ID 7 is advisable to be used due to stability. Generally speaking, the two adversarial teams are separately named "blue team" and "yellow team". It can be observed from these patterns that the colour of the centre mark of a certain robot can identify itself to its team. The rest of four colour marks which surround the centre mark are used to identify different robot ids and orientation in a team.

The SSL Vision System is a powerful shared vision system developed by volunteers from Carnegie Mellon University. By calibrating the overhead cameras to supervise the whole field and setting up the right team pattern image marker and parameters, SSL Vision System is able to transmit frames of robots and the ball to the vision server on every communication cycle. Then SSL Vision System will decode the received frame information into standardized parameters like position, and transmits the decoded information to our strategy server by using Google Protocol Buffer.



Variable	Value
Vision System	
RoboCup SSL Multi-Cam	
Global	
Camera 0	
Global	
Image Capture	
Capture Control	
start capture	Start
stop capture	Stop
reset bus	Reset
auto refresh params	false
re-read params	Refresh
DC1394	
Conversion Settings	
convert to mode	yuv422_uyvy
de-bayer	false
de-bayer pattern	rggb
de-bayer method	nearest
de-bayer y16 bits	16
Capture Settings	
Camera Parameters	
brightness	
exposure	
sharpness	
white balance	
enabled	true
auto	false
one-push	Auto!
value U	496
value V	391
use absolute	false
absolute value	0.000000
hue	
saturation	
gamma	
shutter	
gain	
iris	
focus	
temperature	
trigger	
trigger delay	
white shading	
frame rate	
YUV Calibrator	
Blob Finding	
min_blob_area	5
Visualization	
enable	true
image	true
greyscale	true
thresholded	true
blobs	false
Camera 1	
Global	
Image Capture	

```
Capture: 24.90 fps | Display: 17.14 fps | 18.10 its/s
Capture: 40.80 fps | Display: 15.37 fps | 15.37 its/s
```

Figure **61.** Screenshot of SSL Vision System [19]

5.2 Introduction of Google Protocol Buffers

Google Protocol Buffer provides a flexible and efficient way to serialize structured data. Compared with an XML file, Google Protocol Buffer is much simpler, smaller, faster, less ambiguous and more user-friendly.

All data flows related to the SSL - Vision System are encoded with Google Protocol Buffer. In Google Protocol Buffer, a .proto file needs to be defined for the data structure and type of message to be transmitted.

There are three .proto files defined in SSL-Vision System.

messages_robocup_ssl_detection.proto: this file includes ball and robot information, which is the detection result of one camera frame.

The complete code is here: [3]

```
message SSL_DetectionBall {
  requiredfloat  confidence =1;
  optionaluint32 area       =2;
  requiredfloat  x          =3;
  requiredfloat  y          =4;
  optionalfloat  z          =5;
  requiredfloat  pixel_x    =6;
  requiredfloat  pixel_y    =7;
}

message SSL_DetectionRobot {
  requiredfloat  confidence  =  1;
  optionaluint32 robot_id    =  2;
  requiredfloat  x           =  3;
  requiredfloat  y           =  4;
  optionalfloat  orientation =  5;
  requiredfloat  pixel_x     =  6;
  requiredfloat  pixel_y     =  7;
  optionalfloat  height      =  8;
}

message SSL_DetectionFrame {
  requireduint32             frame_number  =1;
  requireddouble             t_capture     =2;
  requireddouble             t_sent        =3;
  requireduint32             camera_id     =4;
  repeated SSL_DetectionBall  balls         =5;
  repeated SSL_DetectionRobot robots_yellow =6;
  repeated SSL_DetectionRobot robots_blue   =7;
}
```

From this file, we could get the ball position and robot position in a raw data format, which can be used as the input data for the Kalman Filter of our system.

messages_robocup_ssl_geometry.proto: this file includes the field size of SSL and information related to camera calibration.

messages_robocup_ssl_wrapper.proto: this file includes the previous two .proto files and creates a wrapper for all the detection results that need to be transmitted.

An example client and an example graphical client are included inside the SSL Vision System's directory. These two clients can be used to as a demo client to receive raw vision data from the SSL Vision System.

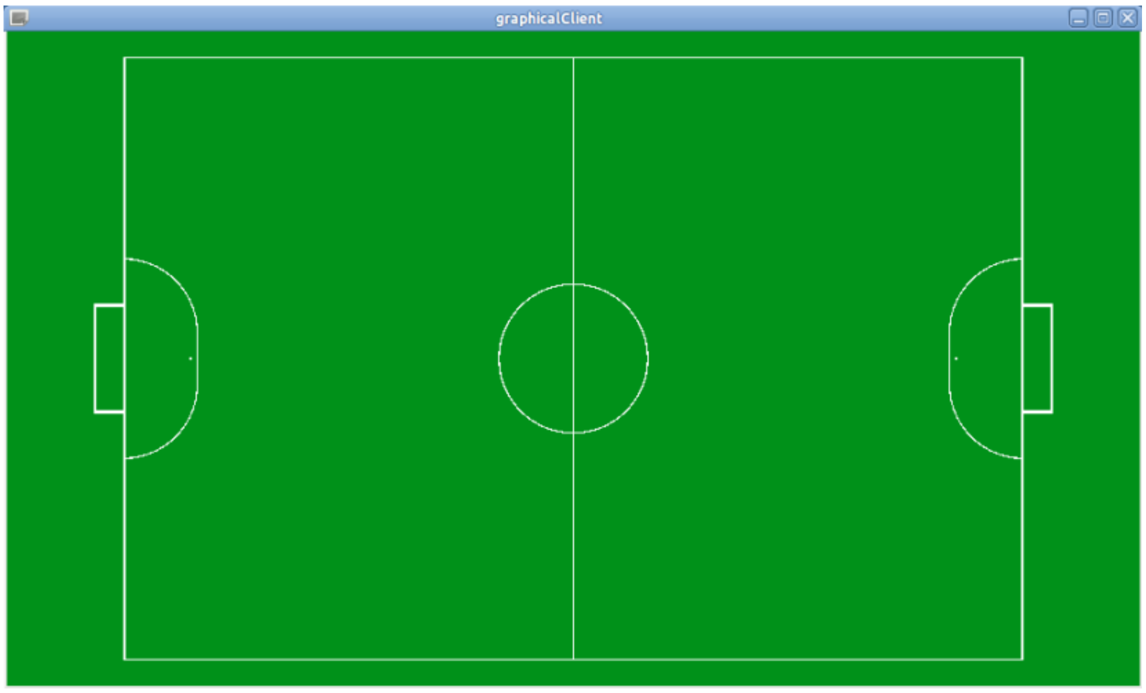


Figure **62.** Graphical Client

5.3 Introduction of Kalman Filter

Although SSL Vision System have already provided a powerful data processing and communicating functions, however, the vision system is not inherently capable of removing interferences and noises which are produced together with vision information, and it is not able to reduce the inevitable transmission delays between vision system and strategy server.

Thus, a vision filter that is able to reduce potential erroneous measurement values and predict the future motion is needed in our strategy software system.

Kalman Filter, which is invented by Rudolf E. Kálmán [20], is an efficient recursive algorithm that is able to optimally estimate future system states based on the noisy raw data input. The Kalman Filter algorithm has been widely used in RoboCup competitions and in other industrial fields like robotics, automation, aeronautics, etc.

5.4 Principle of Kalman Filter

5.4.1 Principle of Kalman Filter

Usually speaking, there are two main steps in Kalman Filter Algorithm: State Predict step and Measurement Update step.

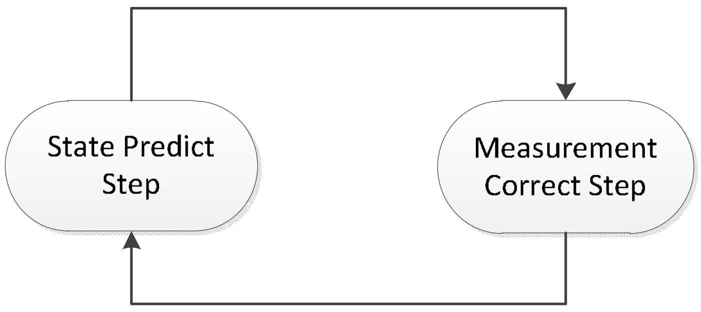


Figure **63.** Kalman Filter Steps

These two steps are executed in a consecutive cycle. That is to say Kalman Filter will first execute State Predict Step and then executes Measurement Correct Step in the first initial round and execute these two steps in the next rounds based on the calculated value in the previous round.

The merit of the Kalman Filter is that it can calculate a predict/estimate value that is close to the real value in reality after many rounds of iteration. The directly measured value from the outside world is believed to be jeopardized by all kinds of interference and delays, so the measured value cannot be taken directly as an accurate value for further calculation. Many rounds of iterative calculation are required for Kalman Filter because in the early stage of iteration the calculated result by Kalman Filter can be far from the real value, however, Kalman Filter will use the measured value which is also not accurate to calibrate its estimated value in the next round. After several rounds of calculation, measurement and calibration, Kalman Filter will yield a reliable result. The detailed explanation of how this works is illustrated below.

5.4.2 State Predict Step

This step involves two mathematical equations:

$$\hat{X}_{k|k-1} = A_k \hat{X}_{k-1|k-1} + B_k u_k \qquad (1)$$

$$P_{k|k-1} = A_k P_{k-1|k-1} A_k^T + Q_{k-1} \qquad (2)$$

In the (1) equation, $\hat{X}_{k|k-1}$ is called the “priori” estimated state variable which is predicted in k round based on k-1 round results. State variable is a set of variables that are used to describe the system state of a dynamic system. For example, in RoboCup SSL, we can set the state variable of the ball as a set of variables like x-velocity, y-velocity, x-coordinate, y-coordinate, rotation angle and etc. Mathematically, these variables can be grouped into a state variable matrix for calculation. For the state variable $\hat{X}_{k|k-1}$, the hat on top of the letter X means that this state variable is not the real value of the state variable X, instead it is an estimated or predicted value calculated by the equation (1). Besides, the subscript of $\hat{X}_{k|k-1}$ identifies this estimate state variable is in its k round whose calculation is based on the k-1 round result.

$\hat{X}_{k-1|k-1}$is the calibrated state variable in the k-1 round. Together with the state transition model A_k, it is used to estimate state variable X in k round. Note that we need to initialize this variable, that is $\hat{X}_{0|0}$, for the first round of process.

$A_k$is called state transition model, and it is used for calculating the new state variable X in its k round based on k-1 round result and that is why it is called the state transition model. Mathematically, A_k can be expressed as a matrix.

u_k is called system input variable (different from measurement input). For example, when applying the Kalman Filter to robots, this variable can be set as a set of values like x-velocity, y-velocity, rotating velocity and etc. based on our command. For some dynamic systems, there is no system input variable u_k. For example, when applying the Kalman Filter to the ball, we do not send any command input to the ball system, thus there is no u_k in the system as well.

B_k is the control-input model which applied to system input variable u_k to calculate the next round system variable X.

By applying equation (1), we can get an estimate of system variable X in k round purely based on the information in k-1 round and command input information. However, the estimate X may not be the same as the real value in reality. It may have some variance compared with the real value. Therefore Kalman Filter needs to calculate "priori" error covariance in equation (2).

In equation (2), $P_{k|k-1}$ is called the "priori" error covariance matrix in round k, and it reflects how much two random variables change together. In this case, $P_{k|k-1}$ can be expressed as:

$$P_{k|k-1} = cov\left(X_k - \widehat{X}_{k|k-1}\right) = E\left[\left(X_k - \widehat{X}_{k|k-1}\right)\left(X_k - \widehat{X}_{k|k-1}\right)^T\right] \qquad (2.1)$$

$P_{k-1|k-1}$is the "priori" error covariance matrix in round k-1, it is used to calculate the "priori" error covariance matrix in round k. The update from $P_{k-1|k-1}$ to $P_{k|k-1}$ reflects the fact that estimate state variable $\hat{X}_{k|k-1}$carries more uncertainty compared with state variable $\hat{X}_{k-1|k-1}$ because process noise w_k and command information is brought into the system.

A_k^T is the transpose form of matrix A_k. This variable is brought into the equation (2) due to the formula of calculating a covariance matrix.

Q_{k-1} is called the process noise covariance matrix and it is one of filter parameters. Process noise w_{k-1} is a random variable appeared in State Process Step of round k and it is distributed under normal probability distribution with a zero mean and a covariance Q_{k-1}:

$$p(w_{k-1}) \sim N(0,\ Q_{k-1}) \qquad (2.2)$$

And process noise w_k is involved in generating the estimate state variable$\hat{X}_{k|k-1}$:

$$\hat{X}_{k|k-1} = A_k \hat{X}_{k-1|k-1} + B_k u_k + w_{k-1} \qquad (2.3)$$

This equation does not explicitly show in the equations of Kalman Filter, but the equation (1) and (2) are actually derived from this equation.

Up to now, Kalman Filter has gone through its first step – State Predict Step. During this step, Kalman Filter calculates a new estimate state variable in k round and it also calculates the uncertainty of this new generated estimate state variable. The uncertainty can be alleviated during the next step when measurement values are brought into the system.

5.4.3 Measurement Update Step

This step involves three mathematical equations:

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1} \qquad (3)$$

$$\hat{X}_{k|k} = \hat{X}_{k|k-1} + K_k (z_k - H_k \hat{X}_{k|k-1}) \qquad (4)$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \qquad (5)$$

These three equations occur in the step of measurement update, which is also called the "posteriori" correction step. During this step, the dynamic system receives measurement from the outside world, like sensor. Then the Kalman Filter uses the received measurement to correct the "priori" estimated state variable $\hat{X}_{k|k-1}$ and error covariance matrix $P_{k-1|k-1}$. The word "priori" refers to the period of time before receiving any measurement in the given round, while "posteriori" refers to the period of time after receiving measurement. After correction, the estimated state variable will be more close to the truth.

Equation (3) is an equation to calculate the important factor – Kalman Gain K_k in the k round. Kalman Gain is crucial in the whole Kalman Filter because it plays a balancing role between the model prediction and received measurement. When Kalman Gain is high, the Kalman Filter will weights more heavily on the measurements, that is to say Kalman Filter will trust more on the measurement data. When Kalman Gain is low, the Kalman Filter will put more weight on the prediction model we built in equation (1), that is to say Kalman Filter will consider the measurement data to be not reliable enough. In the most extreme situations, when Kalman Gain is zero, the measurement data are completely ignored in the following calculations. And when the Kalman Gain is one, the prediction model is considered to be totally wrong, thus the estimate state variable in the previous step will be completely ignored.

Actually, Kalman Gain is derived from the minimization of "posteriori" estimate covariance matrix $P_{k|k}$:

$$\frac{\partial(P_{k|k})}{\partial(K_k)} = 0 \qquad (6)$$

More details of deriving the Kalman Gain can be found from References [21], [22], and [23]

$P_{k|k-1}$ is the same "priori" error covariance matrix which appears in the equation (2).

H_k is called the observation model, which is a matrix that maps the true space into the measurement space. Obviously, H_k^T is the transpose form of H_k.

R_k is called the measurement noise covariance matrix. Similar to the process noise random variable w_{k-1}, there is a measurement noise random variable v_k, which is assumed to be distributed under normal probability distribution with a zero mean and a covariance R_k:

$$p(v_k) \sim N(0,\ R_k) \qquad (7)$$

Note that process noise covariance matrix Q_{k-1} can be different from the measurement noise covariance matrix R_k. These two matrixes are assumed to be independent of each other.

Together with the observation model H_k that we explained above, the measurement noise v_k is also considered to be one part of the measurement data z_k:

$$z_k = H_k \hat{X}_{k|k-1} + v_k \qquad (8)$$

This equation is also not explicitly show in the equations of Kalman Filter, but the equations (3), (4) and (5) are derived from this equation.

After we have calculated Kalman Gain K_k in the equation (3), we use it as a balancing weight in the equation (4) to calculate the "posterior" corrected result of state variable $\hat{X}_{k|k}$ after Kalman Filter received the measurement data in k round.$z_k$is the measurement data, which in our case is the measurement of the position of the ball or robots in the field.

The factor $(z_k - H_k \hat{X}_{k|k-1})$ in the equation (4) is called the "innovation" or "residue". It reflects the difference between the measurement data z_k and predicted measurement $H_k \hat{X}_{k|k-1}$ based on the observation model H_k and the "posterior" estimated state variable $\hat{X}_{k|k-1}$. When the innovation is zero, it means that the actual measurement data are the same as the predicted measurement data. And the Kalman Gain plays a balancing role between the "posterior" estimated state variable $\hat{X}_{k|k-1}$ and the innovation factor as we explained above.

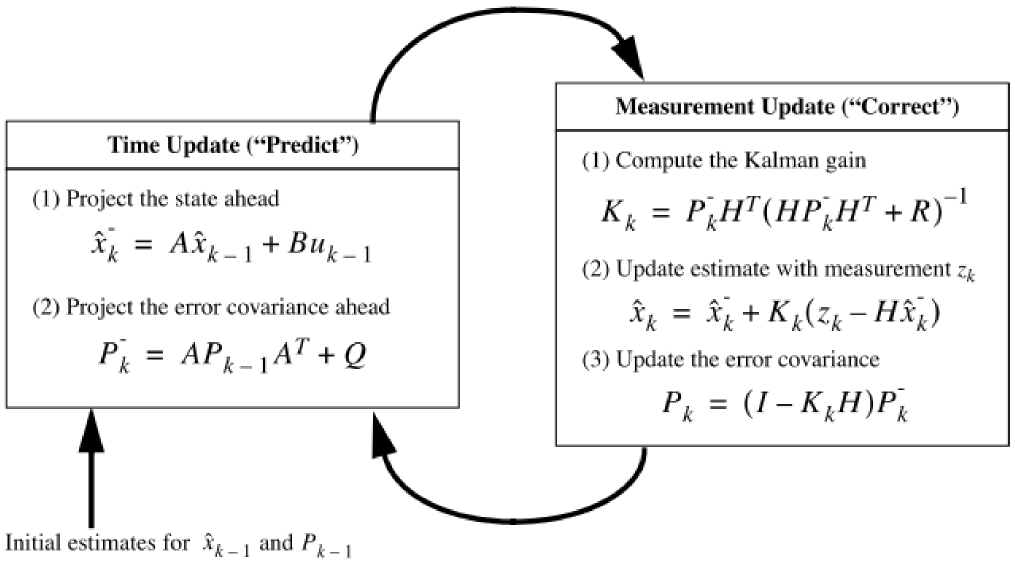
Similar to equation (2), $P_{k|k}$ is called the "posterior" error covariance matrix in round k. In this case, $P_{k|k}$ can be expressed as:

$$P_{k|k} = cov(X_k - \hat{X}_{k|k}) = E\left[(X_k - \hat{X}_{k|k})(X_k - \hat{X}_{k|k})^T\right] \tag{9}$$

More details of deriving the "posterior" error covariance matrix can be found from References [21], [22], and [23].

5.4.4 Summary

The following is a figure which nicely combines the five equations used in Kalman Filter:



Time Update ("Predict")
(1) Project the state ahead
$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1}$
(2) Project the error covariance ahead
$P_k^- = AP_{k-1}A^T + Q$
Measurement Update ("Correct")
(1) Compute the Kalman gain
$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$
(2) Update estimate with measurement z_k
$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$
(3) Update the error covariance
$P_k = (I - K_k H)P_k^-$
Initial estimates for $\hat{x}_{k-1}$ and P_{k-1}

Figure **64.** Two steps in Kalman Filter [24]

From the two steps we discussed above, these two steps are executed recursively a number of times to yield an optimal estimate state variable. This feature is especially beneficial when it is applied to practical implementations, for example when writing iterative functions code when implementing Kalman Filter.

In fact, there are different kinds of Kalman Filter. The one we discussed above is the discrete form of Kalman Filter which is suitable to apply to the situation that the estimate process and measurement process are in a linear relationship.

In a more complex non-linear dynamic system, we may need to apply different forms of Kalman Filters, mainly including Extended Kalman Filter (EKF), Unscented Kalman Filter, Kalman-Bucy Filter, Hybrid Kalman Filter, Extended Kalman-Bucy Filter (EKBF), etc.

In Chapter 3.5, we will have an investigation on the topic of the specific form of Kalman Filter that is suitable when applied to the RoboCup SSL system.

5.5 Introduction of Extended Kalman-Bucy Filter (EKBF)

Extended Kalman-Bucy Filter (EKBF) is the time-continuous, non-linear system variation of the Kalman Filter. Thus it suits perfectly into our RoboCup SSL dynamic system.

5.6 Implementations on Extended Kalman-Bucy Filter (EKBF)

5.6.1 Overview

The paper named Improbability Filtering for Rejecting False Positives [25], illustrates the detailed methodologies to apply Extended Kalman-Bucy Filter into RoboCup SSL and it also introduces a novel way of filtering out false positives by calculating probabilities. The paper also includes some important empirical values which can be used as a good reference in our implementations.

5.6.2 Preparation Works: Development Environment

Considering the fact that the new Botnia RoboCup SSL software system should be running on top of a Linux based machine, we choose the following components as our development environment.

- Ubuntu (10.04 and 10.10 have been both tested) OS
- Qt Framework (version 4.7 or above)
- Qt Creator IDE (version 2.3 or above)
- Subversion (version 1.6 or above)
- G++ Compiler (version 4.4.5 or above)
- CMake build system (version 2.8.2 or above)
- Eigen2 Library
- Google Protocol Buffers (version 2.3 or above)
- OpenGL Library (version 2.1 or above)
- OpenGL Utility Library (version 1.3 or above)
- Libdc1394 Library (version 2.0 or above)
- Libjpeg Library
- Libpng Library
- OpenCV Library (version 2.1 or above)

Ubuntu OS, Qt Framework and Qt Creator IDE can be respectively downloaded from their official website.

By executing the following command in a terminal window, Ubuntu (10.04 or 10.10) should install all the required libraries:

```
sudo apt-get install build-essential gcc g++ subversion libqt4-dev libeigen2-dev
protobuf-compiler libprotobuf-dev libdc1394-22 libdc1394-22-dev cmake libjpeg-
dev libpng12-dev libavformat-dev ffmpeg libcv2.1 libcvaux2.1 libhighgui2.1
python-opencv opencv-doc libcv-dev libcvaux-dev libhighgui-dev
```

After installing the above components successfully, we can start configuring the projects inside Qt Creator.

SSL Vision System [3]has already integrated a graphical sample client “graphicalClient” (Figure 13) after we have checkout a copy of source code from its SVN repository. This graphical sample client is able to use Google Proto Buffer to receive vision data from the vision server and display the ball and robots on the field. Thus it serves to be a good base for our further development.

5.6.3 Preparation Works: EKF OpenCV Support

OpenCV is a powerful open-source library mainly aimed at real time computer vision analysing and processing. OpenCV provides an efficient and user friendly way to process computer visions and images.

During the process of implementations, OpenCV can handle the raw vision data and efficiently apply the Kalman Filter Algorithms which require an intensive manipulation of matrix calculation in C++ language.

Although the OpenCV library has the capability of handling discrete linear system by employing native inline data structures and functions like CvKalman, cvCreateKalman, cvKalmanCorrect, cvKalmanPredict, it is not suitable to apply these functions directly in our continuous non-linear case.

OpenCV Based Extended Kalman Filter Frame [26] is a simple and clear OpenCV based Extended Kalman Filter (EKF) abstract implementation, which is released under BSD License. By integrating this code frame into our project, it facilitates the process of the implementation. We have made some modifications to the original code to make it even suitable for our needs.

5.6.4 Ball EKBF Implementations Step I: Theory & Configurations

The state variable of ball system is

$$\hat{X}_k = (x\ y\ v_x v_y)^T \qquad (10)$$

Angular velocity and ball orientation are not present in the state variable because they are not needed. Input command is not given into the system either because the ball is not driven by itself.

The ball has two distinguished behaviours in the field. One is that the ball moves freely on the carpeted field. The other is that the ball hits the edge of the field and moves upwards along the inclined edge. In the previous case, the ball can be considered to suffer from a constant friction force; in the latter case, the friction is negligible.

In our case, applying the Kalman Filter to the ball on the carpeted field should be regarded to have a higher priority.

The kinematic model is governed by the following formulas:

$$\hat{X}_{k|k-1} = M\hat{X}_{k-1|k-1} + acc_k = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \hat{X}_{k-1|k-1} + \begin{pmatrix} a_x \Delta t^2/2 \\ a_y \Delta t^2/2 \\ a_x \Delta t \\ a_y \Delta t \end{pmatrix} \quad (11)$$

$$a_x = \begin{cases} -a_{fr} cos\,\Psi & |v| > a_{fr}\Delta t \\ -\dfrac{v_x}{\Delta t} & otherwise \end{cases}$$

The parameter Ψ is the angle of travel of the ball, v is the ball speed, and a_{fr} is the constant friction deceleration value which empirically set as 245mm/s^2.

For RoboCup SSL system, the state transition model A_k can be considered the same as M which is a 4 by 4 matrix. Control-input model B_k and system input variable u_k are not involved in the ball's system.The process noise covariance matrix

Q_{k-1}Is set to be a 2 by 2 matrix and its value is $diag((\sigma_v^2, \sigma_v^2)$. The observation model H_k is a 2 by 4 matrix with the value of (I_2, O_2). The measurement noise covariance matrix R_k is a 2 by 2 matrix with value of $diag(\sigma_{xy}^2, \sigma_{xy}^2)$. Two additional matrix variables W and V need also to be set in this case. Matrix W is a 4 by 2 matrix with the value of $(O_2, I_2)^T$, and matrix V is a 2 by 2 matrix with value of I_2.

The empirical value of the standard variation σ_{xy} given by paper [15] is set to be 25mm, and the velocity standard variation σ_v is governed by the following formula:

$$\sigma_v = \max\left(\frac{R}{d}\sigma_R + \left(1 - \frac{R}{d}\right)\sigma_0, \sigma_R\right) \tag{12}$$

And d is the distance between the ball and the nearest robot, R is the maximum robot width. σ_R is set to 100mm/s, and σ_0 is set to 10mm/s.

All these parameters have been tuned based on the practical field tests to fit better to our environment.

Overall, applying the Kalman Filter to the system is a process of finding the optimal performance by tuning the Kalman Filter parameter matrixes. And the next step would be to code these parameters and the algorithm of the following five formulas into our project:

$$\hat{X}_{k|k-1} = A_k \hat{X}_{k-1|k-1} + B_k u_k \tag{13}$$

$$P_{k|k-1} = A_k P_{k-1|k-1} A_k^T + Q_{k-1} \tag{14}$$

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1} \tag{15}$$

$$\hat{X}_{k|k} = \hat{X}_{k|k-1} + K_k (z_k - H_k \hat{X}_{k|k-1}) \tag{16}$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \tag{17}$$

Here is an explanation of **GLSoccerView::init_EKF()** which is used to initiate the EKF environment and set up related parameters. All detailed codes can be found in the appendix.

Initialize settings:

```
Initialize Kalman Filter

SetDimensions(intx_dimension,intu_dimension,intz_dimension,intw_dimension,i
ntv_dimension)
```

Set Process noise covariance matrix and measurement noise matrix:

```
Process noise covariance matrix, this should be very small (meaning very reliable
and accurate)

Measurement noise covariance matrix, this should be big  (meaning not reliable
and accurate)
```

Set “priori” and “posteriori” error covariance matrix:

```
setkalman_P_predicted
setkalman_P_last
setkalman_P_updated
```

Set other involved parameters matrixes, like innovation, Kalman Gain, received measurement matrix:

```
setkalman_S
SetZero(kalman_S)
setkalman_K
SetZero(kalman_K)
setkalman_z
SetZero(kalman_z);
setkalman_z_predicted
SetZero(kalman_z_predicted)
setkalman_x_last
setkalman_x_predicted
setkalman_x_updated
```

Set up A, W, H, V these four matrixes in their respective functions, and execute a check to ensure all settings are correct:

```
Get_A()
Get_W()
Get_H()
Get_V()
```

The four overwritten functions are listed and commented as follows:

```
A:=df/dx
Set Jacobian matrix
OutputMat(kalman_A);
```

```
Set kalman_W
```

```
Set kalman_H
```

```
Set kalman_V
```

Two functions which are used to get predicted results are also implemented and commented:

```
f(x,u,0)

Update "kalman_x_last" to be "kalman_x_updated"
Calculate kalman_x_predicted
```

```
zk:=h(xk-,0)
h(xk,0)=H*xk-
Calculate kalman_z_predicted
```

Up to now, we have set up the initial configurations of the Kalman Filter. Tuning these parameters to the optimal status is a challenging work which requires a combination of empirical theory and practical adjusting. For the further development, the capability of auto-tuning Kalman Filter parameters based on a given dynamic system would be a desired yet more challenging work which may require further knowledge on artificial intelligence and machine learning.

5.6.5 Ball EKBF Implementations Step II: Iterative Callings

The gist of the Kalman Filter is to iteratively apply the five formulas to the system. In each iterative round, state Prediction step and measurement update step have been repeated. However, each repetition would result in a closer estimation by combining the latest estimation and new captured measurement data. In order to balance the filter performance and timing delays, we apply the Kalman Filter with the number of iterations equalling to 10.

The following code fragment explains how to iteratively make a proper estimation by employing the Kalman Filter algorithm:

```
1) Get ball's location
Adjust the looping times based on field testing
Update "kalman_P_last" to be "kalman_P_updated"
2) Get measurement raw data
Apply false positive rejection
Get a copy of old valid data in case of false positive
3) Update
Set "kalman_z_predicted" to be "H*X_predicted"
Set filtered result back to the ball object
```

By comparing the raw unfiltered data with filtered data, we discovered that when the ball is in a still condition, the raw coordinate of the ball tends to be varying all the time, which verifies that raw data contains false data resulting from interferences and delays. However, the filtered data have a much less variability tendency, which reflects the reality of the still ball in a better way.

5.6.6 Ball EKBF Implementations Step III: False Positive Rejection

Another novel discovery illustrated in the paper [25] is the way to eliminate false positives that can appear in the filtering process by employing the method named Improbability Filtering (ImpF).

False positives occur in the situation that the measurement value during the Measurement Update Step is significantly different from the estimated value made during the State Predict Step. If these false positives are not eliminated from Kalman Filter, they will be brought into calculation and cause an obvious erroneous estimation in the next iteration.

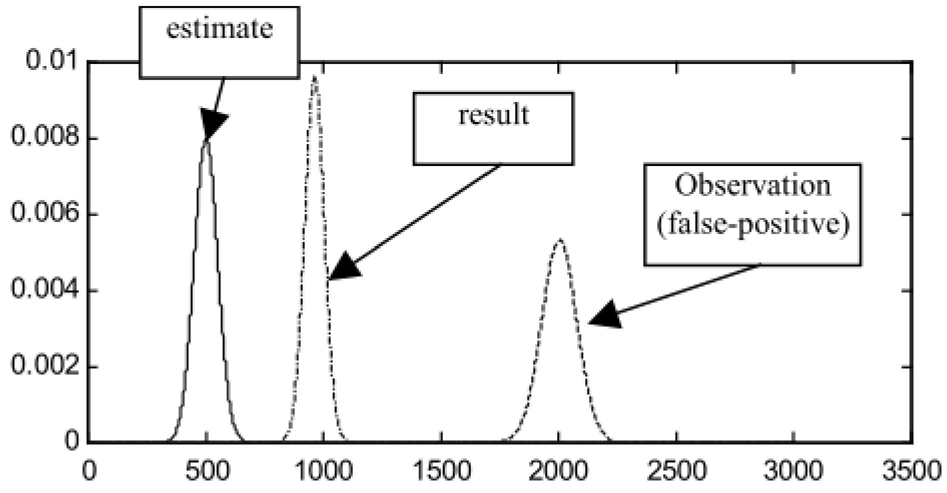


Figure **65.** 1D example of false positive [25]

In Figure 15, we can see clearly the disastrous effect result from the existence of false positives on a 1 dimensional system. For example, in the first round, the estimated value made by Kalman Filter is 500. But in the Measurement Update Step, a measurement value of 2000 is captured, thus cause a new erroneous estimation of 1000 in the next round.

The novel solution to address this problem raised by paper [25] is to calculate the conditional probability density function (pdf) for each of measurement values the system received.

The pdf function is governed by the following formula:

$$P\left[z' \middle| \widehat{X}_k, P_k\right] = \frac{1}{(2\pi|C_k|)^{n/2}} e^{-(z' - H_k \widehat{X}_{ki})^T C_k^{-1} (z' - H_k \widehat{X}_{ki})/2} \qquad (18)$$

With $C_k = H_k P_k H_k^T + R$

C_k: the state covariance matrix transformed to measurement space

n: the number of state variables

z' : measurement value

Note that the accepted range of probability $P\left[z' \middle| \widehat{X}_k, P_k\right]$ should be determined based on on-field testing in order to achieve the optimal performance.

The following code fragment gives the implementation of checking whether false positives exist.

```
boolGLSoccerView::check_false_positive(floatraw1,floatraw2)
{
boolretVal=false;

CvMat*dummy_z=cvCreateMat(2,1,CV_32F);
cvSetReal2D(dummy_z,0,0,raw1);
cvSetReal2D(dummy_z,1,0,raw2);

CvMat*kalman_C=cvCreateMat(2,2,CV_32F);
CvMat*kalman_Ht=cvCreateMat(kalman_H->cols,kalman_H->rows,CV_32F);
CvMat*kalman_H_x_P=cvCreateMat(kalman_H->rows,kalman_P_predicted->cols,CV_32F);
CvMat*kalman_H_x_P_x_Ht=cvCreateMat(kalman_H->rows,kalman_H->rows,CV_32F);

cvTranspose(kalman_H,kalman_Ht);
cvMatMul(kalman_H,kalman_P_predicted,kalman_H_x_P);
cvMatMul(kalman_H_x_P,kalman_Ht,kalman_H_x_P_x_Ht);
cvAdd(kalman_H_x_P_x_Ht,kalman_R,kalman_C);
doubleKalman_C_det=cvDet(kalman_C);

intn=kalman_x_dimension;
doubledenominator=pow((2*3.1415926*Kalman_C_det),(n/2));

CvMat*kalman_H_x_x=cvCreateMat(2,1,CV_32F);
cvMatMul(kalman_H,kalman_x_updated,kalman_H_x_x);
CvMat*kalman_z_minus_H_x_x=cvCreateMat(2,1,CV_32F);
cvSub(dummy_z,kalman_H_x_x,kalman_z_minus_H_x_x);

CvMat*kalman_z_minus_H_x_xt=cvCreateMat(1,2,CV_32F);
cvTranspose(kalman_z_minus_H_x_x,kalman_z_minus_H_x_xt);
CvMat*kalman_Ct=cvCreateMat(2,2,CV_32F);
cvTranspose(kalman_C,kalman_Ct);

CvMat*kalman_z_minus_H_x_xt_Ct=cvCreateMat(1,2,CV_32F);
cvMatMul(kalman_z_minus_H_x_xt,kalman_Ct,kalman_z_minus_H_x_xt_Ct);

CvMat*kalman_z_minus_H_x_xt_Ct_z_minus_H_x_x=cvCreateMat(1,1,CV_32F);
cvMatMul(kalman_z_minus_H_x_xt_Ct,kalman_z_minus_H_x_x,kalman_z_minus_H_x_xt_Ct
_z_minus_H_x_x);

doublenumerator=pow(2.7183,(-
0.5*cvDet(kalman_z_minus_H_x_xt_Ct_z_minus_H_x_x)));

doubleprobability=numerator/denominator;

qDebug()<<Probability:"<<probability;

    //Future adjustment probability range should be based on on-field testing
    //Rather than the "accept-all" policy used in this example
    if(probability>1||probability<1.0e-5)
{
retVal=false; //Bad value
}
else
{
retVal=true; //Good value
}

return retVal;
}
```

And we can call this function in the previous code fragments to ignore all the measurement values belonging to the false positive category.

5.7 Ball EKBF Testing

I made a YouTube video [27] to show the test results. This video is taken in a real RoboCup SSL environment, with raw data transmitted by SSL Vision System. In the video, the orange object represents the ball on the field after filtering and the blue object represents a soccer robot on the field without filtering. It can be seen clearly that the position of the soccer robot varies and the image of it also flashes from time to time. However, the position of ball tends to be quite still and the image of the ball is also much stable.

5.8 Summary

In this chapter, we introduced a powerful algorithm employed in our system. By using EKBF filter, we are able to filter out unwanted white noises and minimize the problems caused by transmission delays. We also implemented the-state-of-the-art Improbability Filtering technique to reduce false positives appeared.

6 SAFTY NAVIGATION

6.1 Introduction on Safety Navigation

One of the most crucial aspects in maintaining an efficient and collision-free dynamic soccer robot system is collision detection and prevention mechanism. This mechanism should be deployed into our strategy software system and it will guarantee that our own teammates will not collide with each other during the competition. Note that this mechanism is mainly responsible for maintaining a collision-free environment within our own teammates rather than preventing the potential collision against opponent robots. The feature of preventing the potential collision against opponent robots requires a separate algorithm explained in Chapter 5.

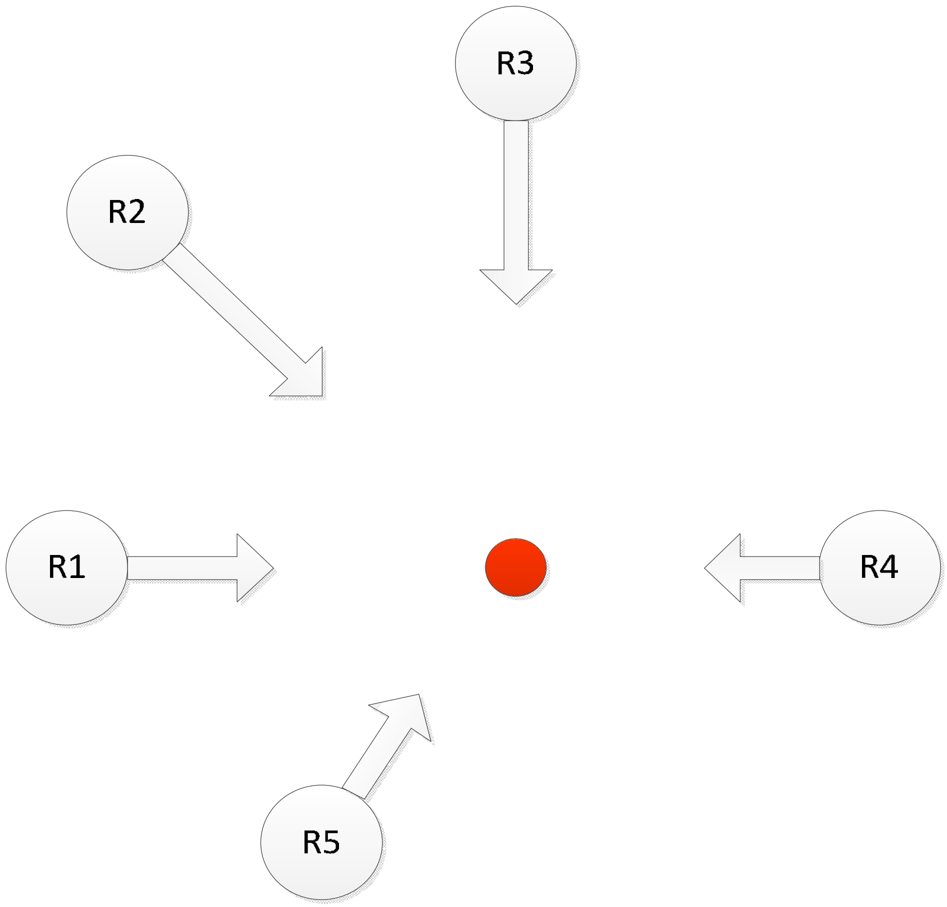


Figure **66.** Demo of Safety Navigation

Figure 16 is a typical case where safety navigation algorithm works. In this imaginary scenario, R1, R2, R3, R4 and R5 are five soccer robots of our team, and the orange object is the ball. If in this case where all of the five our team robots are chasing towards the ball, safety navigation algorithm will coordinate this system by sending proper commands to each of the five robots to prevent disastrous collision.

In paper [28], James Robert Bruce explains a novel safety navigation algorithm and we found it well suitable in our case. This Chapter, we will explore this algorithm and present a simulation-based implementation of this algorithm. While the implementation has mostly followed the core algorithm logic, some methods have been modified to make it more suitable for our needs.

6.2 Introduction of Dynamics Safety Search Algorithm

Dynamics Safety Search (DSS) is a novel algorithm operates for multiple robot agents to provide an exact guarantee of safety.

This algorithm is originated from the Dynamic Window approach (DW) with improvements in many aspects. Compared with Dynamic Window approach, some significant improvements include:

- Provide an exact guarantee of safety.
- Support multiple robot agents at the same time.
- Partial support moving obstacle collision detection and prevention.
- It is an Anytime Algorithm with $O(n^2)$ complexity.

Some other algorithm like Joint Planning algorithm shares some of the listed features. However, the Joint Planning algorithm has an exponential complexity which results in a much larger computation delays, so it does not fit into a system like RoboCup SSL which requires a high timing precision.

Assuming under an ideal environment, meaning there is a perfect communication between command sender and receiver, a perfect dynamics whose behaviour satisfies theoretical computation, this algorithm can guarantee a safe navigation between multiple robot agents.

6.3 Principle and Implementations on Dynamics Safety Search Algorithm

6.3.1 Assumptions and Notations

Some assumptions and constraints need to be applied, and some frequently used notations need to be formulated before getting a further understanding of the algorithm.

- Each robot has a safety radius, denoted as *ROBOT_RADIUS* in the C++ code.
- Each robot has a maximum acceleration value, denoted as *MAXACCEL*.
- Each robot has a maximum deceleration value, denoted as *DECCEL*.
- Each robot has a maximum velocity value, denoted as *MAXVELOCITY*.
- The control period of the system is denoted as *FIXTIME*.

Some fundamental kinematic equations of classical mechanics are also employed in the algorithm, for example:

$$v_f = v_i + at \qquad (1)$$

wherev_f is the final velocity, v_i is the initial velocity, a is the acceleration and t is the time of duration.

$$x_f = x_i + v_i t + \frac{1}{2}at^2 \qquad (2)$$

wherex_f is the final position, x_i is the initial position, v_i is the velocity, t is the time of duration, and a is the acceleration.

6.3.2 Structural Hierarchy of the Algorithm

Following the top-down manner, DSS Algorithm can be divided into three tiers: top, middle and bottom. Here is an overview of the functions in each of the tiers:

- Top Tier consists of three functions: *DynamicsSafetySearch*, *ImproveAccel*, *CheckAccel*.
- Middle Tier consists of two functions: *CheckRobot*, *MakeTrajectory*.
- Bottom Tier consists of one function: *CheckParabolic*.

These six functions have been implemented in the C++ code from the pseudo code in paper [28] by former Botnia team member Xu Zhang and me. Some functions, like *CheckParabolic* are totally rewritten by me to satisfy our needs. In order to test this algorithm more efficiently, we create a Graphical User Interface (GUI) by using the Qt Framework to make it well-visualized.

In order to make it more easy to understand, I use Doxygen Documentation System to generate class diagrams, dependency graphs, collaboration diagrams, inheritance diagrams, call graphs and caller graphs from the source code directly.

Figure 17 is a dependency graph for the main window. It can be seen clearly that the whole algorithm simulation program is formed by the core DSS Algorithm and Qt Framework and Qwt Widgets Library.

In the core DSS Algorithm part, relevant header files include: *RManage.h, Robot.h, Rparabolic.h*.

The Qt drawing and plotting part include: *mainwindow.h, mainpanel.h, QMainwindow, plot.h, QtGui, qwt_plot_curve.h, qwt_plot_grid.h, QTimer, qwt_plot.h, dialog.h*.

Other helper part include: *RComplex.h, RVector.h, defines.h, math.h*.

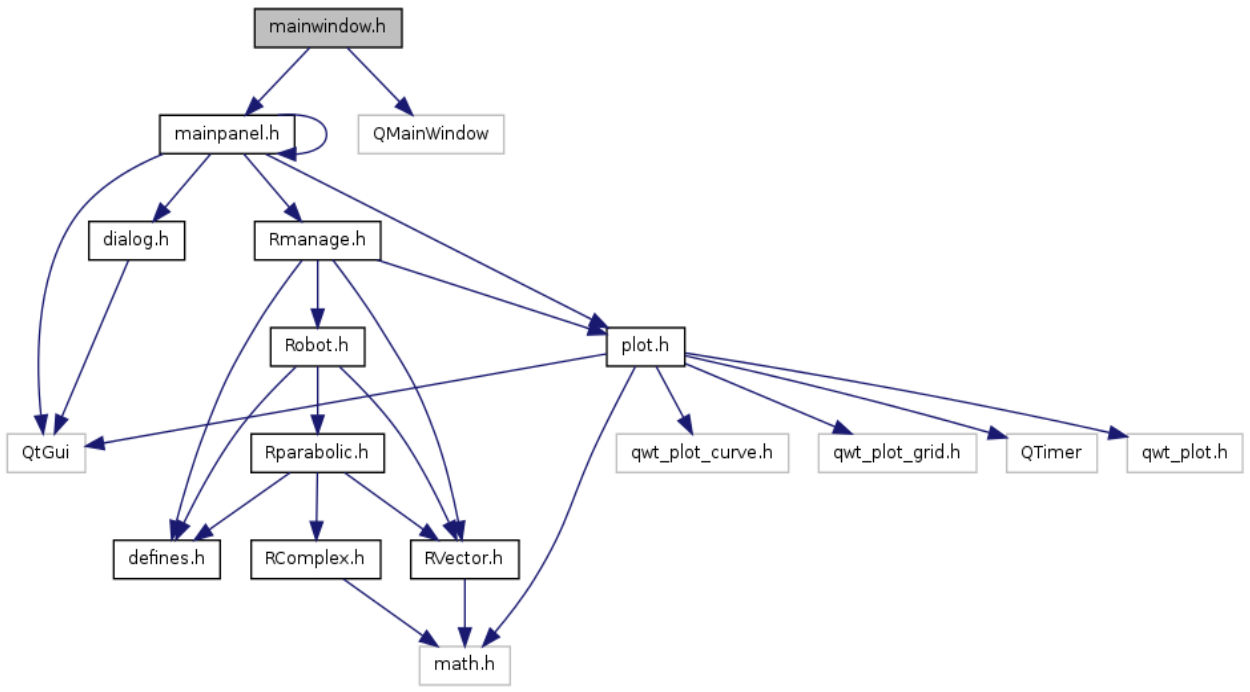


Figure **67.** Dependency Graph

We will explore these three algorithm tiers in 4.3.3 to 4.3.6.

6.3.3 Top Tier

The main procedure in the top tier is *DynamicsSafetySearch*. In the procedure, the algorithm first checks the velocity of each robot in each control cycle. If the checked robot has a velocity, we set the maximum deceleration to the robot. Then we calculate the deviation between the desired acceleration and our set value. We also calculate the deceleration effective time duration in case the robot may move in the opposite direction. Finally, we use *the ImproveAccel procedure* to provide a new acceleration value for each of the robots.

This procedure is important in the whole algorithm because it can assure every robot to be safe in the first place by forcing each robot to decelerate in each control cycle.

The procedure *ImproveAccel* is called from the procedure *DynamicsSafetySearch* and it works by checking the desired acceleration value of robot by using *CheckAccel* function. If the desired acceleration is valid, we will set the desired acceleration value into the robot directly, the deviation to be zero and we finish this procedure. This "short-circuit" mechanism makes the algorithm really fast for most of the time. However, if the desired acceleration is not valid, we will select a random acceleration value from an acceleration set. Then we check the selected acceleration to see if it satisfies the safety condition defined in *CheckAccel* function. We iterate each possible value in the acceleration set to find the one with lowest deviation from our desired acceleration yet can guarantee safety. Finally, we set the optimal selected acceleration value into the robot and we finish *ImproveAccel* procedure.

In the C++ code implementation, these two functions have been packaged into a single *dss* function under *RManage* class, and it is called in every update interval.

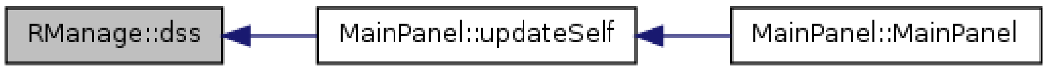


Figure **68.** DSS Function Caller Graph

The function *CheckAccel* is called by *ImproveAccel* procedure. It works by checking *CheckSafetyObs* function to see if the robot may collide with the boundary of the field. It also examines the collision possibility with each teammate robot by calling *CheckRobot* function. If the given acceleration value survives these two tests, this function will verify the acceleration value to be a valid one.

This function is implemented as the *checkAccel* function under *RManage* class:

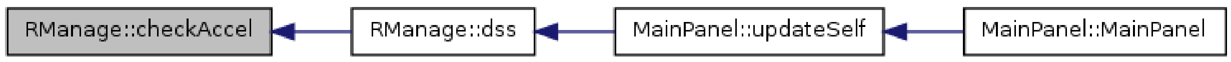


Figure **69.** checkAccel Function Caller Graph

Up to now, we have finished the explanation for the Top Tier function series.

6.3.4 Middle Tier

Function *CheckRobot* and *MakeTrajectory* are defined in the middle tier.

CheckRobot function firstly makes a trajectory for each of the robots to be checked by using *MakeTrajectory* function. The trajectory is made of three segments and we need to check whether these three segments can collide with the other three segments by using *CheckParabolic* function defined in the bottom tier. If no collision happens between any two segments of trajectories, then the algorithm considers these two checked robot to be safe for each other.

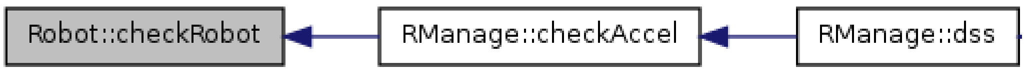


Figure **70.** checkRobot Function Caller Graph

MakeTrajectory function constructs a three-segment trajectory for a given robot. The first segment trajectory is made by the robot's current position, velocity, acceleration and control cycle time duration. Thus, the velocity of the robot in the first segment trajectory is still increasing. The second segment trajectory is made after the maximum deceleration applies to the robot until a full stop. Thus, the velocity of the robot in the second segment trajectory is starting to decrease until a full stop. In the third segment trajectory, since the robot is fully stopped, it has a zero-value acceleration and velocity value.

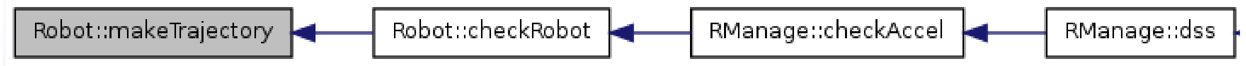


Figure **71.** makeTrajectory Function Caller Graph

The plotting of velocity-time graph can be seen from the simulation result made by our program.

6.3.5 Bottom Tier and Parabola Intersection Checking

In the bottom tier, there is only one function *CheckParabolic*. However, this function plays the most fundamental role in the whole algorithm. This function is responsible for checking whether two non-linear trajectories may collide with each other. In the real implementation, I take a new approach to solve this problem instead of the original one illustrated on paper [28].

The principle of Parabola Intersection Checking mechanism is explained as follows.

We assume the robot A has its radius R_a, initial position P_a, velocity V_a, acceleration A_a and time duration t. And robot B has similar parameters:

$$P_a(t) = P_a + tV_a + t^2 A_a \qquad (3)$$

$$P_b(t) = P_b + tV_b + t^2 A_b \qquad (4)$$

We can calculate the distance between these two trajectories:

$$d(t) = |P_a(t) - P_b(t)| - (R_a + R_b) \qquad (5)$$

In order to find the time t, which is the root of the solution:

$$|P_a(t) - P_b(t)| - (R_a + R_b) = 0 \qquad (6)$$

We use several shorthand notations to make it easier:

$$P_{ab} = P_a - P_b \qquad (7)$$

$$V_{ab} = V_a - V_b \qquad (8)$$

$$A_{ab} = A_a - A_b \qquad (9)$$

$$R_{ab} = R_a - R_b \qquad (10)$$

$$d(t) = |P_{ab} + tV_{ab} + t^2 A_{ab}| - R_{ab} = 0 \qquad (11)$$

Expand this equation by square it:

$$(P_{ab} \cdot P_{ab}) + 2t(P_{ab} \cdot V_{ab}) + 2t^2((P_{ab} \cdot A_{ab}) + (V_{ab} \cdot V_{ab})) + 2t^3(V_{ab} \cdot A_{ab}) + t^4(A_{ab} \cdot A_{ab}) - R_{ab}^2 = 0 \qquad (12)$$

Then we can collapse the coefficients:

$$a = A_{ab} \cdot A_{ab} \qquad (13)$$

$$b = 2(V_{ab} \cdot A_{ab}) \qquad (14)$$

$$c = 2\big((P_{ab} \cdot A_{ab}) + (V_{ab} \cdot V_{ab})\big) \qquad (15)$$

$$d = 2(P_{ab} \cdot V_{ab}) \qquad (16)$$

$$e = (P_{ab} \cdot P_{ab}) - R_{ab}^2 \qquad (17)$$

So we get a simplified quartic equation:

$$t^4a + t^3b + t^2c + td + e = 0 \qquad (18)$$

Ferrari's solution is an elegant way to solve quartic equations, we let:

$$A = \frac{-3b^2}{8a^2} + \frac{c}{a} \qquad (19)$$

$$B = \frac{b^3}{8a^3} - \frac{bc}{2a^2} + \frac{d}{a} \qquad (20)$$

$$C = \frac{3b^4}{256a^4} + \frac{cb^2}{16a^3} - \frac{bd}{4a^2} + \frac{e}{a} \qquad (21)$$

$$P = \frac{-A^2}{12} - C \qquad (22)$$

$$Q = \frac{-A^3}{108} + \frac{AC}{3} - \frac{B^2}{8} \qquad (23)$$

$$R = \frac{-Q}{2} + \sqrt{\frac{Q^2}{4} + \frac{P^3}{27}} \qquad (24)$$

$$U = \sqrt[3]{R} \qquad (25)$$

Then we get the value y, which is decided based on the value of U:

$$y = \begin{cases} -\frac{5}{6}A + U - \frac{P}{3U} & if\ U \neq 0 \\ -\frac{5}{6}A + U - \sqrt[3]{Q} & if\ U = 0 \end{cases} \qquad (26)$$

$$W = \sqrt{A + 2y} \qquad (27)$$

$$X = -\frac{B}{4A} \qquad (28)$$

$$Y = 3A + 2y \qquad (29)$$

$$Z = \frac{2B}{W} \qquad (30)$$

Finally, we get four roots of quadratic equation:

$$t_1 = X + \frac{+W + \sqrt{-(Y + Z)}}{2} \qquad (31)$$

$$t_2 = X + \frac{+W - \sqrt{-(Y + Z)}}{2} \qquad (32)$$

$$t_3 = X + \frac{-W + \sqrt{-(Y - Z)}}{2} \qquad (33)$$

$$t_4 = X + \frac{-W - \sqrt{-(Y - Z)}}{2} \qquad (34)$$

And our solution should be the minimum of the four possible roots:

$$t = \min(t_1,\ t_2,\ t_3,\ t_4) \qquad (35)$$

The implementation in C++ code is listed as follows:

```
StatusRParabolic::checkParabolic(RParabolicp1,RParabolicp2,doubler
)
{
    double tMin=-1;

RVectorXab=RVector(p2.m_postion.x-p1.m_postion.x,p2.m_postion.y-
p1.m_postion.y);
RVectorVab=RVector(p2.m_velocity.x-
p1.m_velocity.x,p2.m_velocity.y-p1.m_velocity.y);
    RVectorAab=RVector(p2.m_acceleration.x-
p1.m_acceleration.x,p2.m_acceleration.y-p1.m_acceleration.y);

double Rab=r;

double XabXab=Xab*Xab;
double XabVab=Xab*Vab;
double XabAab=Xab*Aab;
double VabVab=Vab*Vab;
double VabAab=Vab*Aab;
double AabAab=Aab*Aab;

double A=AabAab;
double B=2*VabAab;
double C=2*(XabAab+VabVab);
double D=2*XabVab;
double E=XabXab-Rab*Rab;

if(fabs(A)<0.1)
if(A==0) A=0.1;
else A=(A>0?1:-1)*0.1;

double a=-(3*B*B)/(8*A*A)+C/A;
double b=(B*B*B)/(8*A*A*A)-(B*C)/(2*A*A)+D/A;
double c=-(3*B*B*B*B)/(256*A*A*A*A)+(C*B*B)/(16*A*A*A)-
(B*D)/(4*A*A)+(E/A);
```

```
double P=-(a*a)/12-c;
double Q=-(a*a*a)/108+(a*c)/3-(b*b)/8;

Complex R=Complex::plus((-Q/2),Complex::Sqrt((Q*Q)/4+(P*P*P)/27));
Complex U=Complex::Pow(R,1.0/3.0);

Complexy(0,0);

if(Complex::Abs(U)<0.00001f)
y=Complex::minus(Complex::plus(-
(5.0/6.0)*a,U),Complex::Pow(Q,1.0/3.0));
else
y=Complex::minus(Complex::plus(-
(5.0/6.0)*a,U),Complex::divide(P,Complex::multiple(3,U)));

Complex W=Complex::Sqrt(Complex::plus(a,Complex::multiple(2,y)));
Double X=-B/(4*A);
Complex Y=Complex::plus(3*a,Complex::multiple(2,y));
Complex Z=Complex::divide(2*b,W);

Complex
t1=Complex::plus(X,Complex::divide(Complex::plus(W,Complex::Sqrt(C
omplex::negative(Complex::plus(Y,Z)))),2));
Complex
t2=Complex::plus(X,Complex::divide(Complex::minus(W,Complex::Sqrt(
Complex::negative(Complex::plus(Y,Z)))),2));
Complex
t3=Complex::plus(X,Complex::divide(Complex::plus(Complex::negative
(W),Complex::Sqrt(Complex::negative(Complex::plus(Y,Z)))),2));
Complex
t4=Complex::plus(X,Complex::divide(Complex::minus(Complex::negativ
e(W),Complex::Sqrt(Complex::negative(Complex::plus(Y,Z)))),2));

std::list<double> tList;
std::list<double>::iterator it=tList.begin();

if(Complex::IsNaN(t1)==false && Complex::IsReal(t1)==true &&
t1.Re>=0)tList.push_back(t1.Re);
if(Complex::IsNaN(t2)==false && Complex::IsReal(t2)==true &&
t2.Re>=0)tList.push_back(t2.Re);
if(Complex::IsNaN(t3)==false &&Complex::IsReal(t3)==true &&
t3.Re>=0)tList.push_back(t3.Re);
```

```
if(Complex::IsNaN(t4)==false && Complex::IsReal(t4)==true &&
t4.Re>=0)tList.push_back(t4.Re);

if(tList.size()>0)
{
it=tList.begin();
tMin=*it;

for(it=tList.begin();it!=tList.end();it++)
if(*it<tMin)tMin=*it;
}

return(tMin>0)?Unsafe:Safe;
}
```

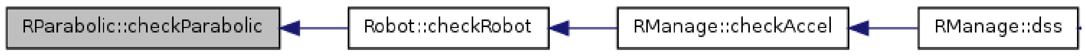


Figure **72.** Caller Graph of checkParabolic Function

6.3.6 Collaboration Diagrams and Call Diagrams

Up to now, we have explained the three tiers in the algorithm. The followings are some important Collaboration diagrams and call diagrams of the implementation work.

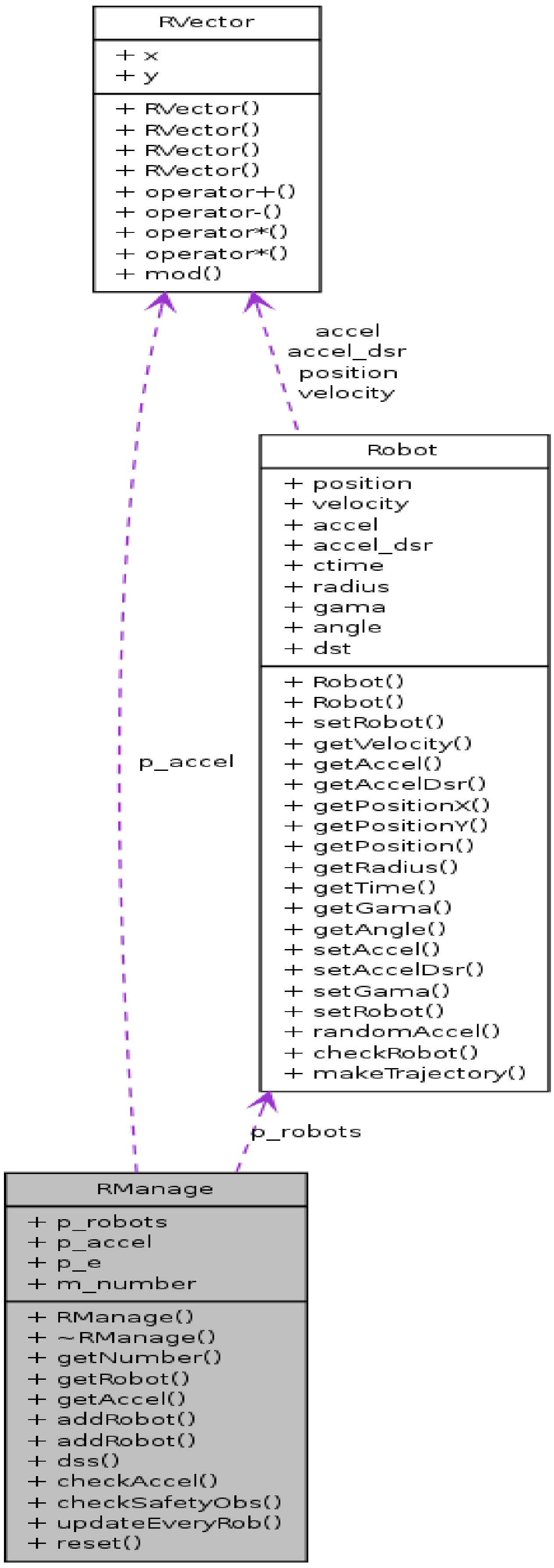

RVector
+ x
+ y
+ RVector()
+ RVector()
+ RVector()
+ RVector()
+ operator+()
+ operator-()
+ operator*()
+ operator*()
+ mod()
accel
accel_dsr
position
velocity
Robot
+ position
+ velocity
+ accel
+ accel_dsr
+ ctime
+ radius
+ gama
+ angle
+ dst
+ Robot()
+ Robot()
+ setRobot()
+ getVelocity()
+ getAccel()
+ getAccelDsr()
+ getPositionX()
+ getPositionY()
+ getPosition()
+ getRadius()
+ getTime()
+ getGama()
+ getAngle()
+ setAccel()
+ setAccelDsr()
+ setGama()
+ setRobot()
+ randomAccel()
+ checkRobot()
+ makeTrajectory()
p_accel
p_robots
RManage
+ p_robots
+ p_accel
+ p_e
+ m_number
+ RManage()
+ ~RManage()
+ getNumber()
+ getRobot()
+ getAccel()
+ addRobot()
+ addRobot()
+ dss()
+ checkAccel()
+ checkSafetyObs()
+ updateEveryRob()
+ reset()

Figure **73.** Collaboration Diagram for RManage Class

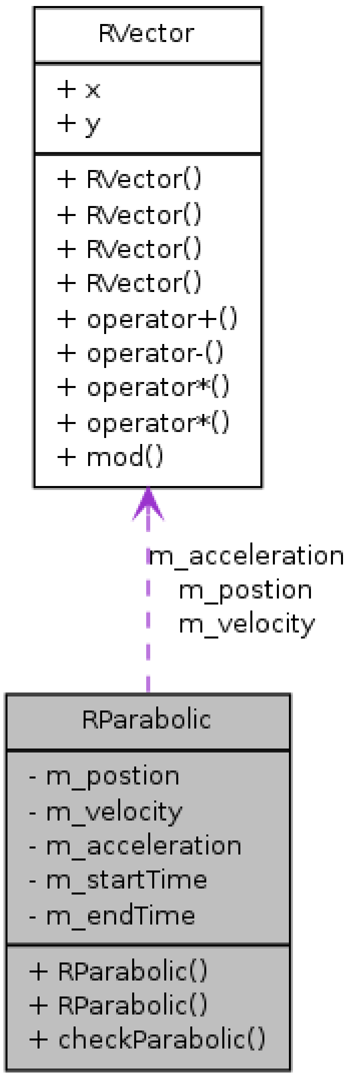


Figure **74.** Collaboration Diagram for RParabolic Class

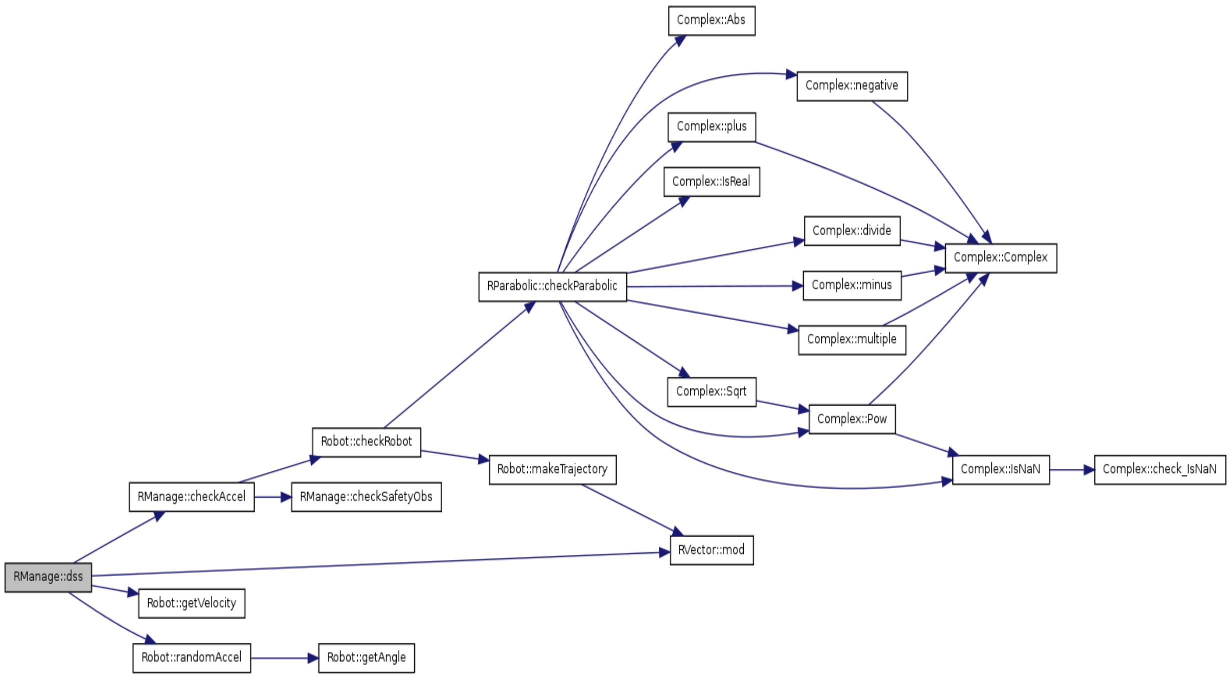


Figure **75.** Call Diagram of DSS Function

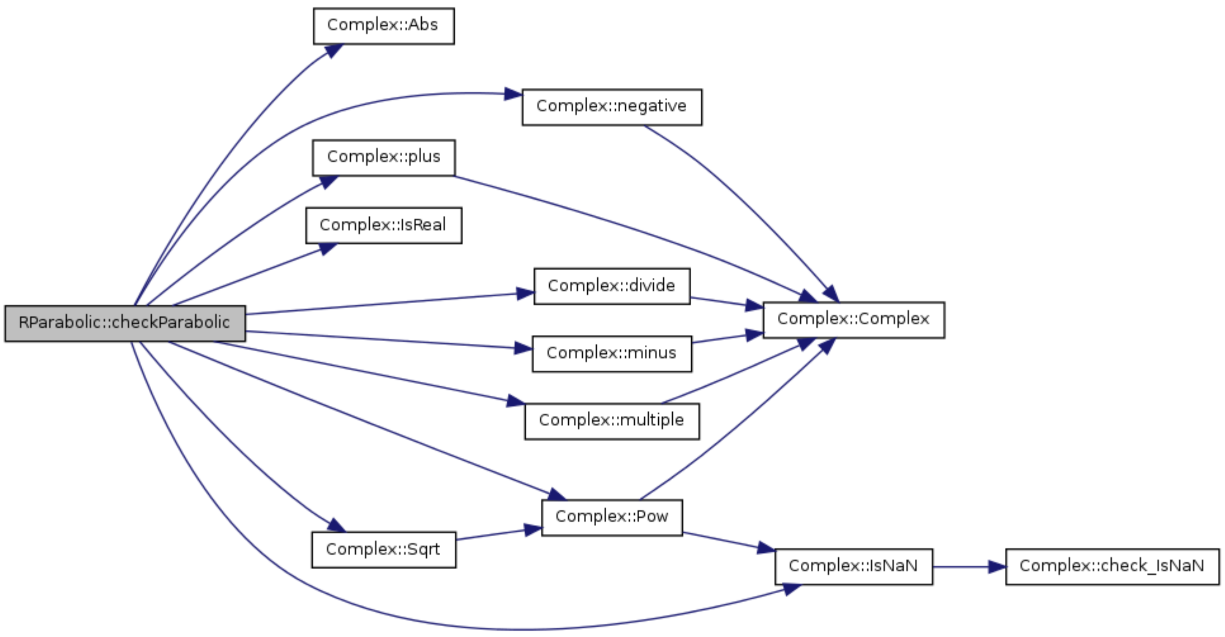


Figure **76.** Call Diagram of checkParabolic Function

6.3.7 Testing

In the simulation program, firstly we simulate a situation where two robots are moving towards each other with a velocity of 1 m/s and without acceleration.

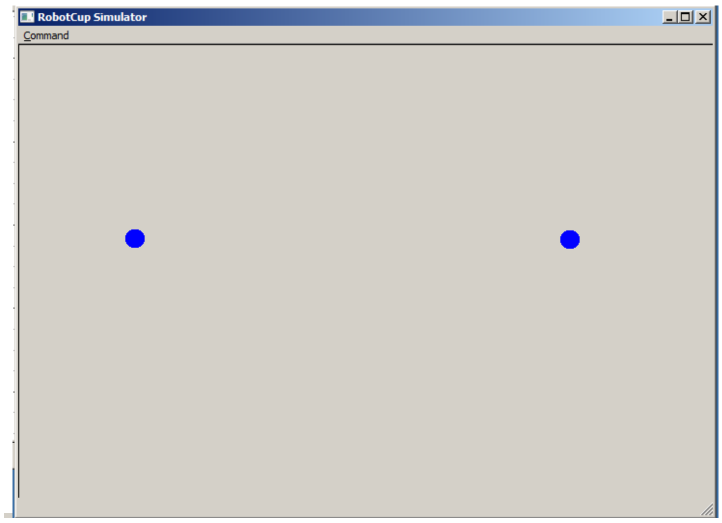


Figure **77.** Two Robots Simulation 1

After we start the simulation:

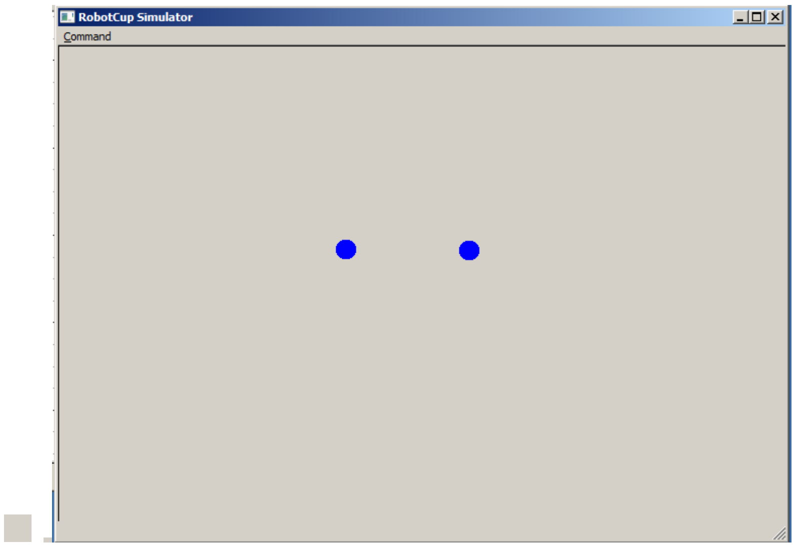


Figure **78.** Two Robots Simulation 2

When the simulation is finished, we can see from Figure 29 that the two robots stop in a safe distance, which is the diameter of the robot in this case.

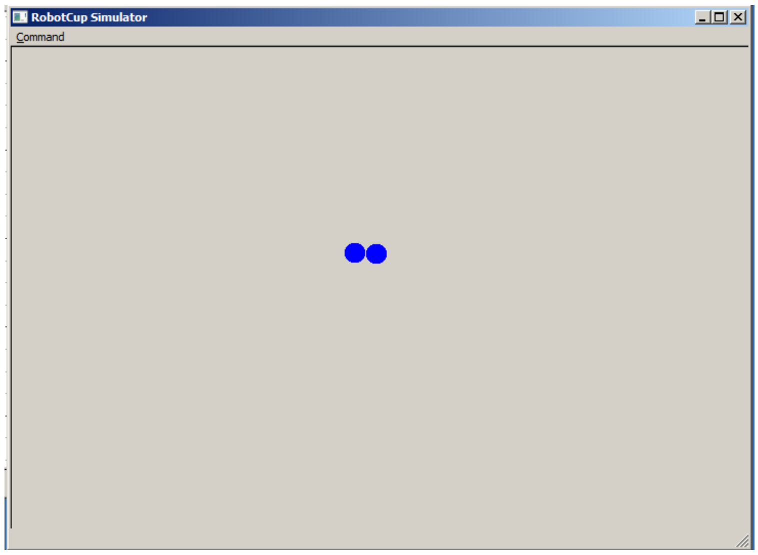


Figure **79.** Two Robots Simulation 3

And we can also plot a real-time velocity-time chart for the first robot (left one). The robot decelerates itself to zero velocity after it is applied the maximum deceleration.

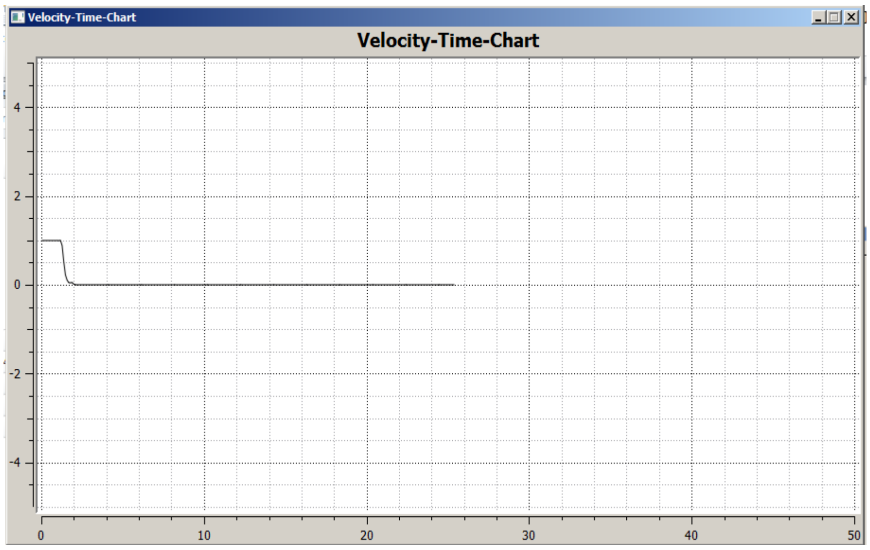


Figure **80.** Velocity-Time Chart

Secondly, we can test a simulation when the robot has an acceleration. In this case, robot1 has a velocity -1 m/s and an acceleration 1 1m/s^2. Robot2 has a velocity -1 m/s.

After starting the simulation:

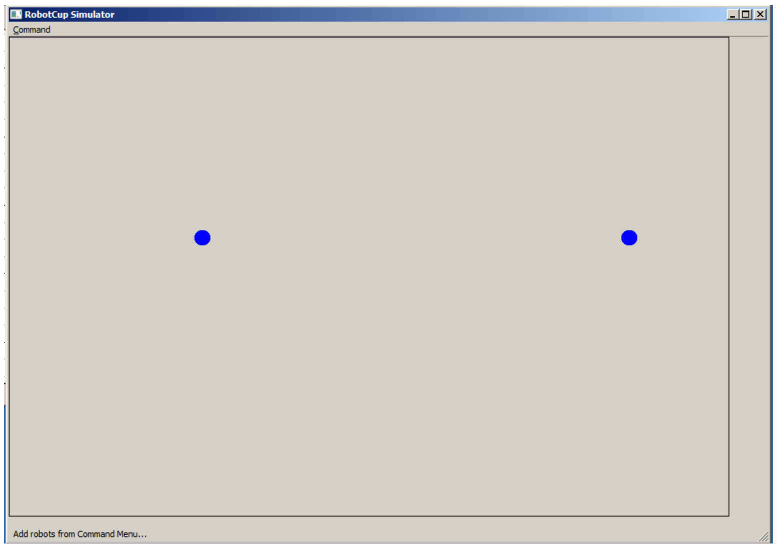


Figure **81.** Two Robots Simulation 4

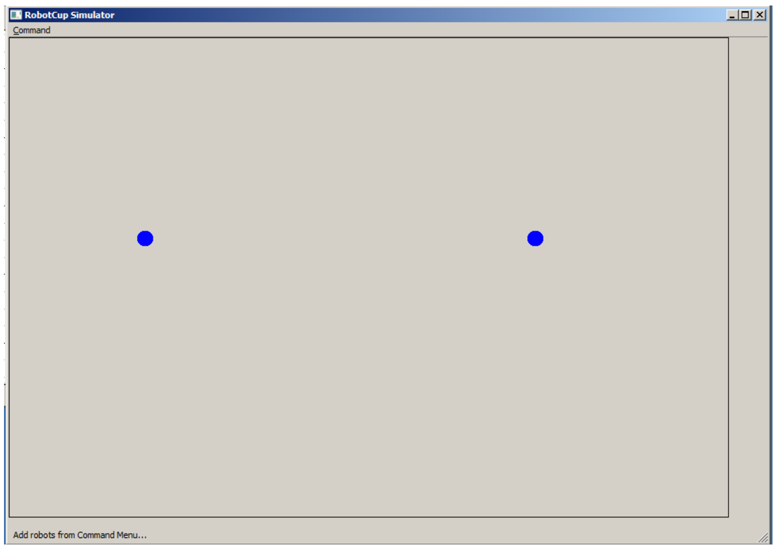


Figure **82.** Two Robots Simulation 5

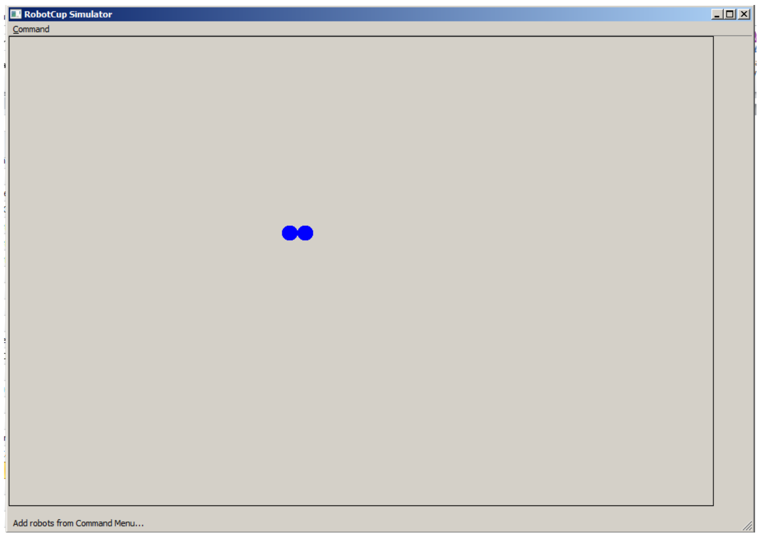


Figure **83.** Two Robots Simulation 6

From Figure 31 to Figure 33 we can see that robot1 first moves to left with the velocity decelerated to zero, then it moves to right with an acceleration and finally it decelerates again to zero velocity to avoid collide with robot2. This can also be seen clearly from the velocity-time chart in Figure 34.

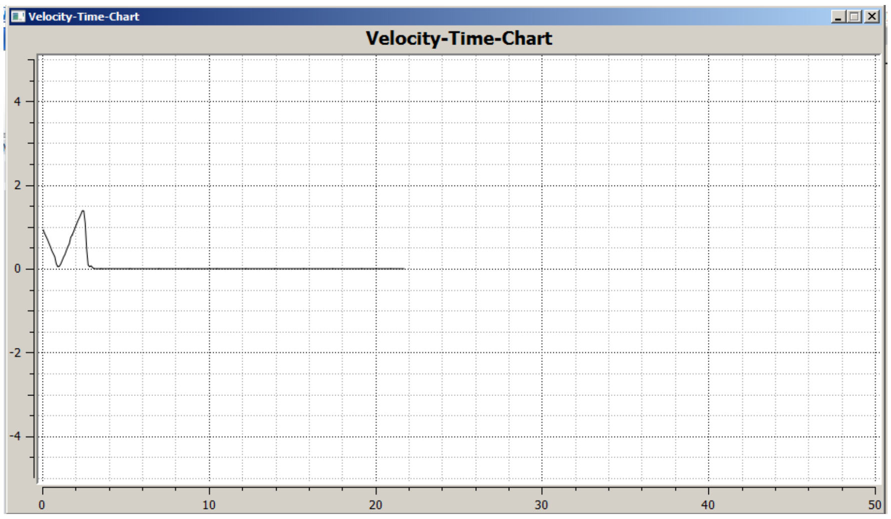
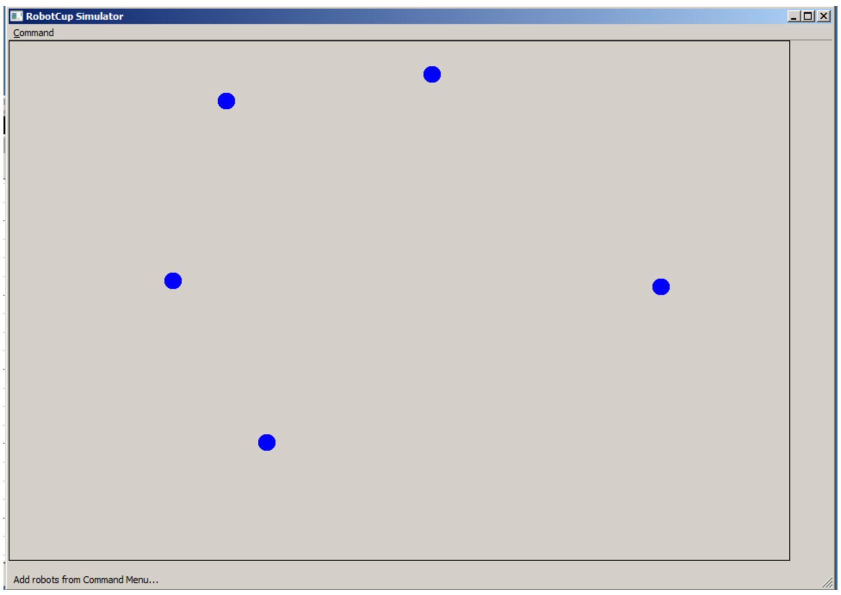


Figure **84.** Velocity-Time Chart

This program can also support multiple robot agents as promised in the algorithm. Here is an example to roughly simulate the situation in Figure 16.



Add robots from Command Menu...

Figure **85.** Multiple Robots Simulation 1

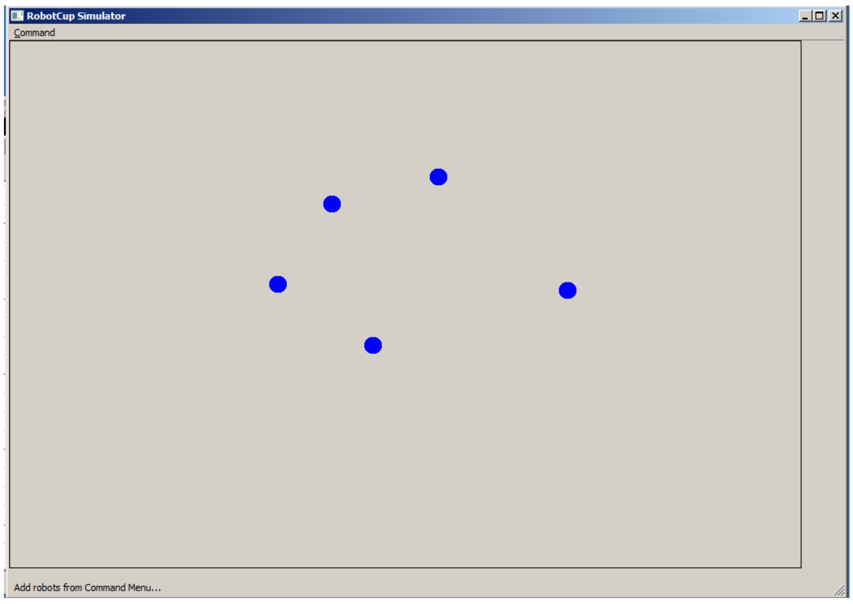


Figure **86.** Multiple Robots Simulation 2

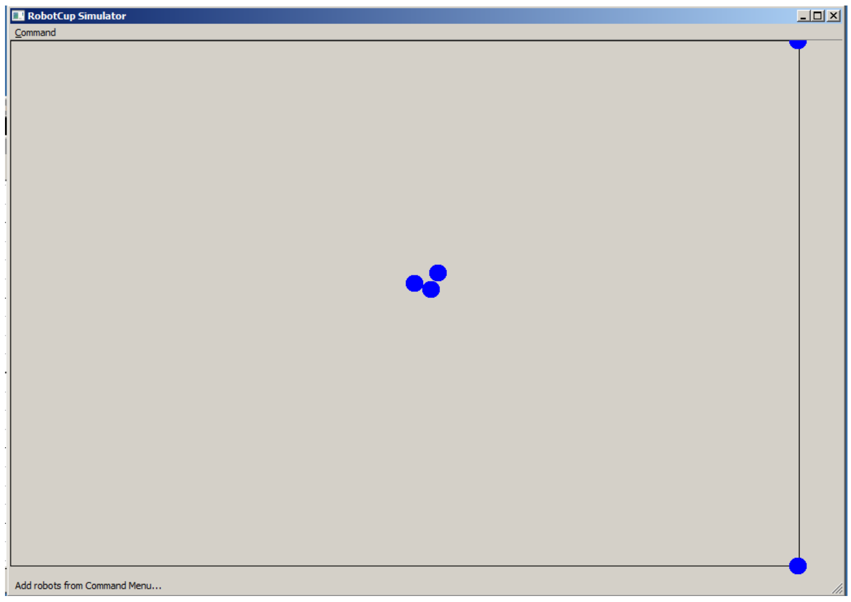


Figure **87.** Multiple Robots Simulation 3

It can be seen that for multiple robot agents moving at the same time, this algorithm can still handle the system well. Those robots (Robot 1, 3, 4) that may collide with each other should not interfere the movement of other robots (Robot 2, 5). (Robot ID is made based on Figure 16)

By testing, the algorithm proves itself to be an effective one and the implementation is also a successful work.

6.4 Summary

In this chapter, we have implemented a crucial algorithm to maintain safety among multiple robot agents in our team. Collision detection and prevention has always been a critical topic in the domain of robotics. The application of Dynamic Safety Search algorithm can also be used in automation and aeronautics, for example unmanned aerial vehicle.

7 MOTION PLANNING

7.1 Introduction of Motion Planning

In Chapter 6, we mention that in order to prevent the potential collision against opponent robots, we require to employ a separate algorithm in the RoboCup SSL software system. The algorithm we used here is named Extended Rapidly-exploring Random Tree (ERRT) Algorithm.

Motion planning is a field in robotics on the topic of how to produce a valid path and navigate the robot from the initial position to the goal position without colliding with obstacles in that domain.

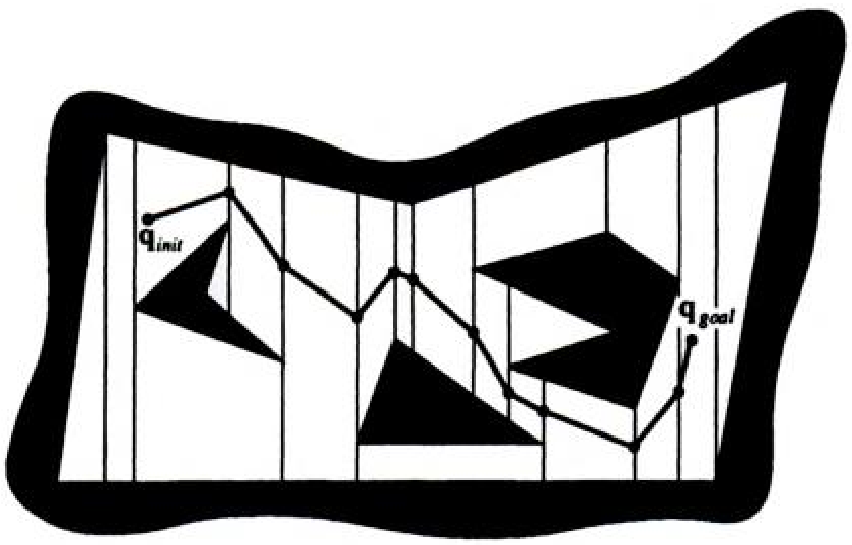


Figure **88.** Illustration of Motion Planning [29]

For instance, in Figure [29], there is a domain contains multiple obstacles and an initial position and a goal position. The robot needs to construct a valid path to navigate itself to successfully reach the goal position.

Motion planning together with safety navigation we introduced in Chapter 4, we can construct a complete motion system to apply it in the RoboCup SSL competition.

7.2 Possible path planning solutions

There are several existing path planning algorithms available, however, they have their own defects. [30] [31]

1) Geometric Algorithms (e.g., Visibility graph, Cell decomposition)

Algorithms of this category cannot scale well with the number of obstacles.

2) Grid-Based Search Algorithms (e.g., A*, D*, Field D*, Dijkstra's algorithm)

Algorithms of this category require an explicit representation of the free space, leading to computational inefficiency. For instance, A* algorithm may try all edges, while RRT can probabilistically subsample all edges.

3) Potential Fields

Many heuristic parameters must be adjusted for each individual problem.

4) Sampling-Based Algorithms

Ariadne's Clew algorithm: Difficult to solve optimization problems.

Expansive-space planning: Requires substantial parameter tuning for different problems.

Random-walk planner: Has trouble to move across long, winding domains.

7.3 Introduction on Rapidly-exploring Random Tree (RRT) Algorithm

Rapidly-exploring Random Tree (RRT) is an efficient algorithm and data structure to explore and make path planning in a non-convex and high-dimensional space without colliding with obstacles. RRT algorithm was first developed by Steven M. Lavalle and James Kuffner.

RRT algorithm has many practical applications in the field of robotics, gaming and aeronautics. It can be applied to high-dimensional space; however, we only apply it in a two-dimensional domain in the RoboCup SSL.

The main advantage of RRT algorithm is that it can find a valid path in a complex domain in most of the times. However, the found path does not guarantee to be the optimal one.

In order to find a valid path in Figure 38, here is how RRT algorithm does:

1) In a general configuration space C, we first set the root of the tree to be the initial point q_{init}.
2) Then we generate a random point q_{rand} in the collision-free space C_{free}.
3) Then we choose the nearest vertex $q_{nearest}$ to the random pointq_{rand} in the tree.
4) Then we expand a certain distance v from $q_{nearest}$ directly to q_{rand}, thus we create the point q_{new}.
5) If the q_{new} is not locate in the obstacle space C_{obs}, we add this point and the new edge into the tree. Otherwise, we go back to step 2.
6) Keep looping from step 2 to step 5 until q_{new} is close enough goal position q_{goal}.

Figure 39 is a graphical illustration of how the RRT algorithm is executed. In the figure, the four black points in a rectangle box represent an existing RRT tree. The green point represents the generated random point in the collision-free space and the yellow point represents a new point generated in the direction from the nearest point in the RRT tree towards the random green point. Finally, the initial point and goal point are also represented in the figure.

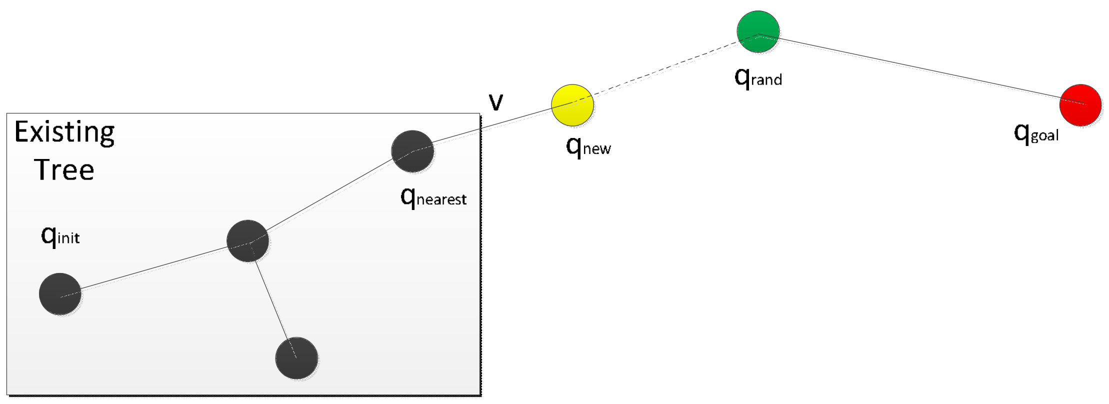


Figure **89.** Illustration of RRT Algorithm

The following figure gives a simulation of the growth of an RRT tree.

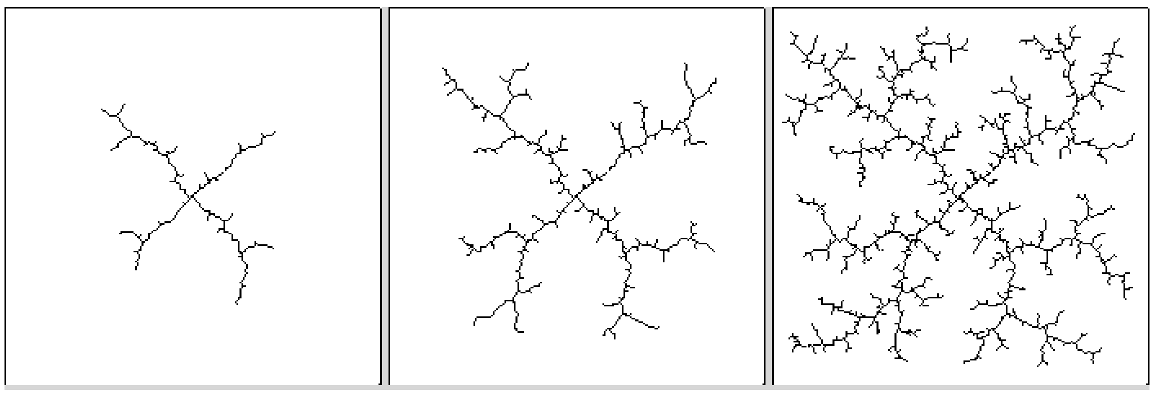


Figure **90.** Growth of RRT Tree [32]

Starting from a single initial point, the RRT tree will quickly expand the whole configuration space and find a valid path towards the goal position.

Although the RRT algorithm has been an effective method when we are dealing with path planning problem, there still many improvements can be made to optimize the efficiency of algorithm.

7.4 Robot model used in SSL

In the SSL competition, we used a holonomic robot model which has the following assumptions [22].

1) The robot has a safety radius.

2) The robot has control over its acceleration within some set.

3) The robot has a maximum deceleration for emergency stop.

4) The robot has a maximum allowed velocity.

The 4-omni-wheel holonomic soccer robot we used in the competition is shown in Figure 2.

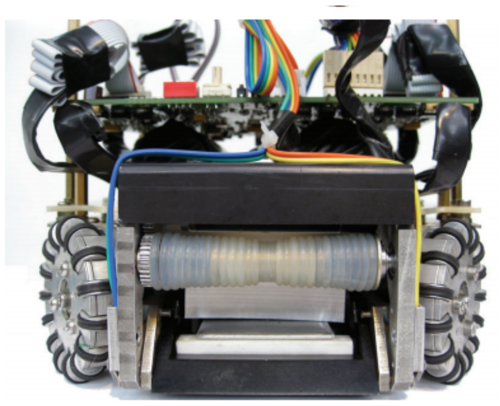


Figure **91.** Soccer Robot in SSL

7.5 RRT and RRT* features and issues

RRT algorithm has some significant features:

- No need to have an explicit representation of the free space.
- Scale well to changes in the environment, e.g.: obstacles.
- Effectively handle systems with complex constraints.

- State-of-the-art and most widely used robot path planning algorithm today.

Some more work can be done to improve the effect of RRT:

- By using the RRT* algorithm to optimize the path.
- Path smoothing technique.
- Search speed can be improved by constructing two RRT tree in both initiate position and the goal position.

Here is an implementation of normal RRT and RRT* algorithm.

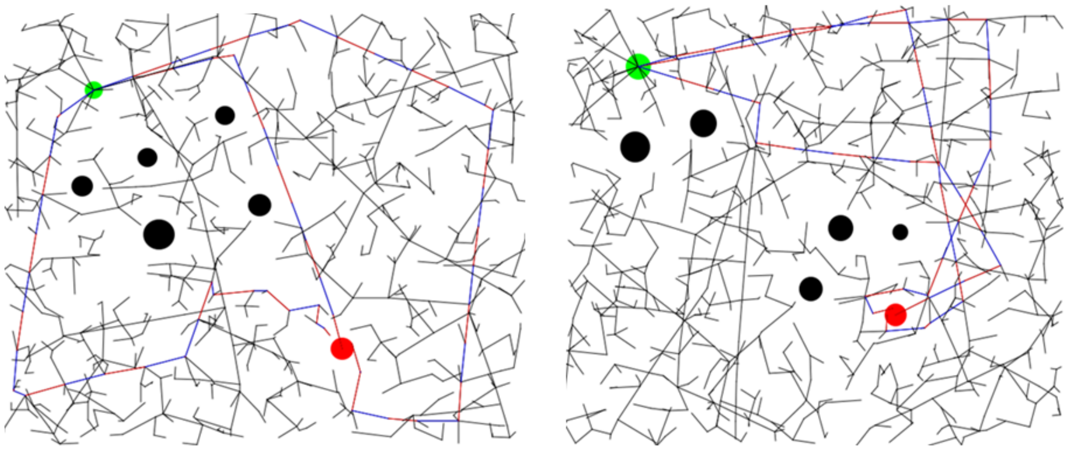


Figure **92.** Multiple Run of RRT

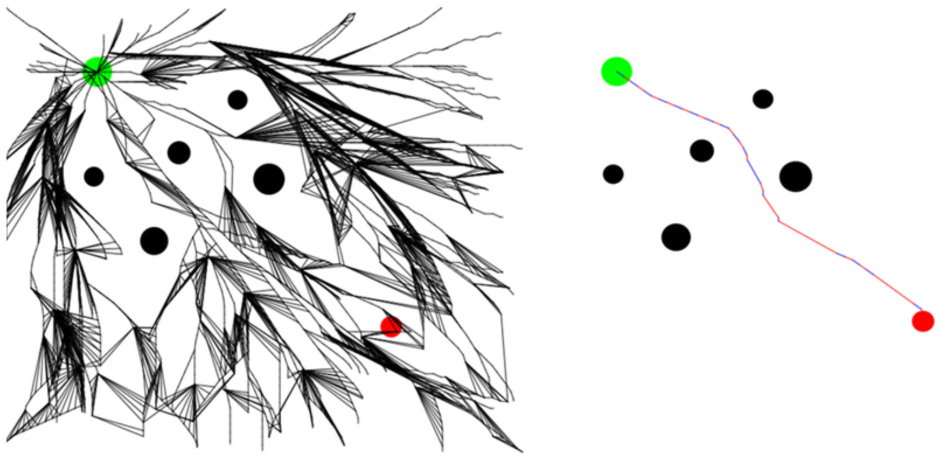


Figure **93.** Run of RRT*

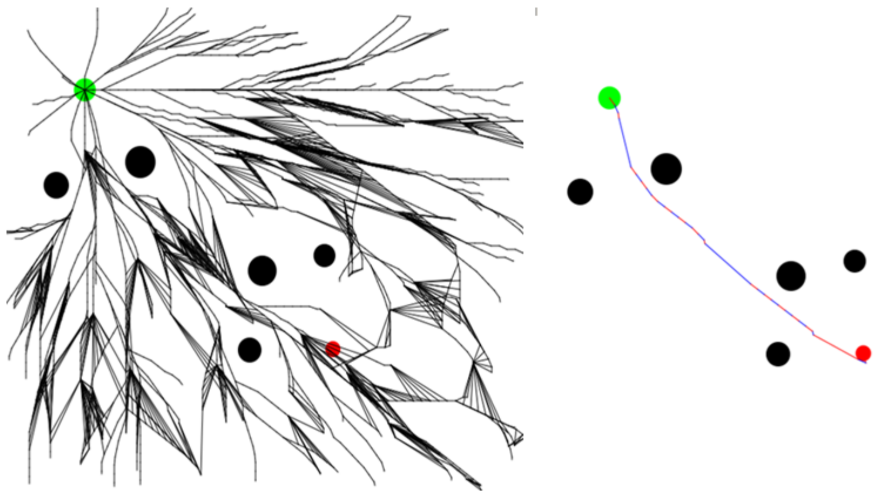


Figure **94.** Run of RRT*

7.6 Safety-guaranteed RRT

Despite the fact that RRT algorithm has been an effective method when we are dealing with the path planning problem, it fits not that well when applying into a real field environment.

It has the following flaws when we apply the baseline RRT algorithm into RoboCup SSL competition.

1) In RRT, it presumes the moving object to be a particle without considering the physical feature of the moving object.

2) In RRT, it assumes the moving object is moving at a constant velocity, which is not feasible for a reality robot agent.

3) In RRT, it does not consider the dynamics feature of robot agent, thus the generated path cannot guarantee safety in the dynamic domain.

Based on the robot model mentioned in the previous part, the following dynamics formula can be listed:

$$v_f = v_i + at \qquad (1)$$

where v_f is the final velocity, v_i is the initial velocity, a is the acceleration and t is the time of duration.

$$x_f = x_i + v_i t + \frac{1}{2} at^2 \qquad (2)$$

where x_f is the final position, x_i is the initial position, v_i is the velocity, t is the time of duration, and a is the acceleration.

Obviously, the moving velocity of each robot agent cannot exceed the maximum allowed value V_{max}. Therefore, we have the formula:

$$\|v_i + Ca_i\| \leq V_{max} \qquad (3)$$

where v_i is the velocity of i-th robot, a_i is the acceleration, and C is the time for a fixed control period.

Suppose the emergency stop maximum deceleration is D, then we have:

$$v_i + D \cdot t = 0 \qquad (4)$$

where t is the time required to guarantee robot come to a stop under maximum deceleration and current velocity.

So, it is possible to calculate the safety distance S required to guarantee deceleration without hitting any obstacle.

$$S = \left\| \frac{v_i^2}{2 \cdot D} \right\| \qquad (5)$$

In order to maintain safety, among moving robot with obstacles, the safety distance S is needed to be considered in the RRT algorithm.

Therefore, we have an improved version of RRT algorithm named safe rrt which is described in pseudo-code:

```
proceduresafe_rrt() : Path
    startNode ← new MyNode
    myTree ← new RRT Tree
    myTree.addNode(startNode, NULL)
    while not found do
    randomNode ← getRandomNode(myTree, env)
    nearestNode ← getNearestNode(myTree, random Node)
```

```
        if validSegment(nearestNode, randomNode) do
                newNode ← createNewNode(nearestNode,
        randomNode, maxLen)
                myTree.addNode(newNode, nearestNode)
                nearestNode ← newNode
                if hit Target do
                found ← true
                return path
                end
        end
end
```

The safe_rrt() procedure includes some sub-routines which we have applied safety check in it.

In it, getRandomNode() is a function, which returns a random node in the given environment domain without colliding with any obstacles.

```
functiongetRandomNode(myTree:Tree, env) : MyNode
while loop do
randomX ← randomized x inside env
randomY ← randomized y inside env
randomNode ← new MyNode(randomX, randomY)
```

```
nearestNode ← getNearestNode(myTree, random Node)
nearestX ← nearestNode.x()
nearestY ← nearestNode.y()
rotateAngle ← atan((randomY-nearestY) / (randomX-nearestX))
rect ← create rectangle based on safe distance and rotate angle
foreach obstacle do
    if collide(obs, rect) do
    collide ← true
    end
end
if !collide do
    loop ← false
    return randomNode
end
end
end
```

In getRandomNode() function, we construct a virtual rotated rectangle boundary based on the random node and the nearest node, then check it against each obstacle. If no collision happens, the function returns the random node.

```
functiongetNearestNode(myTree, randomNode) : MyNode

x ← randomNode.x()

y ← randomNode.y()

min ← max value available

for node in myTree do

d ← distance between randomNode and node

if d< min do

        min ← d

        nearestNode ← node

end

end

return nearest Node

end
```

IngetNearestNode(), we compare the randomized node against each node from the tree to calculate the distance between them. Therefore, we get the nearest node from the existing tree.

```
function validSegment(nearestNode, randomNode) : bool

randomX ← randomized x inside env

randomY ← randomized y inside env
```

```
randomNode ← new MyNode(randomX, randomY)
nearestNode ← getNearestNode(myTree, random       Node)
nearestX ← nearestNode.x()
nearestY ← nearestNode.y()
rotateAngle ← atan((randomY-nearestY) / (randomX-nearestX))
sweepRect ← create rectangle based on safe distance and rotate angle
foreach obstacle do
        if collide(obs, sweepRect) do
        return false
        end
end
return true
end
```

Function validSegment() calculates whether any collision may happen in the way between the nearest node and randomized node. It works by constructing a virtual rectangle which sweeps the rectangle area covered from the nearest node to the randomized node.

7.7 Testing

We have implemented the novel algorithm into a C++ based simulation software to better illustrate.

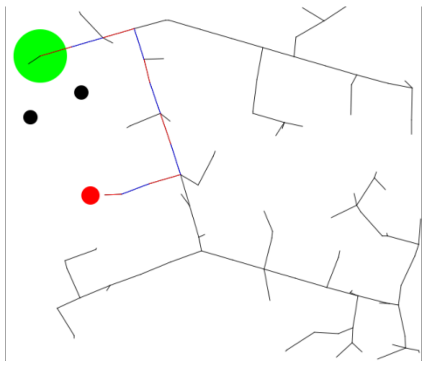


Figure **95.** Demo 1

Figure 3 demonstrates an example which employed the safety rrt algorithm. The green circle indicates the start position of the robot and its size. The red circle indicates the goal position for the robot, while two black circle indicates the obstacles existing in the field. By running the algorithm, it returns a valid highlight path for the robot navigation.

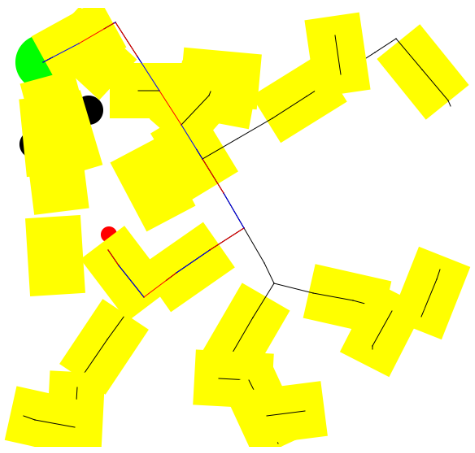


Figure **96.** Demo 2

In Figure4, it shows how getRandomNode() function works in the algorithm. The virtual yellow rectangles confined by the safety distance are created to guarantee no collision happens for the new randomized node.

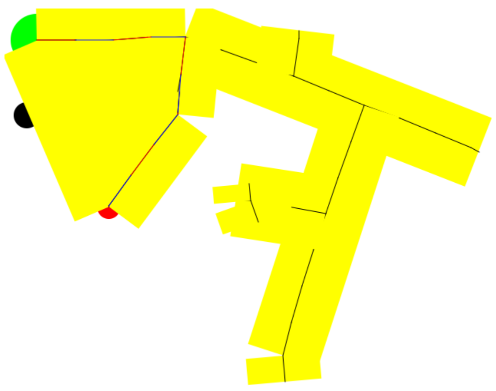


Figure **97.** Demo 3

In Figure5, it shows how validSegment() works in the algorithm. The virtual yellow rectangles are confined by the area which cover from the nearest node to the random node. Thus it guarantees no collision may happen in the path.

7.8 Summary

RRT algorithm excels at exploring free space in larger environments and it is parallelizable.

From the experiments and analysis, it can be verified that the new safety rrt algorithm is more applicable to the reality field environment such as RoboCup SSL competition because of the safety mechanism.

The safety distance should be decided based on the physical configurations of the robot. In this paper, we use a rectangle-bound area to calculate the collision possibilities. For future work, a partial ellipse bound area can be used to make comparisons.

8 OVERVIEW OF TO-DO LISTS IN BOTNIA SSL TEAM

8.1 Structure

In order to build a complete new Linux-based strategy software system, many tasks are required to be done. Since RoboCup is a relatively large project, among all those tasks, there are some tasks which are especially crucial to the success of the whole system need to be done in a relatively higher priority. Taking the bottom-up development approach, here is a graph which briefly describes these tasks.

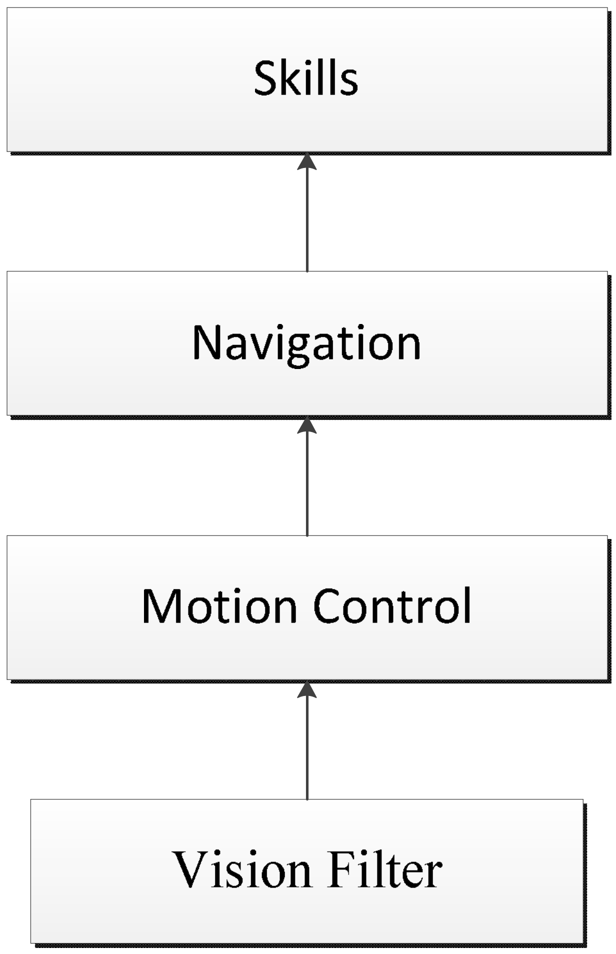


Figure **98.** the Hierarchy of Priority Tasks

From the bottom to top, there are mainly four major tasks which serve as the foundation of the whole RoboCup Botnia software system.

8.2 Vision Filter

From Figure 2 we can see that there is a high definition camera that oversees the whole field in real time, and this camera transmit captured vision frames to the vision server and the strategy server which runs our strategy software system in every frame interval. However, the captured vision frame is not noise-free. But instead, together with valuable vision information, vision frame also includes a certain level of random Gaussian white noise produced during the capture and transmission.

In order to remove these unwanted Gaussian white noise to improve the accuracy of computation, a vision filter needs to be employed in the software system.

In the Chapter 3 of this thesis, Kalman Filter will be introduced in detail.

8.3 Motion Control

8.3.1 Introduction

Motion control has been widely used not only in robotics, but also in the whole industrial control systems.

Common motion controls used in RoboCup are velocity control, position control, force control, power control, etc. These controls are vital to the accuracy of decision making in the upper-layer strategy software system.

For instance, the wheel velocity of soccer robots can be controlled by output PWM (Pulse Width Modulation) value which is determined by the strategy software system. If we hope the robot moves in a straight line, then the value of velocity of all wheels should be set identically by the system. However, it is very unlikely that all wheels can run at the same velocity for a relatively long distance because of possible flaws from the inner motors and all kinds of disturbance from the outside environment.

In order to provide a better motion performance and anti-disturbance capability, motion control needs to be implemented in the system. More specifically, some auto-tuning methods need to be integrated.

8.3.2 Bang-bang controller

Bang-bang controller is a three-position (negative, zero, and positive) feedback controller.

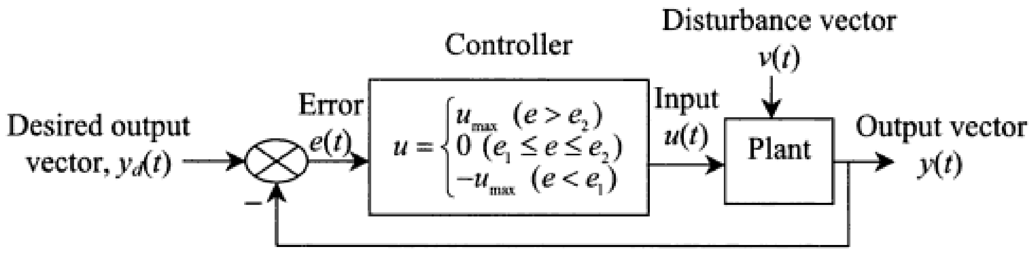

Disturbance vector
$v(t)$
Controller
Desired output vector, $y_d(t)$
Error $e(t)$
$u = \begin{cases} u_{max} & (e > e_2) \\ 0 & (e_1 \le e \le e_2) \\ -u_{max} & (e < e_1) \end{cases}$
Input $u(t)$
Plant
Output vector $y(t)$
−

Figure **99.** A Bang-bang, Closed-loop Control System [33]

From Figure 6, we can see that in Bang-bang controller, the controller generates input u (t) for the plant based on a calculated error value e(t) and its algorithm.

The error value e(t) is calculated by the difference of desired output vector $y_d(t)$ and observed output vector y(t)

$$e(t) = y_d(t) - y(t) \qquad (1)$$

Base on calculated e(t), bang-bang controller can determine the input value u(t) for the plant:

$$u(t) = \begin{cases} u_{max}, & when\, e > e_2 \\ 0, & when\, e_1 < e < e_2 \\ -u_{max}, & when\, e < e_1 \end{cases} \qquad (2)$$

In the formula, u_{max} is a preset maximum value.

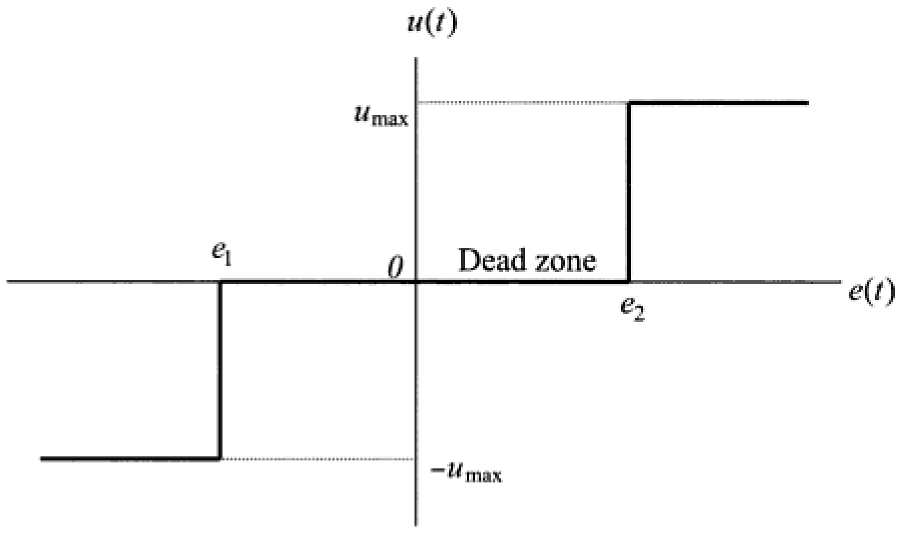


Figure **100.** Output of Bang-bang Controller (input of plant) [33]

The bang - bang controller can always product the constant maximum positive or negative value when the error value e (t) is not located in the dead zone. Thus, bang-bang controller is more reasonable when it is used in the plant environment that requires a constant and maximum input.

8.3.3 PID Controller

Proportional Integral Derivative (PID) Controller is a widely used loop feedback controller in robotics and automation.

PID Controller can minimize the difference between the desired set value and actual measured value by tuning the suitable value for the three controllers (PID) involved in the control process.

Proportional controller or P controller's output value is proportional to the calculated error value through the parameter K_p. Integral controller or I controller's output is proportional to the amount of time that errors have been accumulated. Derivative controller or D controller's output value is proportional to the changing rate of error with respect to time.

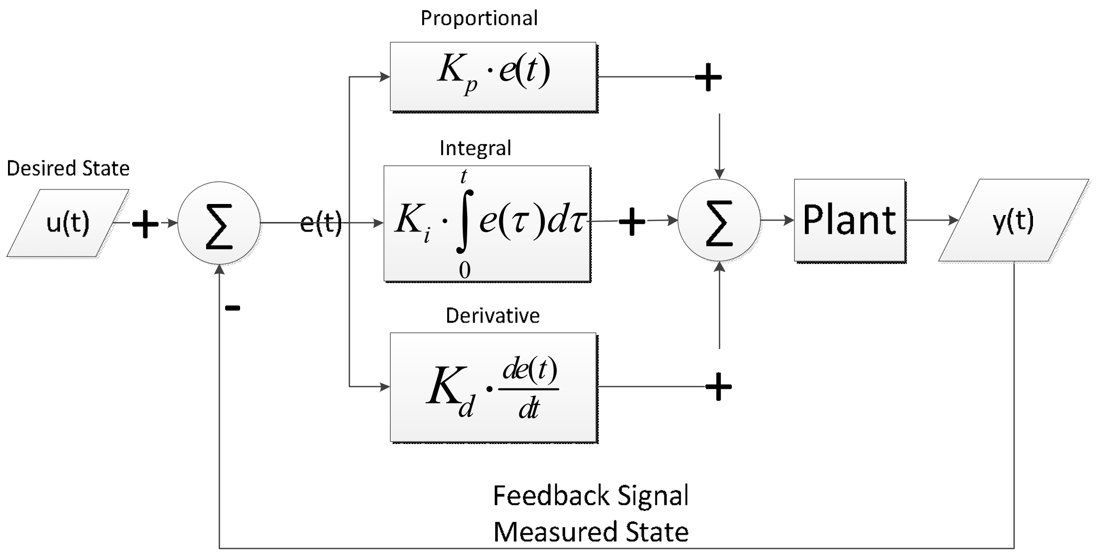


Figure **101.** PID Controller Block Diagram

In the Figure 8, K_p is the proportional gain, K_i is the integral gain, and K_p is the derivative gain.

The formula for PID Controller is:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \qquad (3)$$

In RoboCup competition, PID controller needs to be integrated in order to achieve a better performance. By automatically tuning three parameters in PID controller with the algorithm like genetic algorithm, it can alleviate the interferences caused by environment.

8.4 Navigation

8.4.1 Introduction

With a robust support from the previous two steps, the major task for this step is related to the problem on how to direct a robot from one position to another desired position efficiently and safely.

In the RoboCup Small Size League, there are two adversarial teams with five soccer robots per team moving at the same time in the field. Because of the fact that the "world" of the field may keep changing on every control interval, team robot players need to recalculate to find a suitable path to move without colliding with any other robots in the field. It requires the strategy software system to employ some high-efficiency algorithms to achieve this goal. Two well-known algorithms frequently will be introduced here.

8.4.2 ERRT

Rapidly-exploring Random Tree (RRT) is an algorithm developed by Steven M. Lavalle and James Kuffner. RRT has been widely used to efficiently solve the problems about searching non-convex, high-dimensional space.

Extended Rapidly-exploring Random Tree (ERRT) is an improved searching algorithm based on RRT with optimization on many aspects. By using ERRT algorithm, it is possible for the robot to build a viable path to the target position on every control cycle.

Details related to ERRT algorithm will be illustrated in Chapter 4.

8.4.3 Safety Navigation

Safety Navigation Algorithm, which is developed by James Robert Bruce from Carnegie Mellon University, is a novel algorithm for keeping safety among a set of cooperating robots. Under conditions of noiseless sensing, perfect dynamics, and perfect communication, this algorithm can guarantee no collisions will take place as the robots move about the environment. [28]

This algorithm is crucial for maintaining a non-collision system for our team robots. It is documented that by employing this algorithm into the strategy software system, the number of collisions between high-speed SSL soccer robots is dramatically reduced.

Details related to Safety Navigation algorithm will be illustrated in Chapter 5.

8.5 Skills

Skills are a series of motions to achieve a desired goal. With the support from the previous three steps, it is possible to make soccer robots accomplish some basic skills, for example moving in a straight path, or sine path, moving around a circle, kicking, dribbling, passing balls and simple cooperation between two robots like "one-touch shot" strategy.

The topic on how to assemble multiple basic skills to build up the effective attack or defence strategies for soccer robots is beyond the scope of this thesis. Some effective strategies have already been developed and existed in our code base.

8.6 Summary

This chapter, we describe some of the most important to-dos to build up a sound and robust system in a bottom-up approach. Many works have already been done for these four modules in Botnia SSL Team.

Next chapter, we will start describing the principle of Vision Filter module in detail and how it can be implemented and integrated into the whole system.

9 EXPECTATION OF RESEARCH DIRECTION

9.1 Complex Network Analysis

9.1.1 Theoretical Definition

- Complex graph

Real-world graphs $G\ (n, m)$ (n nodes, m edges) are more or less “complex” in the sense that different topological features deviate from random graphs. [34]

- Betweenness

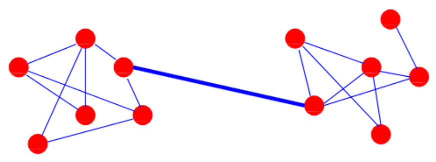


Figure **102.** a Typical Strong Betweenness Connection

Edge betweenness is the total number of shortest paths from any pair of vertices that cross the edge(Anthonisse, 1971)and it is omputable with algorithms based on breadth-first-search,with complexity O(m*n) (Brandes, 2001)

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \qquad (1)$$

Whereσ_{st} is the total number of shortest paths from nodes to nodet and $\sigma_{st}(v)$ is the number of those paths that pass through. (Freeman, Linton, 1977)

Figure **103.** Formula of Betweenness Connection

9.1.2 Problem and Potential Solution

In the field, there are twelve robots going to compete in two teams. A good algorithm should not only focuses on the completion of the robot's movement of its own team, but also detect the robots in the other team. In doing this, the algorithm should have the intelligence to make a deduction on the tendency of its opponent. At first I considered if we have enough resources, we may use the machining learning algorithm to accomplish this goal, and then the problem is laying on modelling of classification of different situation. However, according to the evaluation of the modelling process, it is quite time-consuming. In fact, a mature and ready-established logic is much easier to adopt and to apply.

A complex network was initially studied by people's curiosity on social networks. After researching for years, it was adopted for the analysis of internet social network. My suggestion is that we also can apply and enhance the algorithms to this RoboCup game. As we only have six robots in the field, the computation would not be very complicated. As a consequence, it is reasonable to make a research on this area.

The communities of robots can be defined through a fitness measure.

For example, in a famous paper wrote by Newman& Girvan in 2004 describes the steps of the algorithm as following:

Girvan-Newman algorithm

1. Calculate the betweenness of all edges
2. Remove the one with highest betweenness
3. Recalculate the betweenness of the remaining edges
4. Repeat from 2

The complexity of the algorithm is O (m n) O (n3) [on a sparse graph]

9.1.3 Research Perspective

The problems in our case are similar but still different.

1. There are six robots in opposing team and there is not known connection between any of them. But we can also assume that there exist virtual connections between each of them. We probably need to do theoretical work to formulate this kind of analysis.

2. In the RoboCup game, unlike a social network, the more frequent cooperation is between two robots. The cooperation between several robots should be considered to give a penalty factor.

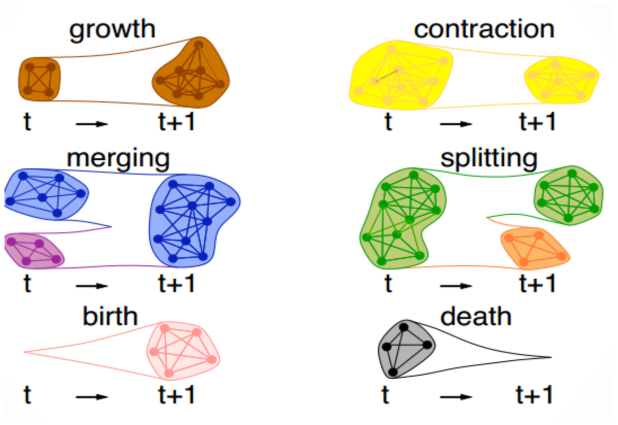


Figure **104.** Types of Dynamic Community

3. Besides the cooperation detection, we also can have the community detection. This function is essential for cooperation detection. A good algorithm in the community detection can improve the accuracy of cooperation detection dramatically.

Auther	Label	Order
Girven & Neman	GN	$O(n * m^2)$
Clauset et al.	Clauset et al.	$O(n * \log(n)^2)$
Blondel et al.	Blondel et al.	$O(m)$
Guimera et al.	Sim. Ann	Parameter dependent
Radicchi et al.	Radicchi et al.	$O(m^4/n^2)$
Palla et al.	Cfinder	$O(\exp(n))$
Van Dongen	MCL	$O(n * k^2)$, $k < n\, parameters$
Rosvall & Bergstorm	Infomod	Parameter dependent
Rosvall & Bergstorm	Infomap	$O(m)$
Donetti & Munos	DM	$O(n^3)$
Newman & Leicht	EM	Parameter dependent
Ronhovde & Nussinov	RN	$O(n^\beta)$, $\beta \sim 1$

Figure **105.** Different Algorithms Related to Cooperation Detection. [35]

1. Monitoring the past actions of the opponent robots can be also a contribution to the analysis. By considering this, the process then will still remain as a stochastic learning process based on the complex network.

9.2 Expert System and Inference System

9.2.1 Theoretical Definition

- Expert System

The expert system is a computer software that emulates the decision making process of a human expert. The expert system can normally contribute to the development of a complex system that needs a large scale of knowledge base.

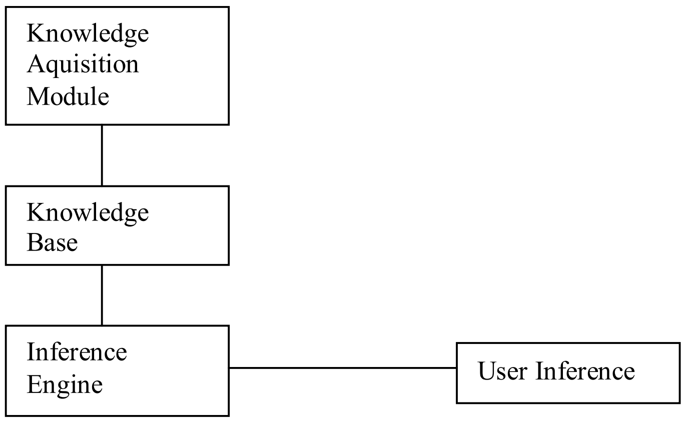


Figure **106.** Components of Expert System [36]

- Inference System

The expert system contains two key concepts. One of them is the knowledge base and another of them is the inference system. As one key concept in the expert system, the inference system normally can establish the connection between the knowledge base and the query. Inside the inference system, the knowledge is normally listed as a set of rules formalized in a specific language.

Normally we store several kinds of logic into the system. E.g. propositional logic, predicates of order 1 or more, epistemic logic, modal logic, temporal logic and fuzzy logic.

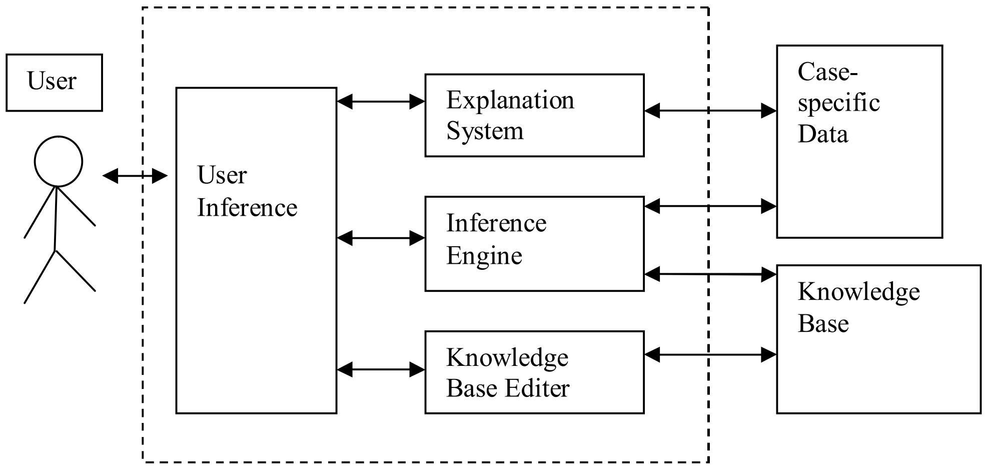


Figure **107.** User's View of Expert System [37]

9.2.2 Problem and Potential Solution

The problem in our project is that our project is huge and complex. But we do not have enough time to manage the documentation in our daily lives.

The expected solution of this problem is to establish a system that can give information to the developer and the user by giving a reasonable quarry. We also can treat this system as an information system framework which we can use to store and analyse information related to this multi robot system.

9.2.3 Research Perspective

The research perspective can be focused on two points.

1. The first purpose to consider about is how to represent the relationship between the classes in an expert system.

2. An expert system is depending not only on the inference logics but also on the algorithms related to data mining and machine learning. The more real model of developing expert system can be combining inference system and also deduction

9.3 Heterogeneous Computing For Fast Computation

9.3.1 Theoretical Definition

- Heterogeneous Computing

The heterogeneous computing refers an electronic system which contains different computational unit. For example, the combination of CPU, GPU and FPGA can be called as a heterogeneous system.

By giving the tasks to the GPU, the performance can be increased rapidly. The following graph can show how the speed can be increased by heterogeneous computing.

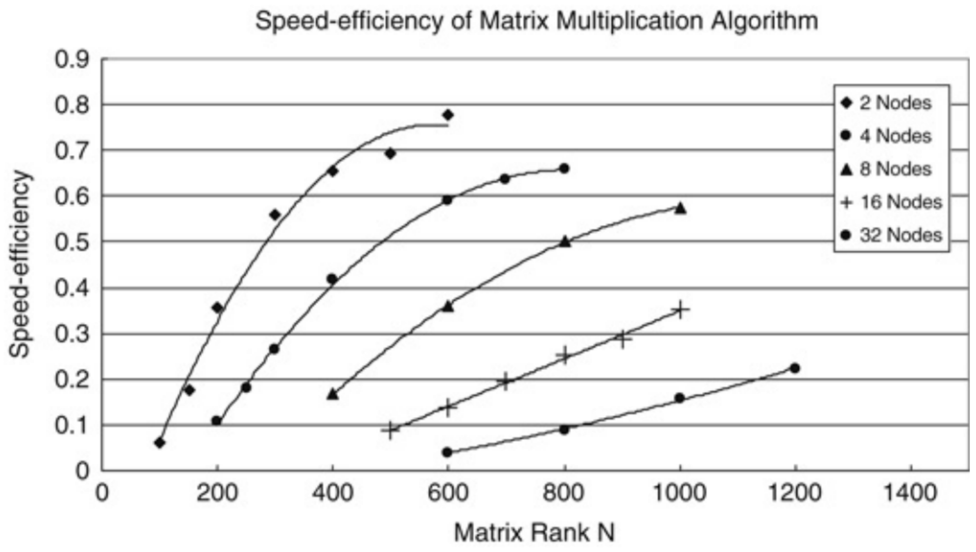


Figure **108.** Speed-efficiency of Matrix Multiplication Algorithm [38]

The New Landscape of Parallel Computer Architecture.1 Cyclotron Road, Berkeley California, 94720, USA

9.3.2 Problem and Potential Solution

In the near future, as we add more and more functionalities to the calculating process, probably, by using traditional methods, we cannot complete 60 times calculation within one minute, which is the frequency of receiving and analysing the information from the vision system. Although the RRTs algorithms have logarithmic complexity, as randomized algorithms, in some real cases, they still cannot perform very well.

Algorithm	Propabilistic Completeness	Asymptotic Optimality	Computational Complexity
RRT	Yes	No	$O(\log(n))$
RRT*	Yes	Yes	$O(\log(n))$
BRRT	N/A	N/A	$O(\log(m * 2n))$

Where N and n are the expended node number, m is the number of nodes on the boundary (details on BRRT algorithm).

Figure **109.** the Computational Complexity of Several RRT Algorithms [39]

	State 1	State 2	State 3	State 4	State 5
Success rate	100	100	99	100	16
Time(ms)	1.04	2.83	2.96	1.96	7.74
Length(mm)	3754	4096	4146	4229	4996

Figure **110.** Time Expenses of RRT Algorithm Using C++ [1]

	State 1	State 2	State 3	State 4	State 5
Success rate	100	100	100	100	100
Time(ms)	1.04	2.83	2.96	1.96	7.74
Length(mm)	3754	4096	4146	4229	4996

Figure **111.** Time Expenses of BRRT Algorithm Using C++ [1]

In this case, we need to focus on the methods which can improve the performance of calculation. The two obvious ways we have considered about are parallel computing and heterogeneous computing. The first way, parallel computing is easier to implement but it is also difficult to make optimization on improving the speed limit.

Caused by the time consumption of network communication, the heterogeneous computing, however, may have more possibility to help us to complete the mission. The combination of CPU and GPU can be a powerful mechanism to increase the speed as the GPU contains more floating point units.

9.3.3 Research Perspective

1. The heterogeneous computing is efficient in many ways, but it also requires an efficiently designed pattern for fast communication.

9.4 Conclusion

The previously listed views are only suggestions. If any of the direction turned out to be unsuitable for the future development of our project, you can drop it.

10 SUMMARY

At this moment, although the system did not really finish all functions that we intended to make at first place, we still think we have built a foundation for future development of the system.

We not only managed to build the strategy software, several algorithm simulators but also implemented the critical algorithms for this project. As a result the other team members can use what we have done to continue implementing the Strategy-Tactic-Play (STP) structure of the strategy software. Simultaneously they can also insert the implemented code to a newer version. The reliability of the strategy software and the algorithm simulators is relatively robust. There exist some bugs in the software, but the overall performance is modest.

The research contribution of this thesis lies in following aspects. Firstly we applied the EKBF to filter the noise of the ball. Secondly we innovatively simulated several crucial algorithms in the algorithms simulators. Thirdly, we used a newly designed structure to implement the strategy software. And finally, we suggested several research directions for the future development of the team.

However, our work also has some limitations. There is some functionality under development. The robustness and efficiency are not always ensured.

By standing on the ground of innovation, we also would like to encourage other members in the team to continue thinking about the solutions of the system. No matter it is related to improving the design or algorithm, we would like to hear your voice and we can evaluate the plan together.

The best way of thinking a new topic is to absorb the knowledge from a new field. For example we were taking a summer course called evolution algorithms for solving complex problems. The professor recommended us to research an uprising field called morphogenetic robotics. In fact, after considering the relationship between that topic and our robot project, several ideas showed up in our head. They actually attracted us a lot and we guess if we have time, we would like to have a deep dive into it. As a consequence, we also would like to encourage the people who consider doing research in robotics or finding a job related to robots to try hard to learn by themselves.

There are several sources of learning from internet. The first one is probably Coursera and Udacity. You can find a lot of courses related to artificial intelligence (AI) and robotics. The second way of checking details in your interesting subject is to do a simple search using Google scholar.

Both ways enable you to enjoy the process of developing concepts by yourselves. If you have really tasted the feeling of devotion, you know how they can give you the feeling of happiness.

We are here wishing a good luck to everyone who reads this a bright future.

11 APPENDIX

11.1 Qt Style Sheet for strategy software's user interface

```
/* The tab widget frame */
 QTabBar::tab {
background: gray;
border: 10px solid gray;
color: white;
 }
 QTabBar::tab:selected {
background: lightgray;
border-color: #9B9B9B;
border-bottom-color: lightgrey;
border-right-color: lightgrey;
 }
/* make non-selected tabs look smaller */
 QTabBar::tab:!selected {
margin-top: 6px;
 }
QGroupBox::title {
background: gray;
color: white;
padding: 10px;
 }

QGroupBox
{
background-color: gray;
margin-bottom: 2ex; /* leave space at the top for the title */
border: 10px solid lightgray;
}
QPushButton
{
color: white;
background: gray;
border-color: #9B9B9B;
border-top:transparent;
border-bottom:  transparent;
border-right: transparent;
border-left:  transparent;
padding: 10px;

}

 QPushButton::hover {
```

```
background: lightgray;
color: white;
padding: 10px;
 }
QPushButton:focus:active
{
background: lightgray;
color: white;
padding: 10px;
}
QRadioButton {
color: white;
background: gray;
border-top: transparent;
border-bottom: transparent;
border-right: transparent;
border-left: transparent;
padding: 10px;
}
 QRadioButton::hover {
background: lightgray;
color: white;
padding: 10px;
 }

QRadioButton:checked
{
background: lightgray;
color: white;
padding: 10px;
}
QCheckBox
{
color: white;
background: gray;
border-top: transparent;
border-bottom: transparent;
border-right: transparent;
border-left: transparent;
padding: 10px;

}
 QCheckBox::hover {
background: lightgray;
color: white;
padding: 10px;
 }
QTableWidget
```

```
{
color: white;
background: lightgray;
padding: 10px;

}
 QTableWidget::hover {
background: lightgray;
color: white;
padding: 10px;
 }
QHeaderView::section {
color: white;
background: grey;
border-top: transparent;
border-bottom: transparent;
border-right: transparent;
border-left: transparent;
}
```

11.2 DECT Description

Digital Enhanced Cordless Telecommunications (DECT™) is the ETSI standard for short-range cordless communications, which can be adapted for many applications and can be used over unlicensed frequency allocations world-wide.

DECT™ is suited to voice (including PSTN and VoIP telephony), data and networking applications with a range up to 500 metres.

DECT™ dominates the cordless residential market and the enterprise PABX (Private Automatic Branch eXchange) market. DECT™ is also used in the Wireless Local Loop to replace copper in the 'last mile' for user premises.

The ETSI Technical Committee DECT (TC DECT) has the overall responsibility over the technology. [40]

11.3 Complete code for Kalman Filter

11.3.1 Ball EKBF Implementations Step I: Theory & Configurations

Here is some frequently used data structures and functions excerpted from the EKF code frame.

```
//Data Structures:
//============================Dimensions=====================//
//Dimension of State Vector
```

```
int kalman_x_dimension;
//Dimension of Measuremen tVector
int kalman_z_dimension;
//Dimension of Control Vector
int kalman_u_dimension;
//Dimension of Process Noise Vector
int kalman_w_dimension;
//Dimension of Measurement Noise Vector
int kalman_v_dimension;

//=================================Vectors========================//
//State Vector
CvMat* kalman_x_last;//x_hat_k-1_k-1
CvMat* kalman_x_predicted;//x_hat_k_k-1
CvMat* kalman_x_updated;//x_hat_k_k

//Control Vector
CvMat* kalman_u;//u_k-1

//Measurement Vector
CvMat* kalman_z;//z_k
CvMat* kalman_z_predicted;//z_hat_k

//===============================Matrices=======================//
CvMat* kalman_P_last;//P_k-1_k-1
CvMat* kalman_P_predicted;//P_k_k-1
CvMat* kalman_P_updated;//P_k_k

CvMat* kalman_A;//Jacobian matrix of partial derivatives with respect to x
CvMat* kalman_W;//Jacobian matrix of partial derivatives with
       //respect to w

CvMat* kalman_H;//Jacobian matrix of partial derivatives with
                      //respect t ox
CvMat* kalman_V;//Jacobian matrix of partial derivatives with
       //respect to v

CvMat* kalman_Q;//Process noise covariance matrix
CvMat* kalman_R;//Measurement noise covariance matrix

CvMat* kalman_S;//Innovation or residual covariance matrix
CvMat* kalman_K;//Kalman gain
```

```
//Functions:

//==========================Auxiliary Functions==========================//
Int CheckDimension(CvMat* M1,CvMat* M2);
Int GetSubMatrix(CvMat* M1,CvMat* M2,int start_row,int start_col);
Int SetSubMatrix(CvMat* M1,CvMat* M2,int start_row,int start_col);

//Combine M1 M2 as [M1 M2]
Int RowCombineSubMatrix(CvMat* M1,CvMat* M2,CvMat* M3);
//Combine M1 M2 as [M1;M2]
Int ColCombineSubMatrix(CvMat* M1,CvMat* M2,CvMat* M3);
//Delete M1(:,col) from M1
Int DelCol(CvMat* M1,CvMat* M2,int col);
//Delete M1(row,:) from M1
Int DelRow(CvMat* M1,CvMat* M2,int row);
//Insert column V before col(Obased)
Int InsertCol(CvMat* M1,CvMat* V,CvMat* M2,int col);
//InsertcolumnVtbeforerow(Obased)
Int InsertRow(CvMat* M1,CvMat* Vt,CvMat* M2,in trow);
Int OutputMat(CvMat* M,int order=0);

//=======================Output Matrices to Files=======================//
Int OutputMatFile(CvMat* M,char* filename,int num,int order=0);

//======================Initialize Functions============================//
//Note: start using EKF from here!

int SetDimensions(int x_dimension,
int u_dimension,
int z_dimension,
int w_dimension,
int v_dimension);
int InitMatrices();
int ClearMatrices();



//========================EKF Working Functions========================//

//=============================Setter===================================//
//Note: We should set this value from the camera raw data
//Se tMeasurement
Int Set_z(CvMat* z);

//Set process noise Q & measurement noise R
Int Set_NoiseCovariance(CvMat* Q,CvMat* R);

//==========================Getter=====================================//

//======The following pure virtual functions need to be rewritten======//
//====================(by calculating Jacobian function)==============//
//f(x,u,0)
virtual int Get_x_predicted()=0;

//A:=df/dx
```

```
//Jacobian matrix of partial derivatives off with respect to x
virtual int Get_A()=0;

//W:=df/dw
// Jacobian matrix of partial derivatives off with respect to w
virtual int Get_W()=0;

//H:=dz/dx
// Jacobian matrix of partial derivatives off with respect to x
virtual int Get_H()=0;

//zk:=h(xk-,0)
Virtual int Get_z_predicted()=0;

//V:=dh/dv
// Jacobian matrix of partial derivatives off with respect to v
virtual int Get_V()=0;

//========================Predict Step============================//
//P-:=A*P*At+W*Q*Wt
int Get_P_predicted();

//========================Update Step=============================//

//S:=H*P-*Ht+V*R*VtV=I
//Innovationorresidualcovariancematrix
int Get_S();

//K:=P*Ht*S_inv
//KalmanGain
int Get_K();

//x_updated:=x_predicted+K(z-z_predicted);
int Get_x_updated();

//P:=(I-K*H)*P_predicted
int Get_P_updated();
```

Initialize settings:

```
qDebug()<<"init_EKF() starts:";
temp_mat1=cvCreateMat(4,1,CV_32F);
temp_mat2=cvCreateMat(2,1,CV_32F);

last_raw_x=0.0;
last_raw_y=0.0;
last_raw_vx=0.0;
```

```
last_raw_vy=0.0;

//Initialize Kalma nFilter
//SetDimensions(intx_dimension,intu_dimension,intz_dimension,intw_dimension,int
v_dimension)
SetDimensions(4,4,2,4,2);
```

Set Process noise covariance matrix and measurement noise matrix:

```
//Process noiseco variance matrix, this should be very small(meaning very
reliable and accurate)
kalman_Q=cvCreateMat(2,2,CV_32F);
cvSetReal2D(kalman_Q,0,0,0.000001);
cvSetReal2D(kalman_Q,0,1,0);
cvSetReal2D(kalman_Q,1,0,0);
cvSetReal2D(kalman_Q,1,1,0.000001);

//Measurement noise covariance matrix, this should be big(meaning not reliable
and accurate)
kalman_R=cvCreateMat(2,2,CV_32F);
cvSetReal2D(kalman_R,0,0,5);
cvSetReal2D(kalman_R,0,1,0);
cvSetReal2D(kalman_R,1,0,0);
cvSetReal2D(kalman_R,1,1,5);

set_NoiseCovariance(kalman_Q,kalman_R);
```

Set “priori” and “posteriori” error covariance matrix:

```
kalman_P_predicted=cvCreateMat(4,4,CV_32F);
kalman_P_last=cvCreateMat(4,4,CV_32F);
kalman_P_updated=cvCreateMat(4,4,CV_32F);

cvSetReal2D(kalman_P_predicted,0,0,1000);
cvSetReal2D(kalman_P_predicted,0,1,0);
cvSetReal2D(kalman_P_predicted,0,2,0);
cvSetReal2D(kalman_P_predicted,0,3,0);

cvSetReal2D(kalman_P_predicted,1,0,0);
cvSetReal2D(kalman_P_predicted,1,1,1000);
cvSetReal2D(kalman_P_predicted,1,2,0);
cvSetReal2D(kalman_P_predicted,1,3,0);
```

```
cvSetReal2D(kalman_P_predicted,2,0,0);
cvSetReal2D(kalman_P_predicted,2,1,0);
cvSetReal2D(kalman_P_predicted,2,2,1000);
cvSetReal2D(kalman_P_predicted,2,3,0);

cvSetReal2D(kalman_P_predicted,3,0,0);
cvSetReal2D(kalman_P_predicted,3,1,0);
cvSetReal2D(kalman_P_predicted,3,2,0);
cvSetReal2D(kalman_P_predicted,3,3,1000);

cvSetReal2D(kalman_P_last,0,0,1000);
cvSetReal2D(kalman_P_last,0,1,0);
cvSetReal2D(kalman_P_last,0,2,0);
cvSetReal2D(kalman_P_last,0,3,0);

cvSetReal2D(kalman_P_last,1,0,0);
cvSetReal2D(kalman_P_last,1,1,1000);
cvSetReal2D(kalman_P_last,1,2,0);
cvSetReal2D(kalman_P_last,1,3,0);

cvSetReal2D(kalman_P_last,2,0,0);
cvSetReal2D(kalman_P_last,2,1,0);
cvSetReal2D(kalman_P_last,2,2,1000);
cvSetReal2D(kalman_P_last,2,3,0);

cvSetReal2D(kalman_P_last,3,0,0);
cvSetReal2D(kalman_P_last,3,1,0);
cvSetReal2D(kalman_P_last,3,2,0);
cvSetReal2D(kalman_P_last,3,3,1000);

cvSetReal2D(kalman_P_updated,0,0,1000);
cvSetReal2D(kalman_P_updated,0,1,0);
cvSetReal2D(kalman_P_updated,0,2,0);
cvSetReal2D(kalman_P_updated,0,3,0);

cvSetReal2D(kalman_P_updated,1,0,0);
cvSetReal2D(kalman_P_updated,1,1,1000);
cvSetReal2D(kalman_P_updated,1,2,0);
cvSetReal2D(kalman_P_updated,1,3,0);

cvSetReal2D(kalman_P_updated,2,0,0);
cvSetReal2D(kalman_P_updated,2,1,0);
cvSetReal2D(kalman_P_updated,2,2,1000);
cvSetReal2D(kalman_P_updated,2,3,0);

cvSetReal2D(kalman_P_updated,3,0,0);
cvSetReal2D(kalman_P_updated,3,1,0);
cvSetReal2D(kalman_P_updated,3,2,0);
cvSetReal2D(kalman_P_updated,3,3,1000);
```

Set other involved parameters matrixes, like innovation, Kalman Gain, received measurement matrix:

```
kalman_S=cvCreateMat(2,2,CV_32F);
cvSetZero(kalman_S);
kalman_K=cvCreateMat(4,2,CV_32F);
cvSetZero(kalman_K);
kalman_z=cvCreateMat(2,1,CV_32F);
cvSetZero(kalman_z);
kalman_z_predicted=cvCreateMat(2,1,CV_32F);
cvSetZero(kalman_z_predicted);
kalman_x_last=cvCreateMat(4,1,CV_32F);
kalman_x_predicted=cvCreateMat(4,1,CV_32F);
kalman_x_updated=cvCreateMat(4,1,CV_32F);
```

Set up A, W, H, V these four matrixes in their respective functions, and execute a check to ensure all settings are correct:

```
Get_A();
Get_W();
Get_H();
Get_V();

#if0
qDebug()<<"A:";OutputMat(kalman_A);
qDebug()<<"W:";OutputMat(kalman_W);
qDebug()<<"H:";OutputMat(kalman_H);
qDebug()<<"V:";OutputMat(kalman_V);
qDebug()<<"Q:";OutputMat(kalman_Q);
qDebug()<<"R:";OutputMat(kalman_R);

#endif
```

The four overwritten functions are listed and commented as follows:

```
//A:=df/dx
// Jacobian matrix of partial derivatives off with respect to x
int GLSoccerView::Get_A()
{
kalman_A=cvCreateMat(4,4,CV_32F);
cvSetReal2D(kalman_A,0,0,1);
cvSetReal2D(kalman_A,0,1,0);
cvSetReal2D(kalman_A,0,2,delta_t);
cvSetReal2D(kalman_A,0,3,0);

cvSetReal2D(kalman_A,1,0,0);
cvSetReal2D(kalman_A,1,1,1);
cvSetReal2D(kalman_A,1,2,0);
cvSetReal2D(kalman_A,1,3,delta_t);

cvSetReal2D(kalman_A,2,0,0);
cvSetReal2D(kalman_A,2,1,0);
cvSetReal2D(kalman_A,2,2,1);
cvSetReal2D(kalman_A,2,3,0);

cvSetReal2D(kalman_A,3,0,0);
cvSetReal2D(kalman_A,3,1,0);
cvSetReal2D(kalman_A,3,2,0);
cvSetReal2D(kalman_A,3,3,1);

//OutputMat(kalman_A);

return0;
}
```

```
int GLSoccerView::Get_W()
{
kalman_W=cvCreateMat(4,2,CV_32F);
cvSetReal2D(kalman_W,0,0,0);
cvSetReal2D(kalman_W,0,1,0);
cvSetReal2D(kalman_W,1,0,0);
cvSetReal2D(kalman_W,1,1,0);

cvSetReal2D(kalman_W,2,0,1);
cvSetReal2D(kalman_W,2,1,0);
cvSetReal2D(kalman_W,3,0,0);
cvSetReal2D(kalman_W,3,1,1);

return0;
}
```

```
int GLSoccerView::Get_H()
{
kalman_H=cvCreateMat(2,4,CV_32F);
cvSetReal2D(kalman_H,0,0,1);
cvSetReal2D(kalman_H,0,1,0);
cvSetReal2D(kalman_H,0,2,0);
cvSetReal2D(kalman_H,0,3,0);

cvSetReal2D(kalman_H,1,0,0);
cvSetReal2D(kalman_H,1,1,1);
cvSetReal2D(kalman_H,1,2,0);
cvSetReal2D(kalman_H,1,3,0);

return0;
}
```

```
int GLSoccerView::Get_V()
{
kalman_V=cvCreateMat(2,2,CV_32F);
cvSetReal2D(kalman_V,0,0,1);
cvSetReal2D(kalman_V,0,1,0);
cvSetReal2D(kalman_V,1,0,0);
cvSetReal2D(kalman_V,1,1,1);

return0;
}
```

Two functions which are used to get predicted values are also implemented and commented:

```
//f(x,u,0)
int GLSoccerView::Get_x_predicted()
{
CvMat* temp1=cvCreateMat(4,1,CV_32F);

//Update"kalman_x_last"tobe"kalman_x_updated"
kalman_x_last=cvCloneMat(kalman_x_updated);
cvMatMul(kalman_A,kalman_x_last,temp1);

float ax=-1.0*(cvmGet(kalman_x_last,2,0))/delta_t;
float ay=-1.0*(cvmGet(kalman_x_last,3,0))/delta_t;

float
float_temp2[]={0.5*ax*delta_t*delta_t,0.5*ay*delta_t*delta_t,ax*delta_t,ay*delt
a_t};

CvMat* temp2=&cvMat(4,1,CV_32F,float_temp2);

cvAdd(temp1,temp2,kalman_x_predicted);
//Output Mat(kalman_x_last);

Return 0;
}
```

```
//zk:=h(xk-,0)
int GLSoccerView::Get_z_predicted()
{
//h(xk,0)=H*xk-
cvMul(kalman_H,kalman_x_predicted,kalman_z_predicted);

return 0;
}
```

11.3.2 Ball EKBF Implementations Step II: Iterative Callings

The following code fragment explains how to iteratively make a proper estimation by employing the Kalman Filter algorithm:

```
for(inti=0;i<numBalls;i++){

sslBall=detection.balls(i);

//Get ball'slocation
float raw_x=sslBall.x();
float raw_y=sslBall.y();

int loop=0;
float filtered_x;
float filtered_y;

boolflag=true;

float raw_z1[]={0.0,0.0,0.0,0.0};
float raw_z2[]={0.0,0.0};

//Adjust the looping times based on field testing
while(loop<10)
{
//1)Predict
Get_x_predicted();

//Update "kalman_P_last" to be "kalman_P_updated"
kalman_P_last=cvCloneMat(kalman_P_updated);
Get_P_predicted();
```

```
//2)Get measurement raw data
raw_x=sslBall.x();
raw_y=sslBall.y();

//Apply false positive rejection
flag=check_false_positive(raw_x,raw_y);
qDebug()<<"checkfalse_positive:"<<flag;
#if1
//timeline<3 is the initiating time
if(flag||timeline<1)
{
raw_z1[0]=raw_x;
raw_z1[1]=raw_y;
raw_z1[2]=cvmGet(kalman_x_last,2,0);
raw_z1[3]=cvmGet(kalman_x_last,3,0);

raw_z2[0]=raw_x;
raw_z2[1]=raw_y;

cvSetReal2D(temp_mat1,0,0,raw_z1[0]);
cvSetReal2D(temp_mat1,1,0,raw_z1[1]);
cvSetReal2D(temp_mat1,2,0,raw_z1[2]);
cvSetReal2D(temp_mat1,3,0,raw_z1[3]);

cvSetReal2D(temp_mat2,0,0,raw_z2[0]);
cvSetReal2D(temp_mat2,1,0,raw_z2[1]);

kalman_x_last=cvCloneMat(temp_mat1);
Set_z(temp_mat2);

//Get a copy of old valid data in case of false positive
last_raw_x=raw_x;
last_raw_y=raw_y;
last_raw_vx=cvmGet(kalman_x_last,2,0);
last_raw_vy=cvmGet(kalman_x_last,3,0);

}
else
{

raw_z1[0]=last_raw_x;
raw_z1[1]=last_raw_y;
raw_z1[2]=cvmGet(kalman_x_last,2,0);
raw_z1[3]=cvmGet(kalman_x_last,3,0);

raw_z2[0]=last_raw_x;
raw_z2[1]=last_raw_y;

cvSetReal2D(temp_mat1,0,0,raw_z1[0]);
cvSetReal2D(temp_mat1,1,0,raw_z1[1]);
```

```
cvSetReal2D(temp_mat1,2,0,raw_z1[2]);
cvSetReal2D(temp_mat1,3,0,raw_z1[3]);

cvSetReal2D(temp_mat2,0,0,raw_z2[0]);
cvSetReal2D(temp_mat2,1,0,raw_z2[1]);

kalman_x_last=cvCloneMat(temp_mat1);
Set_z(temp_mat2);

}

#endif

//3)Update
Get_S();
Get_K();

//Set "kalman_z_predicted" to be "H*X_predicted"
cvMatMul(kalman_H,kalman_x_predicted,kalman_z_predicted);
Get_x_updated();
Get_P_updated();

filtered_x=cvmGet(kalman_x_updated,0,0);
filtered_y=cvmGet(kalman_x_updated,1,0);

loop++;

}

#if1
qDebug()<<"raw_x="<<raw_x;
qDebug()<<"last_raw_x="<<last_raw_x;
qDebug()<<"Filteredx="<<filtered_x;
qDebug();
#endif

//Set filtered result back to the ball object
ball.set(filtered_x,filtered_y);

balls[cam].append(ball);
}
```

11.3.3 Ball EKBF Implementations Step III: False Positive Rejection

The following code fragment gives the implementation of checking whether false positives exist.

```
bool GLSoccerView::check_false_positive(float raw1,float raw2)
{
Bool retVal=false;

CvMat* dummy_z=cvCreateMat(2,1,CV_32F);
cvSetReal2D(dummy_z,0,0,raw1);
cvSetReal2D(dummy_z,1,0,raw2);

CvMat* kalman_C=cvCreateMat(2,2,CV_32F);
CvMat* kalman_Ht=cvCreateMat(kalman_H->cols,kalman_H->rows,CV_32F);
CvMat* kalman_H_x_P=cvCreateMat(kalman_H->rows,kalman_P_predicted-
>cols,CV_32F);
CvMat* kalman_H_x_P_x_Ht=cvCreateMat(kalman_H->rows,kalman_H->rows,CV_32F);

cvTranspose(kalman_H,kalman_Ht);
cvMatMul(kalman_H,kalman_P_predicted,kalman_H_x_P);
cvMatMul(kalman_H_x_P,kalman_Ht,kalman_H_x_P_x_Ht);
cvAdd(kalman_H_x_P_x_Ht,kalman_R,kalman_C);
double Kalman_C_det=cvDet(kalman_C);

int n=kalman_x_dimension;
double denominator=pow((2*3.1415926*Kalman_C_det),(n/2));

CvMat* kalman_H_x_x=cvCreateMat(2,1,CV_32F);
cvMatMul(kalman_H,kalman_x_updated,kalman_H_x_x);
CvMat* kalman_z_minus_H_x_x=cvCreateMat(2,1,CV_32F);
cvSub(dummy_z,kalman_H_x_x,kalman_z_minus_H_x_x);

CvMat* kalman_z_minus_H_x_xt=cvCreateMat(1,2,CV_32F);
cvTranspose(kalman_z_minus_H_x_x,kalman_z_minus_H_x_xt);
CvMat* kalman_Ct=cvCreateMat(2,2,CV_32F);
cvTranspose(kalman_C,kalman_Ct);

CvMat* kalman_z_minus_H_x_xt_Ct=cvCreateMat(1,2,CV_32F);
cvMatMul(kalman_z_minus_H_x_xt,kalman_Ct,kalman_z_minus_H_x_xt_Ct);

CvMat*kalman_z_minus_H_x_xt_Ct_z_minus_H_x_x=cvCreateMat(1,1,CV_32F);
cvMatMul(kalman_z_minus_H_x_xt_Ct,kalman_z_minus_H_x_x,kalman_z_minus_H_x_xt_Ct
_z_minus_H_x_x);

    double numerator=pow(2.7183,

    (-0.5*cvDet(kalman_z_minus_H_x_xt_Ct_z_minus_H_x_x)));

double probability=numerator/denominator;

qDebug()<<Probability:"<<probability;

    //Future adjustment probability range should be basedonon-field testing
    //rather than the “accept-all” policy used in this example
    if(probability>1||probability<1.0e-5)
{
retVal=false;//Bad value
}
else
{
retVal=true;//Good value
}

Return retVal;
}
```

12 BIBLIOGRAPHY

1. **Yang, Li.** *Research on Key Technologies of Path Planning for RoboCup Small Size Robot League.* Wuhan : Wuhan University of Technology, 2012.

2. **Anik, Hasan Iqbal.** *AI, Study. <URL: http://anikstech.blogspot.fi/2011/03/different-types-of-task-environments-in.html>.*

3. **SSL-Vision.** *<URL: http://code.google.com/p/ssl-vision/>.*

4. **Sertac Karaman, Emilio Frazzoli.** *Sampling-based Algorithms for Optimal Motion Planning.* s.l. : International Journal of Robotics Research, 2011.

5. **Lavalle, S.M.** *Rapidly-exploring random trees: A new tool for path planning.* Iowa State University : s.n., 1998.

6. **Python Introduction.** *<URL: http://www.pygame.org/wiki/about>.*

7. **Numpy Tutorial.** *<URL: http://www.numpy.org/>.*

8. **SciPy.** *<URL:http://www.scipy.org/>.*

9. **Qt Toturial.** *<URL: http://qt.digia.com/Product/Developer-Tools/>.*

10. **Kafura.** *Object-Oriented Programming and Software Engineering <URL: http://people.cs.vt.edu/~kafura/cs2704/oop.swe.html>.*

11. **Innovative Enterprise Solutions (IES).** *Custom Software Development <URL: http://www.ies.co.ls/?page_id=13>.*

12. **IEEE.** *JSON: The Fat-Free Alternative to XML<URL:json.org>.* 2011.

13. **QJson.** *<URL: http://qjson.sourceforge.net/docs/>.*

14. **google-gson.** *google-gson docs <URL: https://code.google.com/p/google-gson/>.*

15. **J.Romkey.** *A NONSTANDARD FOR TRANSMISSION OF IP DATAGRAMS OVER SERIAL LINES: SLIP.* 1988.

16. **Charles M. Kozierok.** *Serial Line Internet Protocol (SLIP)* . s.l. : <URL: http://www.tcpipguide.com/free/t_SerialLineInternetProtocolSLIP-2.htm>.

17. **STP Structure.** *<URL:http://repository.cmu.edu/cgi/viewcontent.cgi?article=1002&context=robotics>.*

18. **RoboCup Community.** *Laws of the RoboCup Small Size League 2012 <URL: http://small-size.informatik.uni-bremen.de/_media/rules:ssl-rules-2012.pdf>.*

19. **Stefan Zickler, Tim Laue, Oliver Birbach, Mahisorn Wongphati, Manuela Veloso.** *SSL-Vision: The Shared Vision System for the RoboCup Small Size League.* s.l. : Carnegie Mellon University, Computer Science Depart-ment.

20. **IEEE.** *Biography of Kalman. IEEE <URL: http://www.ieeeghn.org/wiki/index.php/Rudolf_E._Kalman>.*

21. **Maybeck, Peter S.** *Stochastic Models, Estimation and Control.* s.l. : Academic Press Inc, 1979. Vol. Volume 1.

22. **Brown, R.G., P.Y.C.Hwang.** *Introduction to Random Signals and Ap-plied Kalman Filtering, 2nd Edition.* s.l. : ohn Wiley & Sons, Inc, 1992.

23. **Jacobs, O.L.R.** *Introduction to Control Theory, 2nd Edition.* s.l. : Oxford Uni-versity Press.

24. **Greg Welch, Gary Bishop.** *An Introduction to the Kalman Filter.* s.l. : Uni-versity of North Carolina at Chapel Hill,Department of Computer Science, 2006.

25. **Brett Browning, Michael Bowling, Manuela Veloso.** *mprobability Fil-tering for Rejecting False Positives.* s.l. : School of Computer Science, Carnegie Mellon University.

26. **Zhaopeng, Gu.** *OpenCV Based Extended Kalman Filter Frame 1.0.0. 2011. <URL: http://mloss.org/software/view/350/>.*

27. **Feng, Bin.** *Extended Kalman Bucy Filter demo on RoboCup <URL: http://www.youtube.com/watch?v=MBO39XHcwpc>.* 2011.

28. **Bruce, James Robert.** *Real-Time Motion Planning and Safe Navigation in Dynamic Multi-Robot Environments.* s.l. : Carnegie Mellon University, School of Computer Science, 2006.

29. **Latombe, Jean-Claude.** *Robot Motion Planning.* s.l. : Stanford University. Kluwer Academic Publishers, 1991.

30. **Lau, George Tin Lam.** *Path Planning Algorithms for Autonomous Border Patrol Vehicles.* s.l. : University of Toronto, 2012.

31. **Frazzoli, Emilio.** *Principles of Autonomy and Decision Making.* s.l. : Massachusetts Institute of Technology, 2010.

32. **LaValle, Steve.** *The RRT Page. About RRTs. <URL: http://msl.cs.uiuc.edu/rrt/about.html>.*

33. **Tewari, Ashish.** *Atmospheric and Space Flight Dynamics: Modeling and Simulation with MATLAB and Simulink (Modeling and Simulation in Science, Engineering and Technology).* 2007.

34. **Jongkwang Kima, Thomas Wilhelmb.** *a Theoretical Systems Biology.* Germany : s.n.

35. **Fortunato, Santo.** s.l. : Aalto University, 2012.

36. **Choi, Jinmu.** s.l. : University of Geogia <URL:http://www.amzi.com/ExpertSystemsInProlog/01introduction.php>, 2002.

37. **Alison.** *Intro to Logic Reasoning* . s.l. : <URL:http://cinuresearch.tripod.com/ai/www-cee-hw-ac-uk/_alison/ai3notes/subsection2_5_2_1.html>.

38. **Chen Yong, Xian-He Sun, Ming Wu.** *Algorithm-system scalability of heterogeneous computing.* s.l. : Illinois Institute of Technology, 2008.

39. **Filipe Militao, etc.** s.l. : Carnegie Mellon University, 2010.

40. **Description, DECT.** <*URL:http://www.etsi.org/index.php/technologies-clusters/technologies/dect*>.