VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Chunqiu Lu

DEVELOPMENT OF STRATEGY FOR MULTI-ROBOT CONTROL IN CONFRONTATIONAL ENVIRONMENTS

Technology and Communication

2014

FOREWORD

This thesis is my undergraduate thesis at the Telecommunication and Information Technology Department, Vaasan Ammattikorkeakoulu (Vaasa University of Applied Sciences).

Firstly, I would like to express my great appreciation to my thesis supervisor, Dr. Yang Liu, for his instruction to my thesis as well as my academic research direction. During my thesis working period, he can always guide me to solve the problem in his distinctive way. Benefited from his profound knowledge and effective advice, I have learned much about scientific research approaches. Moreover, in the past three years, as an Embedded System teacher, he also taught me much professional knowledge.

In addition, I am very thankful to the staff in Vaasan Ammattikorkeakoulu, including Dr. Chao Gao, Mr. Matila Jukka, Mr. Virtanen Antti, Mr. Chavez Santiago, Mr. Ahvonen Jani, Dr. Moghadampour Ghodrat, Dr. Menani Smail and all others staff who has help me in my undergraduate study time. Because all of your help and guide, I have learned not only the theoretical knowledge, but also the philosophy of life.

Finally, I really thank my family for their support and encourage to my study and life during these years. Also, my fellow, including Li Jinpeng, Peng Haoxiang, Zhou Yang in Botnia RoboCup laboratory should be appreciated for their help to my thesis. Hope they can obtain great achievement in the future.

Lu Chunqiu

Vaasa, Finland

11/4/2014

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Degree Program of Information Technology

ABSTRACT

Author	Chunqiu Lu
Title	Development of Strategy for Multi-Robot Control In Confrontational Environments
Year	2014
Language	English
Pages	95 + 5 Appendices
Name of Supervisor	Yang Liu

This thesis introduces the development of team strategy for Botnia RoboCup Small Size League (SSL) Robot Team. It aims to optimize the strategy software and apply STP architecture for team strategy development to obtain satisfactory performance. Meanwhile, introduction to several important algorithms, such as ERRT, safe navigation and STP architecture is another target.

The main contribution of this thesis is that it optimizes the strategy software, thereby providing more stable and convenient platform for testing and research. Also, it applies STP architecture in the team strategy, for the control of both overall team and single robots behaviors, and thus obtains good performance as expected and our system can make appropriate strategy according to different circumstances. Moreover, it successfully utilizes probability theory to scientifically evaluate the actual field situation for decision making based on mathematical model.

The thesis mainly develops on Qt at Linux platform and simulates by Matlab at Windows platform. The implementation methods in this thesis are typical software engineering approaches, which is planning, developing, debugging and testing. For sake of debugging convenience, the algorithm was firstly simulated on Matlab and then implemented in the software.

It can be concluded that STP architecture is very useful for team behavior control and mathematical theory support is very important and necessary for a system.

Keywords RoboCup, Motion Planning, Safe Navigation, Simulation, STP, Probability Theory

CONTENTS

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
CPP	C Plus Plus
DECT	Digital Enhanced Cordless Telecommunications
DHCP	Dynamic Host Configuration Protocol
EKBF	Extended Kalman-Bucy Filters
ERRT	Expanded Rapidly-exploring Random Tree
ETSI	European Telecommunications Standards Institute
fps	frame per second
KD-Tree	K dimensional Tree
GUI	Graphical User Interface
IDE	Integrated Development Environment
OOP	Object Oriented Programming
PABX	Private Automatic Branch eXchange
QSS	Qt Style Sheet
RRT	Rapidly-exploring random trees
SSL	Small Size League
SSM	Skill State Machine
STP	Skills, Tactics and Plays

LIST OF FIGURES AND TABLES

LIST OF APPENDICES

1 INTRODUCTION

1.1 Purpose of Thesis

This thesis introduces the development of team strategy in detail for Botnia RoboCup Small Size League (SSL) Robot Team. The main thesis is the optimization of strategy software and the development of STP architecture for team strategy.

1.2 Overview Structure of Thesis

The whole thesis is divided into eight chapters. The first chapter introduces the background information of this thesis, RoboCup, Botnia SSL Robot Team, motivation of this thesis and terminology. The second chapter illustrates the structure of Botnia Team System, including the overall structure and the precise introduction to every main component. The third chapter presents the strategy software. The fourth chapter focuses on the development of STP architecture for multi-robot control. The fifth chapter shows the future to-do list and the sixth chapter suggests some expectation of research direction in the future. Finally, the seventh chapter summarizes the whole thesis and the eighth chapter, as appendix, lists the mentioned source code of the thesis.

1.3 Background of RoboCup

RoboCup is an international scientific organization, a tournament field designed to improve the state of the art of robotics and AI research. Since 1997, the year of establishment, RoboCup still retains its ultimate goal that is to, set up a team of humanoid robot, obeying FIFA official rules, capable of winning against the winner of World Cup by the middle of 21st century./1/ Until now, five different types of league comprise RoboCup Competition, which are Humanoid, Middle-Size, Simulation, Small-Size, and Standard-Platform separately.

As a part of RoboCup divisions, Small Size robot soccer aims to solve the problem of smart multi-robot cooperation and control in a greatly confrontational domain with a hybrid centralized/distributed system. A Small Size robot soccer match consists of two teams with six robots each and each robot has to conform to

the size specified in the official rules./2/ The following figure illustrates a RoboCup kickoff scene in a match.

Figure 1. RoboCup SSL kickoff in a match/2/

The general dataflow and structure of RoboCup SSL are shown in **Figure 2**. More precisely, a standardized vision system with two cameras tracks the robots and ball on the field and transmits the vision data to off-field computer, strategy server. The strategy server communicates with the referee box and the robots for coordination and control.

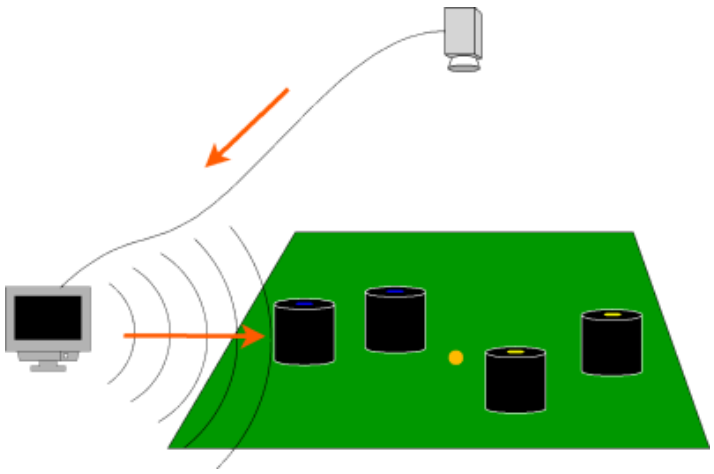

Figure 2. RoboCup SSL dataflow /2/

RoboCup SSL has a standard field shown in **Figure 3** and each participating robot must operate while complying with the official rules such us the limit of the time holding the ball, the limit of defense method. To achieve the goal of zero human input when a foul happens, the referee box is created and it can send the com-

mands from the referee directly to each team through a network. In addition, the referee box can also help tack game time, scoring etc.

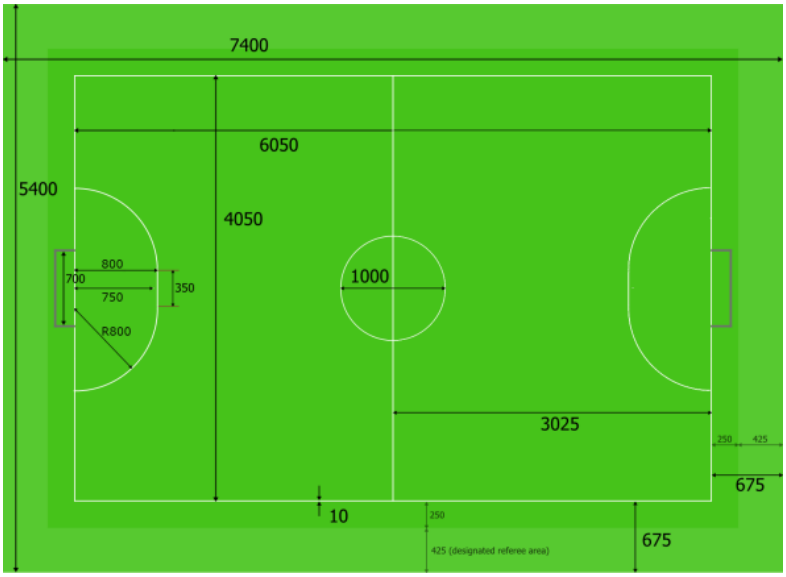

Figure 3. RoboCup SSL standard field /2/

Obviously, to build a powerful team, dedicated design and implementation of both hardware and software is required. Inevitably, RoboCup SSL is a very amazing and challenging field for education and research.

1.4 Background of Botnia SSL Robot Team

Our Botnia SSL Robot Team has participated RoboCup world competition several times and achieved high grades. Until now, we are the only qualified RoboCup SSL team among the Nordic area. /3/

Currently, we have high performance devices containing all necessary components, like the camera of the vision system, standard field, several outstanding robots and powerful strategy server.

We have second generation of Windows-based strategy software system with full function and third generation of Linux-based strategy software system in progress. The third generation enhances the instantaneity and optimizes the code. The topic in this thesis is based on the third generation and the field simulator.

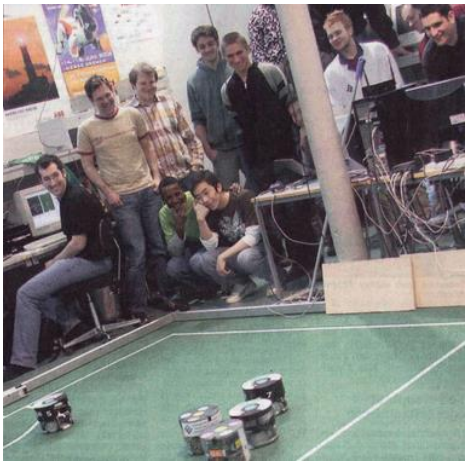

Figure 4. Full view of Botnia SSL Robot Team/3/

Figure 5. Strategy software system of Botnia SSL Robot Team

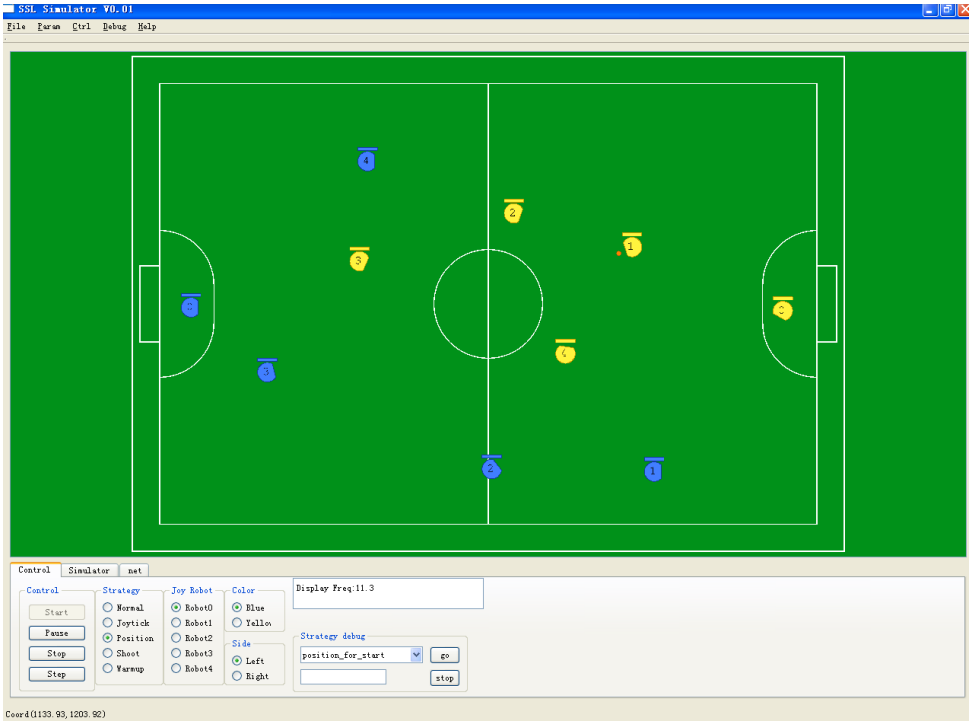

Figure 6. Field simulator of Botnia SSL Robot Team

1.5 Motivation

There is no doubt of the importance of team strategy for the autonomous multi-agent teams operating in confrontational environment. To win the game, the team must be able not only to coordinate together towards long-term goals, but also quickly to respond to sudden changes in conditions and appropriately adjust its actions according to the current situation of opponent, which at least should be as fast as the opponent./4/ Therefore, I chose this topic based on this great research platform.

In addition, my previous classmates, Feng Bin, Gao Yuan and Liu Dong, supervised by Liu Yang, have done a lot of significant work, setting valid foundation for our Botnia SSL Robot Team. Thus, it is highly necessary and adequate to continue the work on this topic and advance our team to higher level.

1.6 Terminology

The abbreviations introduced in this thesis all have corresponding full word. Also, grey highlights are used to emphasize the important source code in present context.

Finally, three types of code representations will appear in this thesis, pseudo-code, code list and API programming example./5/

In addition, as this thesis is based on the work by my previous classmates, to avoid the repetition with their work as much as possible, to the work that they finished or had described in their thesis in detail, this thesis will just make brief introduction.

2 STRUCTURE OF TEAM SYSTEM

This chapter will introduce the overall structure of our Botnia SSL Robot Team system and also the brief illustration of each component at different extent. The next chapter will focus on the strategy module.

2.1 Overall Structure

The whole system consists of several independent but interrelated modules in a closed loop. **Figure 7** illustrates the overall structure of Botnia SSL Robot Team. Obviously, the whole system can be divided into five parts, Vision Module, Strategy Module, Referee Module, Wireless Telecommunication Module and Multi-agent Module.

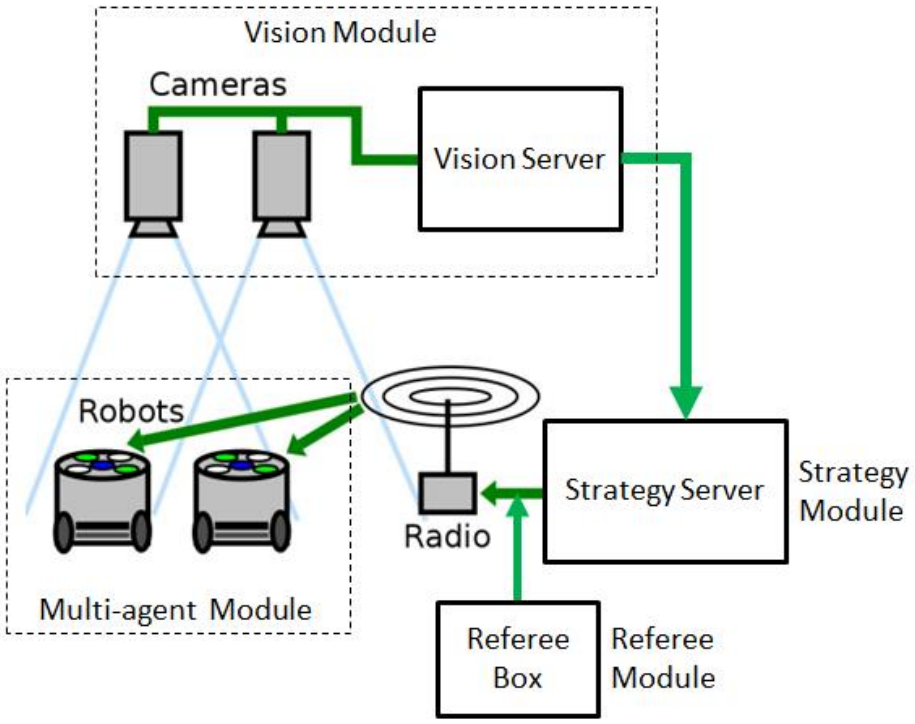

Figure 7. Overall Structure of Botnia SSL Robot Team /6/

The following will briefly introduce the each part of the whole system.

- Vision Module

Consisting of two cameras and one vision server, this module collects processes and transmits the raw data of game field to the strategy module, including the position and velocity of robots and the ball. In 2010, RoboCup responsible committees made a decision that all teams must use the standardized vision module, a shared vision system (including the hardware) /2/. That is also what our Botnia SSL Robot Team uses, called SSL-Vision.

- Strategy Module

Based on the field data from the Vision Module, the Strategy Module can evaluate current situation, generate proper strategy and further commands for each of our own side robots and finally transmit these commands to the Telecommunication Module.

- Telecommunication Module

This module is responsible for the communication between the Strategy Module and the Multi-agent Module. It consists of the transceivers on the Strategy Module side and the Multi-agent Module side.

- Referee Module

As mentioned above, when a foul happens during the match, the Referee Module can send the commands from the referee directly to each team through a network and can also help tack game time, scoring etc. Like the Vision Module, this module is also standardized.

- Multi-agent Module

Obviously, as the final performers of the whole system, this module aims to accurately and effectively execute the commands from the Strategy Module. Every the agent should automatically adjust itself according to its own situation. Also, agents should also send their own information like the quality of electricity of battery, operating conditions back to the Strategy Module through the Telecommunication Module. This function is currently under development.

2.2 Vision Module

As mentioned above, the two standardized cameras, controlled by the Vision Server, are responsible for collecting raw data from the game field. The connection between the server and cameras is FireWire cable with IEEE 1394 interface, whose transmission speed can be 50-400 MB/s for high-speed and real-time data transfer./7/ **Figure 8** shows the High Speed Camera.

Figure 8. High Speed Camera

The Vision Server (SSL-Vision), running on Linux PC, processes and transmits the raw data from the camera to Strategy Module. The sampling rate is 61.50-61.70 fps. **Figure 9** illustrates the UI of Vision Server, with the parameter configuration tree on the left, respective live image of the cameras on the center and color threshold on the right./8/

The user must adjust the parameters on the UI according to current situation thereby transmitting optimizing data to the Strategy Module. Inevitably, influenced by ambient noise, the data after processing cannot accurately represent the actual condition, therefore, the Strategy Module must further optimize the vision data.

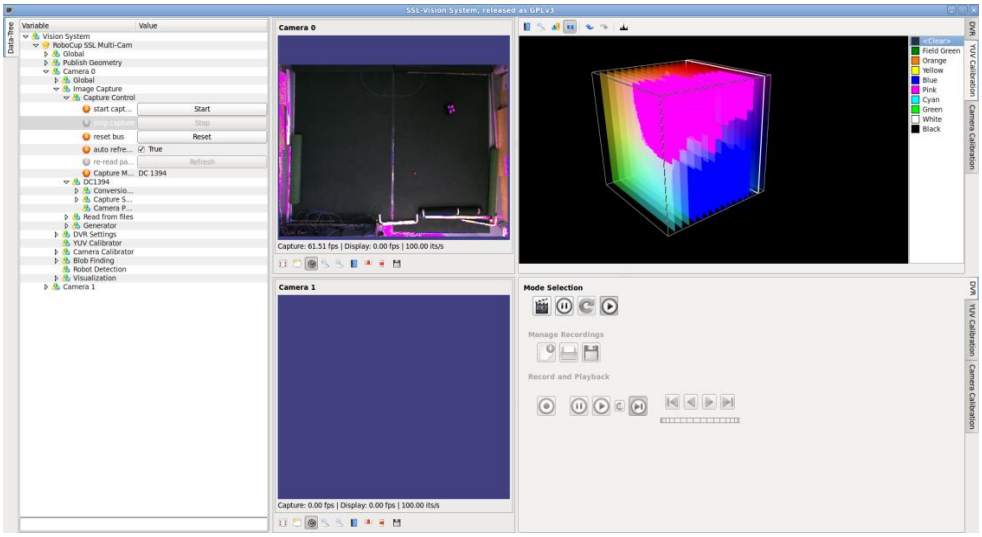

Figure 9. UI of Vision Server

2.3 Telecommunication Module

The Telecommunication Module applies DECT (Digital Enhanced Cordless Telecommunications) and Transparent/DHCP protocol to implement the data transmission from the Strategy Module side to the Multi-agent Module side. More precisely, in our Botnia SSL Robot Team, the DECT protocol connects the layer below Application Layer and Transparent/DHCP protocol connects the Application Layer.

DECT is a ETSI short-range wireless communication standard, which dominates both the wireless residential market and the enterprise PABX market./9/

The transparent protocol loads all the commands for each robot and after receiving the Transparent protocol frames, each robot selects their own particular commands while filtering others redundant information. Because of the outstanding coding scheme of DHCP, originally used for network device configuration, this protocol is also implemented in our system by Gao Yuan./5/ **Figure 11** shows the general communication structure from the Strategy Module side to the Multi-agent Module side.

Figure 10. Transceiver on the Strategy Module side and the Multi-agent Module side respectively/5/

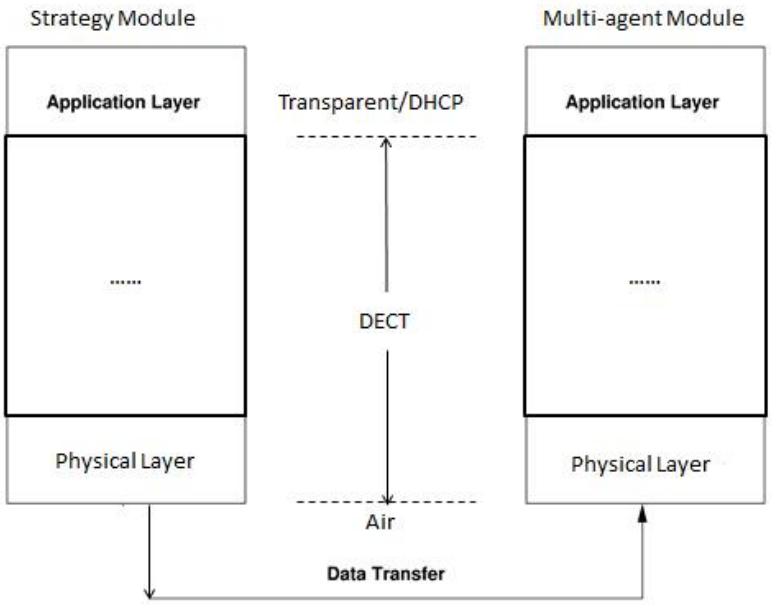

Figure 11. General communication structure from the Strategy Module side to the Multi-agent Module side /5/

2.4 Multi-agent Module

This module consists of several independent robots in the game field and also the embedded software inside. To achieve the high performance in a game, each of our robots has four omni-wheels and one dribbler as well as the transceiver mentioned above. The whole control process is that, firstly, the transceiver side receives the command signal from the Strategy Module, and then selects its dedicated command and starts to execute. For the dribbling command, the embedded

software activates the rotation of dribbler. For the kicking command, the sensor at the front side is activated so that once it detects the ball, the kicking ejector will be triggered to kick in specific power. For the position command, the target velocity will be resolved thereby arranging to each wheel motor separately. To accurately and effectively execute the position commands from the Strategy Module, the robot has a prominent control mechanism, PID control. With this approach, the actual velocity, variation tendency and target velocity all decide the control to the motor. **Figure 13** clarifies the working process of the robot in Botnia SSL Robot Team.

Figure 12. One robot of Multi-agent Module/10/

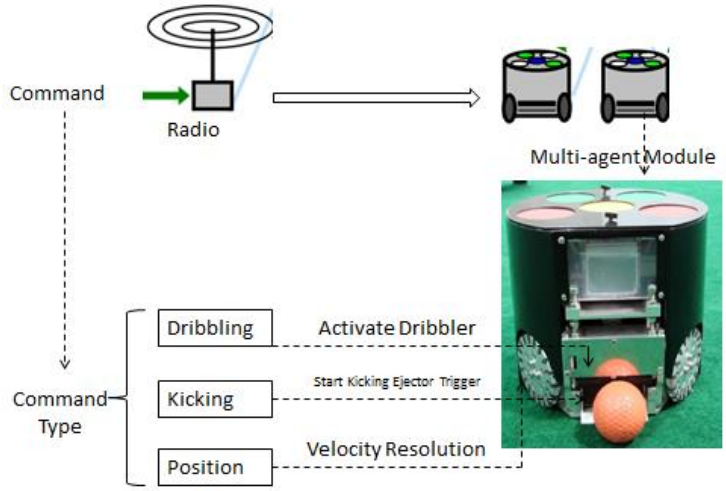

Figure 13. Working process of the robot

3 STRATEGY SOFTWARE

This chapter will introduce the strategy software, including the general introduction, IDE, architecture and overall software structure and after that it focuses on two modules.

3.1.1 Strategy Software Introduction

The strategy software of the Strategy Module is the core component of our system intelligence and implements the high automation of our system.

Influenced by the RoboCup team of CMU and University of Cornell, our latest strategy software develops and runs on the Linux platform based on the second generation of strategy software. Feng Bin and Gao Yuan did the foundation work of this software and the reason that they chose the Linux platform is that it can provide accurate time functions which is necessary for our system which is sensitive to communication environment. More precisely, the Vision Server transmits vision frame every certain short period (61.50-61.70 fps), thus in such case, to respond in real-time, our strategy software must at least make decision within the frame interval time (1/61.70 – 1/61.50 second)./5/

3.1.2 IDE Introduction

Our strategy software system is developed on Qt IDE (Integrated Development Environment). Qt is a cross-platform application and UI development framework for C++ developers which supports the major desktop platforms such as Windows, Linux and some of the mobile platforms such as Windows Phone, Android, as well as embedded platform. In addition, Qt even has the features for most of the common application, including SQL database access, XML parsing, thread management, network support, file handling and so on./16/

The signal-slot construct is its one of the important feature, which enables the communication between objects. For example, in case the GUI widget is triggered, it can send signals containing event information and receive by other objects with special functions known as slots.

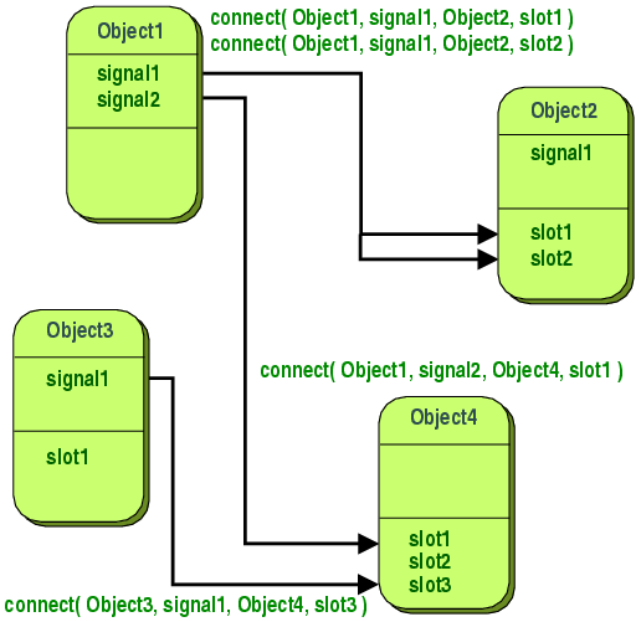

Figure 14. Qt Signals and Slots Mechanism /16/

Another important tool is Qt Designer which can design and build GUI by Qt widgets. It can also integrate programming code to signals and slots mechanism.

3.2 Overall Structure

3.2.1 Overall Architecture

Cyclically, our strategy software completes the whole process from receiving a vision frame for the analysis and finally decision-making. **Figure 15** shows the whole architecture of our strategy software.

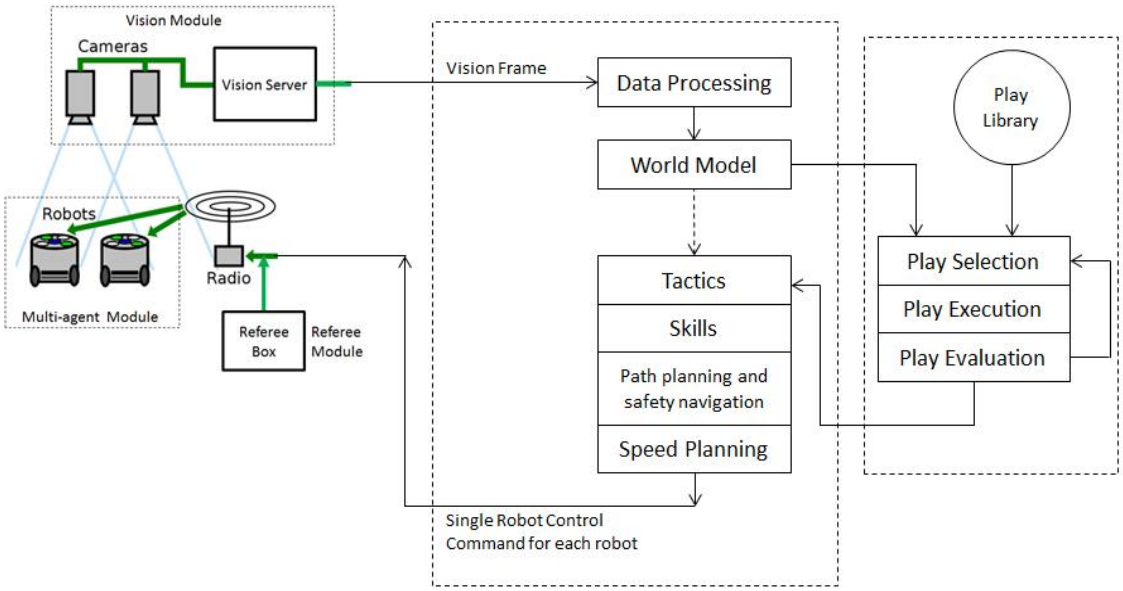

Figure 15. Overview of strategy software of Botnia SSL Robot Team/4/

- Data Processing

The image processing algorithm in the Vision Server cannot effectively process the vision information which results in the relatively high instability. Therefore, it is essential to further process the vision frame and Extended Kalman-Bucy Filters (EKBF) algorithm is applied in our Strategy Software. EKBF can not only decrease the influence of ambient noise and intermittency but can also overcome latency by predicting the parameters (velocity, orientation) of the objects in the field. /11/ Details of algorithms can be found in /5/ and /11/.

- World Modelling

This step utilizes all the available information of the game field world from the previous state to establish a game field world model, reflecting the current situation. All the evaluation and analysis for the game is based on this model. More precisely, the model contains:

1. The fundamental parameters of the objects in the field, like ball velocity, robot orientation, etc
2. Game state information from the referee box, such as game start, game stop and so on

3. Opponent modelling information based on its behavior
4. Advanced predicates based on the above information, e.g. the whether our team is in defense, whether the ball is in our possession, etc./4/

- STP Architecture

Taking advantage of the world model, STP (Skills, Tactics and Plays) architecture provides a competent mechanism for strategy software to make strategy decision from overall tactic to single robot skills. Skills are the set of independent robot fundamental activities, such as drive the ball, face to the ball as so on. Tactics are the combination of skills that can complete a certain relatively complex task, such as pass, steal the ball, etc. Plays are the combination of tactics that can complete a certain long-term task and normally, the task is to get score or defense. As the main part of this thesis, STP Architecture will be illustrated in the next chapter.

- Motion Planning and Safe Navigation

Based on the strategy decision from the previous stage, this stage generates control command for individual robot control. Motion planning consists of path planning and speed planning. The former part helps to find optimal path to the target and the latter part helps to plan the speed in the path. On the other hand, safe navigation helps to avoid collision with obstacles. **Figure 16** clarifies the control architecture of strategy software.

As seen in the figure, three possible sub-stages are involved in control planning. The Strategy Making sub-stage calculates the Final Target for each robot derived from the current state. The path planning and safe navigation sub-stage generates a near-optimal and no obstacle path and short-term target Waypoint while the speed planning sub-stage plans the speed in the short-term path.

Also, if there is no obstacle, the strategy software will directly jump to speed planning. Also, strategy software allows direct control by skipping the sub-stage of Strategy-Making. /4/ /12/ /13/ /14/ /15/

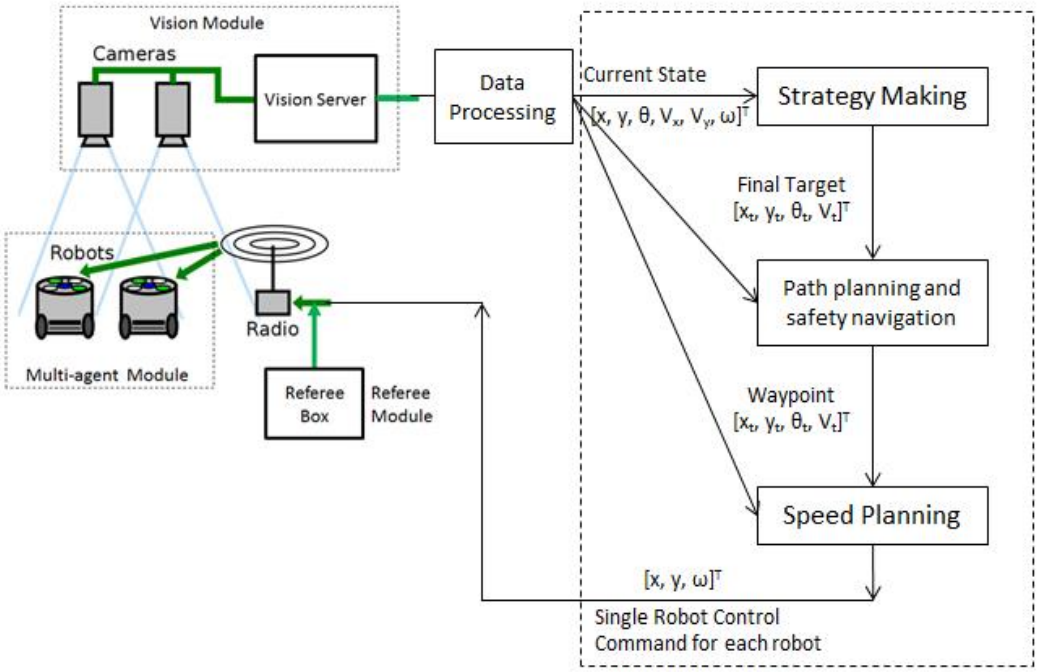

Figure 16. Control architecture of strategy software/4/

3.2.2 Overall Strategy Software Structure

Based on the architecture above, we designed the strategy software whose structure and corresponding class is shown in **Figure 17**. Compared with the overall architecture, Vision Data Processing Module implements the Data Processing stage and World Modelling stage. The Internet Module is used to clearly show the game field parameter. The Control hub Module implements the STP architecture and the Motion Planning & Safe Navigation. The Wireless Module implements the function that sends a command to the multi-agent system. Obviously, the UI module provides an interface for the user to control the whole strategy software.

The following will focus on the GUI Module and the Control Hub Module. For the detail of other module about this software, please see https://www.theseus.fi/handle/10024/59312.

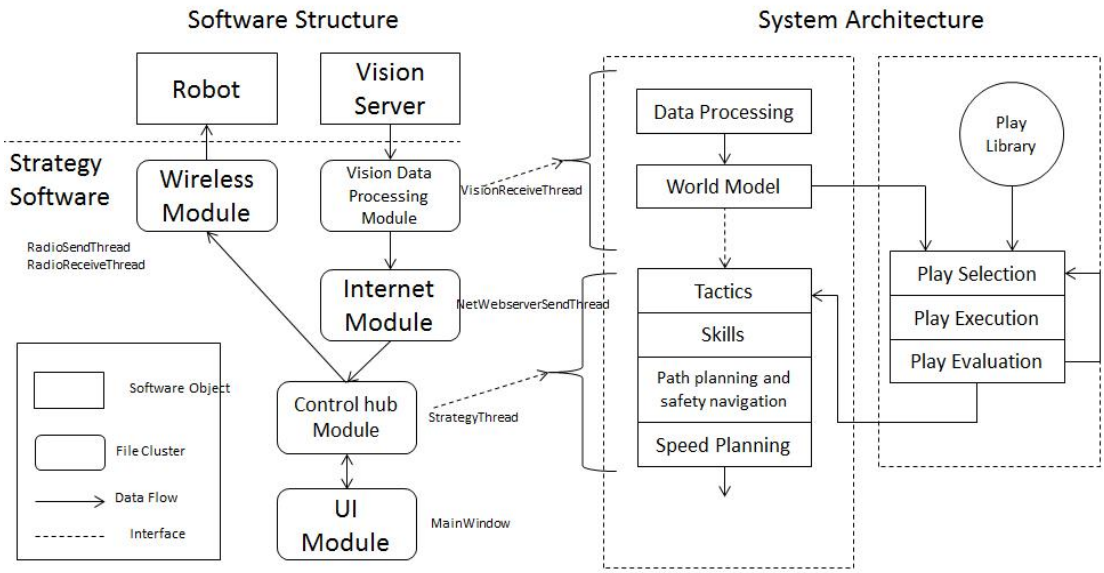

Figure 17. Overall strategy software structure /5/

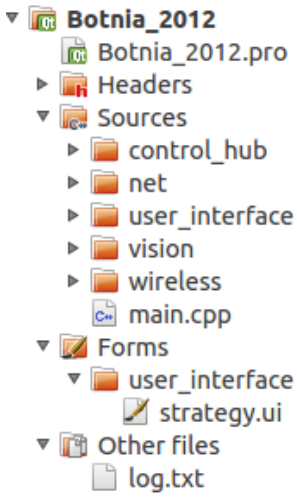

Figure 18. Overall file structure of strategy software

3.3 GUI Module

The GUI Module is a very important part of strategy software. With the GUI Module, we can monitor and control the whole system. This thesis has optimized and modified the function of the GUI Module, therefore it is necessary to illustrate the newer version here. The following part will briefly introduce the development IDE at first and then separately show the details of design and implementation.

3.3.1 Development IDE

The GUI of strategy software is designed by Qt Designer, shown in **Figure 19**. As mentioned previously, Qt Designer can design and build GUI by Qt widgets and can also integrate programming code. Area 1 is the widgets section where the user can drag necessary widgets to design a customized GUI. Area 2 is the design section where the user designs the GUI. Area 3 is the programming section where the user can program the object of GUI. Area 4 is the widget configuration section. The development of GUI also utilizes QSS (Qt Style Sheet) which is a powerful tool enabling the user to more accurately design their GUI.

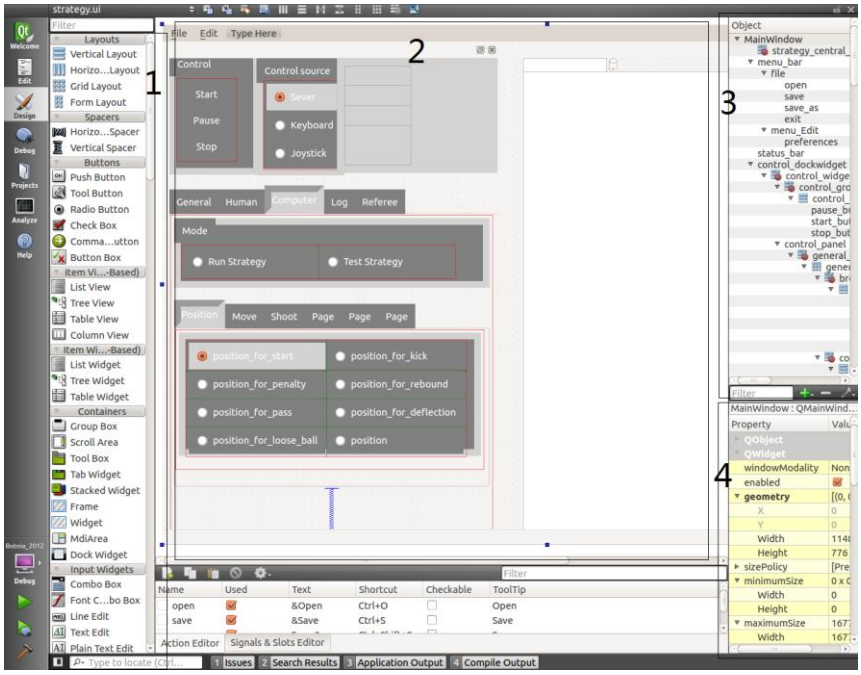

Figure 19. Screenshot of Qt Designer

3.3.2 Design

Figure 20 and **Figure 21** as well as **Figure 5** above show the GUI of our strategy software. The GUI can be divided into two main parts, the control part on the left and monitor part on the right.

- Control Part

For the control part, Area 1 is the master control of the whole strategy software. As our system can be controlled by the automatic intelligence system, or keyboard and joystick for test, Area 2 allows the user to choose the control source. Area 3 is the general system status section, indicating the current system execution status (run, pause or stop), current debug tactic and current control source. Area 4 can show the coordinate of the cursor in the Area 5, which simulates the current game field. Area 8, the group box section, provides more functions. In the first box, the General Box, the user can choose the controlled team and team side at Area 6 and Area 7 respectively.

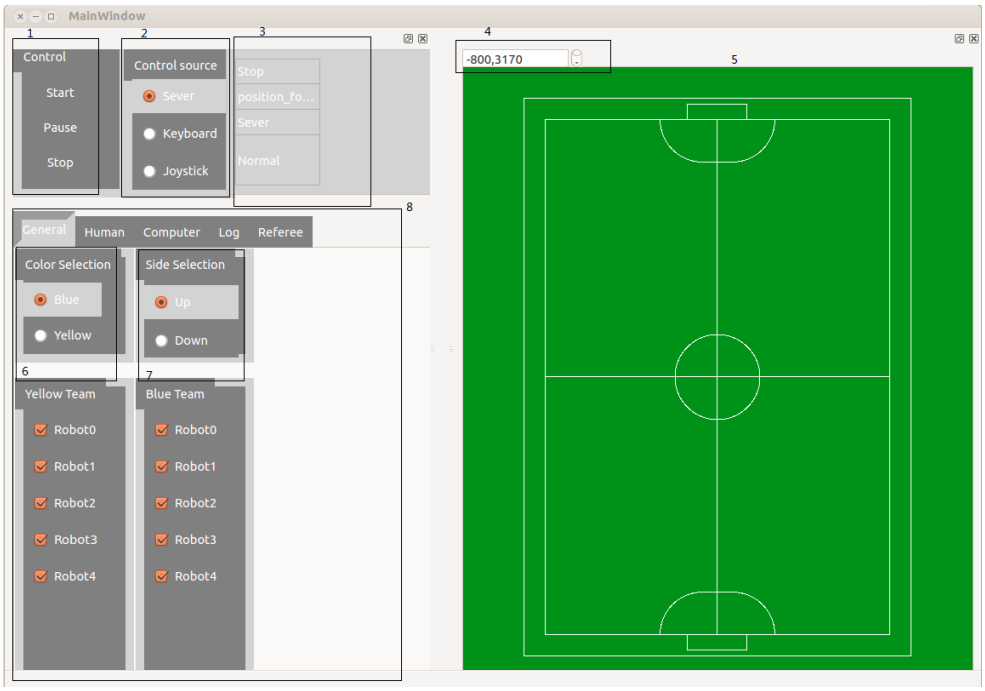

Figure 20. Screenshot 1 of GUI of Strategy Software

In the third box, the Computer Box, if the control course is set as the Server, the user can choose the system in the Run Strategy mode or the Test Strategy mode. In the former mode, the system runs automatically which decides its strategy based on current situation. On the other hand, in the latter mode, the system executes the specific strategy which is chosen in Area 2.

Figure 21. Screenshot 2 of GUI of Strategy Software

- Simulation Part

Simulation part simulates the current game field in top view and proper scale based on the data from the World Model. **Figure 22** shows a screenshot of the simulation animations. Each robot attaches a rectangle bar (Mark NO. 7), indicating its vision instability, and its current tactic (Mark NO. 6). Mark NO. 1 is the moving trajectory with its direction. Mark NO. 2, the dotted line, connect the current position to the target position. Mark NO. 3, the red point, is the next waypoint to the target position. Mark NO.4, the two dotted lines, is the maximal tolerance angle if the task is kicking, while Mark NO. 5 is the best angle. All these functions are designed to help to develop strategy.

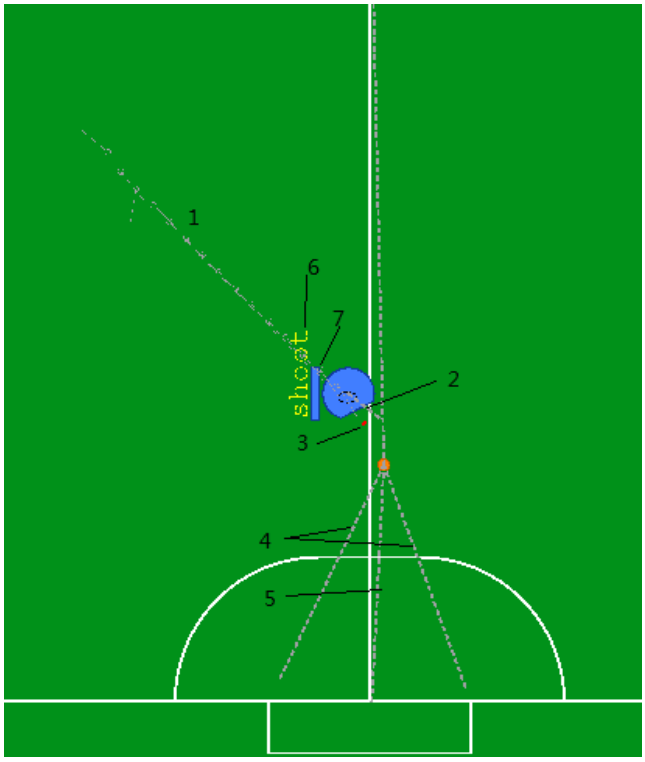

Figure 22. Screenshot 3 of GUI of Strategy Software

3.3.3 Implementation

CPP file strategy_control_window.cpp and its header file implement most of the GUI functions. The whole GUI is defined as a class MainWindow in strategy_control_window.h, shown in **Figure 23**. The Class MainWindow contains all the objects in the customized GUI and defines some corresponding functions. As mentioned above, most of the functions are implemented based on the signals and slots mechanism, shown in **Figure 24**.

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
private:
    void SetupVisionReceiveThread();
    void SetupRefboxReceiveThread();
    void SetupNetwebSendThread();

    void SetupDefaultTeam();
    void SetupStrategyThread();
    void SetupThread();
```

Figure 23. Screenshot of GUI implementation file

```
private slots:
    // method for menu file
    void apply_changes();
    void open();
    void save();
    void save_as();
    void exit();
    // methods for menu edit

void MainWindow::SetupGUIConnection()
{

    /// connect act for menus    ////////////////////////////////////////
    connect(ui->open,SIGNAL(triggered()),this,SLOT(open()));
    connect(ui->save,SIGNAL(triggered()),this,SLOT(save()));
    connect(ui->save_as,SIGNAL(triggered()),this,SLOT(save_as()));
    connect(ui->exit,SIGNAL(triggered()),this,SLOT(close()));
    connect(ui->preferences,SIGNAL(triggered()),this,SLOT(preferences()));
```

Figure 24. The application of Signals and Slots mechanism

Moreover, as the GUI is the only way to control the strategy software, the Class MainWindow also has some functions that set up the threads of other modules mentioned in **Figure 17**. **Figure 25** shows the code example.

```
void MainWindow::SetupStrategyThread()
{
     //SetupDefaultTeam();
    // we do not need the vision recieve thread as the strategy thread already has it radio sender Open Successed
    strategy_thread_ = new StrategyThread(ui->strategy_graphics_view,true);
    // this thread does not include the vision receive thread and the reference box thread, but it can send the ssl package
    strategy_thread_->start(QThread::TimeCriticalPriority);
}
```

Figure 25. Set up of other modules in Class MainWindow

3.4 Control Hub Module

The Control Hub Module is the main Module of strategy software which implements the control mechanism for the multi-agents system. The following will firstly introduce the overall design architecture and then introduces the implementation software structure according to the design.

3.4.1 Overall Architecture

The Control Hub Module aims to appropriately control the whole multi-agent system, thereby beating the opponents, according to the current situations. **Figure 26** shows the overall architecture of this module. The key point of this architecture is to scientifically subdivide the whole process to relative independent sub-stage.

Firstly, once the system starts, the Control Hub Module decides the control source, Joystick or Computer, according to the set at GUI Module. If the Computer is set as the control source, the system then decides how to execute the STP architecture, according to the strategy mode, TEST or RUN. The difference is that in the TEST mode, the Control Hub Module will directly execute the special tactic which is set in the GUI Module. On the other hand, if the RUN mode is set, the system will choose the proper Play after the analysis and evaluation and then execute the tactics of that Play.

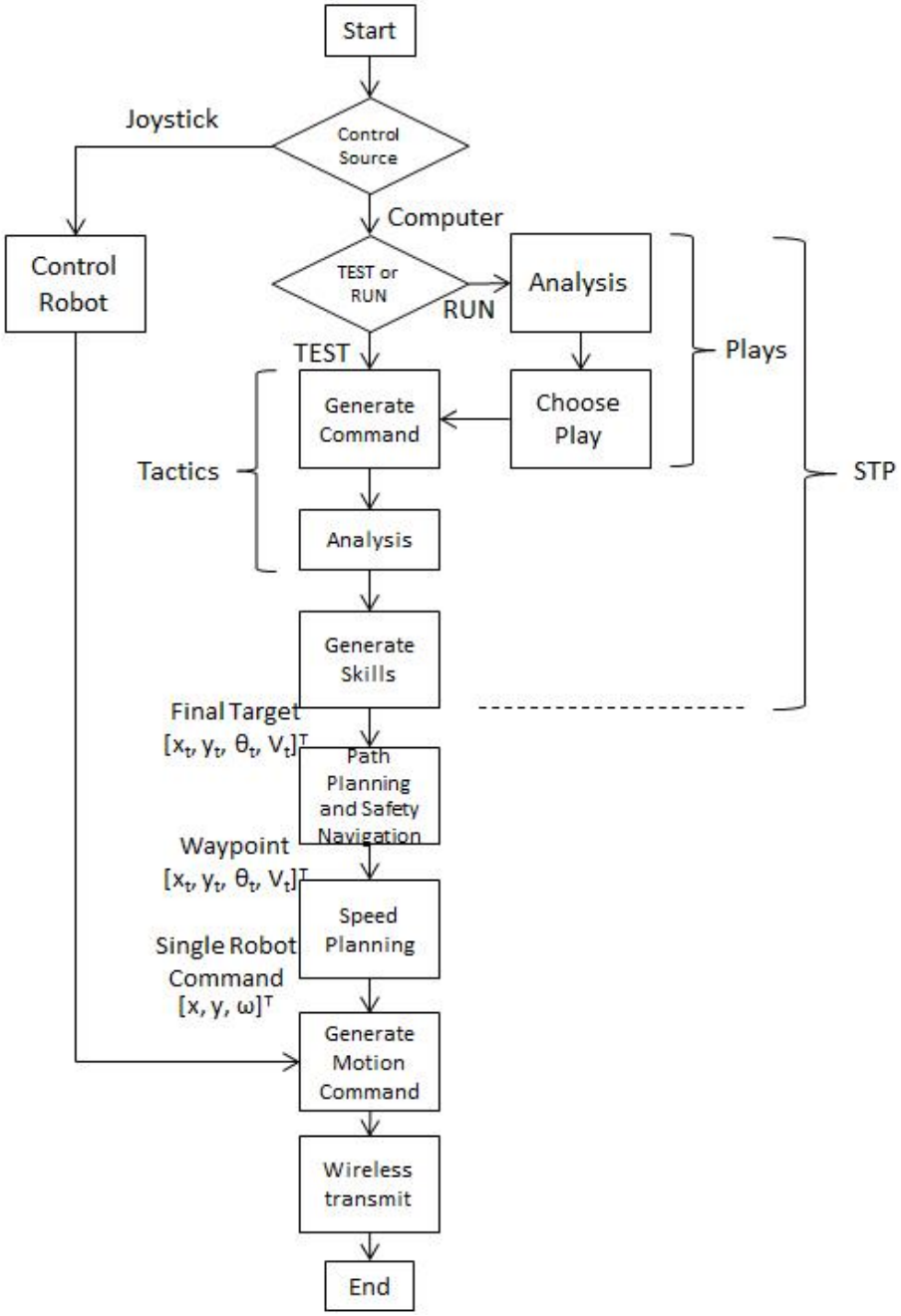

Figure 26. Overall Architecture of Control Hub Module

After the Tactics and Skills execution, the Final Target is generated,

$$f_{STP}=[x_t, y_t, \theta_t, V_t]^T \quad (1)$$

xt, yt is the horizontal and vertical axis of the target point

θt is the direction of the robot at the target point

Vt is the Velocity of robot at the target point, Vx, Vy, ω

After that, the path planning and safe navigation sub-stage generates a near-optimal and no obstacle path and short-term target Waypoint,

$$f([xt, yt, \theta t, Vt]^T) = [x'_t, y'_t, \theta'_t, V'_t]^T \quad (2)$$

Finally, the speed planning sub-stage plans the speed in the short-term path and generates the single robot command for each robot,

$$f([x'_t, y'_t, \theta'_t, V'_t]^T) = [x, y, \omega]^T \quad (3)$$

x,y is the speed on horizontal and vertical axis of the target point
ω is the angular velocity

The reason that divides the target to Final Target and Waypoint is that in such confrontational environment, it is necessary to check the target status and approach the final target waypoint by waypoint every short period for security and flexibility reason.

Furthermore, because of the limited calculation resource of the robot body embedded system, the strategy software undertakes most of the calculation load so that the robot embedded software can focus on the accurate control given special speed parameter.

3.4.2 Software Structure

Based on the architecture above, we implement the Control Hub Module. The following part will firstly introduce the technical background and then illustrate the software structure.

3.4.2.1 Technical Background

- Inheritance

Inheritance is a very important feature in C++, OOP (Object Oriented Programming), which allows the programmer to define a class based on another class. More precisely, it means that, for example, a class can inherit the data and functions from multiple base classes. This feature makes it easier to create and main-

tain the source code. Because of this reason, our strategy software including the Control Hub Module has applied this technology.

- Virtual Method

In C++, the method in the class with the keyword *virtual* in the front is virtual method. Virtual method allows method with the same name of inheriting class to override its virtual method of the base class.

- Polymorphism

Polymorphism is another important feature in C++, based on the feature of inheritance and virtual method. Polymorphism can provide a single interface of various different types, which means that the interface can behave differently according to the contexts. In other words, whether a virtual method is overridden depends on the contexts. /17/

- QThread

The QThread class belongs to QT framework, providing a platform-independent way to manage threads. In QT, the programmer can inherit QThread to develop customized thread program. The QThread start to execute in *void QThread::run ()* function which should be triggered by v*oid QThread::start(Priority priority = InheritPriority)* function. As our software is a multi-thread concurrent program, we have applied this technology./18/

3.4.2.2 File Structure

Figure 27 shows the file structure of the Control Hub Module and its corresponding brief introduction. From the name of the file, it is clear that folder *computer_control* and *human_control* implement the control source. Under *computer_control* folder, *intelligence* folder is responsible for the strategy analysis and execute, while *knowledge_base* folder is responsible for recording the current world status and providing necessary algorithms. In the *intelligence* folder, *strategy_extutor* folder implements the STP architecture and the *world_analysor* folder provides some basic algorithms and functions.

Figure 27. File Structure of Control Hub Module

3.4.2.3 Logic Structure

As shown in **Figure 26**, we implemented each sub-stage with multiple methods shown in **Figure 28**. The whole process is executed in class StrategyThead, defined in *strategy_thread.h*. Play selection in the STP architecture is implemented in class Strategy, defined in *strategy.h*, which will be illustrated in the next chapter.

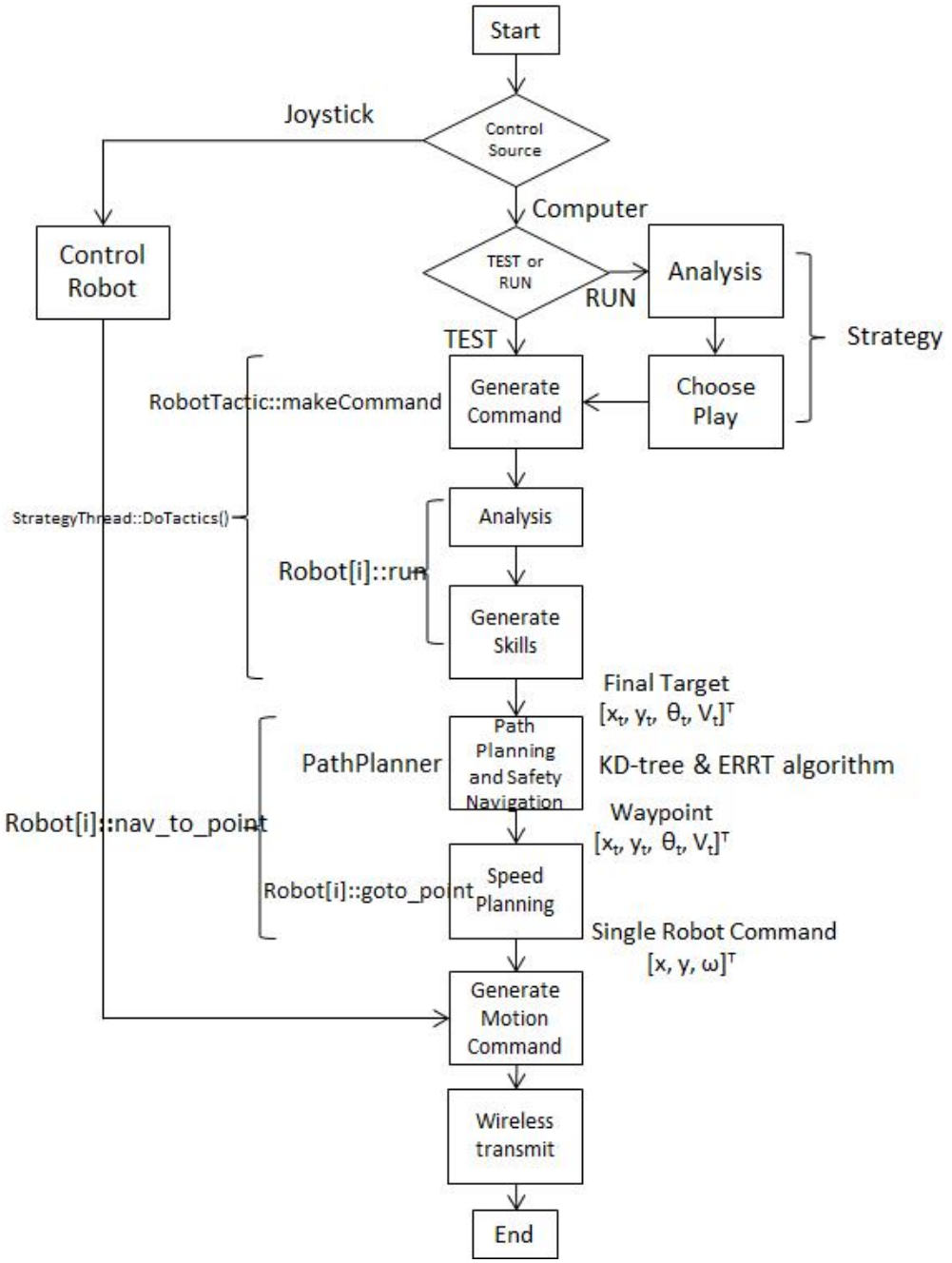

Figure 28. Overall Structure of Control Hub Module

3.4.3 Implementation of Tactic execution and polymorphism

Function *StrategyThread::DoTactics()* starts to execute the tactics given by class Strategy, shown in Figure 29. The Class Tactic is a base class which provides some fundamental method and data and all the tactic classes need to inherit it, so this is a typical example of polymorphism. As the *tactics[i]* is a Tactic pointer, it can point to the object of Tactic and its inheriting class. For example, the appen-

dix gives the definition of a tactic class TShoot which indirectly inherit the Tactic class and its inheritance relationship is shown in **Figure 30**. Once the object of TShoot is assigned to the pointer, the program in **Figure 29** will call the virtual method *void RobotTactic::run(World &world, int me)*. This exactly reflects the essence of polymorphism, which is that the interface can behave differently according to the contexts.

```
//StrategyThread::DoTactics()
        case STRATEGY_TEST:
          if (gui_tactics[i])
            {gui_tactics[i]->run(world, i);  }
            break;
        case STRATEGY_RUN:
            if (tactics[i])
            {tactics[i]->run(world, i); }
            break;
//Definition:
Tactic *tactics[MAX_TEAM_ROBOTS];
```

Figure 29. Execute the tactics in StrategyThread

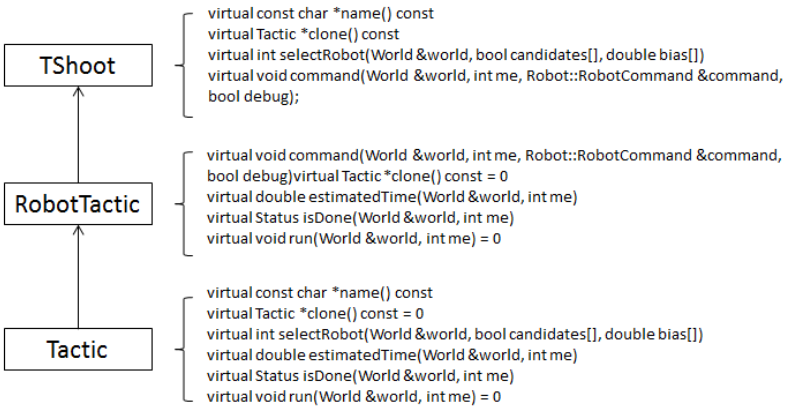

Figure 30. Tactic Class Inheritance Diagram

In the method *RobotTactic::makeCommand* of *void RobotTactic::run(World &world, int me)*, virtual method *command* generates the command for the robot, including the target robot position, the target ball position, etc. Similarly, because of polymorphism, as shown in **Figure 30**, class TShoot overrides the virtual func-

tion *command* of class RobotTactic, thus in this case, the program will call the *TShoot::command(World &world, int me, Robot::RobotCommand &command,bool debug).* This is a very powerful feature, as the tactic becomes easier to develop and maintain. When it is necessary to add new tactics, we can just inherit the Class RobotTactic and implement the corresponding interface. **Figure 31** shows a new tactic class definition example, which allows two robots to pass and receive ball.

```
class TPassandReceive: public RobotTactic
{
private:
   TPass  *PassRobot;
   TReceivePass *ReceiveRobot;
   SPosition *PositionRobot;
public:
   TPassandReceive();
   ~TPassandReceive();
   virtual const char *name() const;
   static Tactic *parser(const char *param_string);
   void LoadConfig();
   virtual Tactic *clone() const;
   virtual int selectRobot(World &world, bool candidates[], double bias[]);
   virtual void command(World &world, int me, Robot::RobotCommand
   &command,bool debug);
};
```

Figure 31. New tactic example

After generating commands, the following steps are to analyze the current situation and generate the target position with corresponding parameter, which is called skills. This part also belongs to the STP architecture and will be introduced in the next chapter in detail.

4 PATH PLANNING AND SAFE NAVIGATION

4.1 Introduction

Path Planning and Safe Navigation aims to find an optimal and safe path to target position and returns to the next waypoint. It is very important in our dynamic and complicated RoboCup environment, thus many approaches are created to solve the problem. The following figure illustrates the comparison between some notable algorithms.

Category	Name	Advantage	Disadvantage
Grid-Based	A*	No polygons	1. Memory usage grows with the size of the field 2. Cannot work in narrow passages/33/
Geometry-Based	Visibility graphs	Depends only on number of obstacles O($n^2 \log n$)/31/	1. path is too close to the obstacles 2. hard to get good polygons
	Trapezoidal Decomposition(Exact cell decomposition)	Easy to implement O(n log n) /34/	Not optimal path/35/
Potential Fields		Easy to implement/35/	Local minima problem/35/
Sampling-Based	RRT(Rapidly-exploring random trees)	efficient and suit high-dim	Not optimal path
	PRM(Probabilistic Road Map)	efficient and suit many problems/36/	

Figure 32. Comparison between different path planning algorithms/27/ /28/ /29/

Among them, one of the popular path planning algorithms is RRT (Rapidly-exploring random trees), which employ randomization to quickly search possible

path. /22/ Furthermore, ERRT (Expanded Rapidly-exploring Random Tree), as a kind of variant of RRT, enhances the efficiency and is utilized by our system in path planning.

4.2 Path Planning

4.2.1 Theory

Path planning intends to find an optimal path from current position to the target position. Our Control Hub Module applies KD-tree (K dimensional tree) and ERRT (Expanded Rapidly-exploring Random Tree) algorithm to solve this problem.

Figure 33 shows the ERRT algorithm, which aims to search for a path from an initial position or state to a goal position or state by expanding a search tree. The main function *ERRTPlan* iteratively chooses a random target state and expands the nearest part of the tree towards the target. The loop terminates when the distance from the tree to goal reaches a threshold.

Function *ChooseTarget* generates specific point state (three probabilities) for function *ERRTPlan* to expand the tree. With probability P[goal], the goal is chosen as a target. With probability P[waypoint], a random waypoint is chosen as a target and the remaining probability is for random state. Typical values for the probability is 0.1 for P[goal] and 0.6 for p[waypoint] respectively. This function is the significant difference with the basic RRT algorithm, as the latter choose target randomly.

Function *Nearest* applies KD-tree algorithm to speed the nearest neighbor lookup./14/

Function *Extend* generates a new state which can be reached from the target state in incremental distance. Function *Distance* calculate the distance between two given state. Function *RandomSate* generates a random state in the given domain. **Figure 34** gives an example about the application of ERRT algorithm. /14/

```
function ERRTPlan (env:environment,initial:state,goal:state):rrt-tree
        var nearest, extended,target:state;
        var tree:errt-tree;
        nearest := initial;
        errt-tree := initial;
        while(Distance (nearest,goal) < threshold)
            target = ChooseTarget (goal);
            nearest = Nearest (tree,target);
            extended = Extend (env, nearest, target);
            if extended ≠ EmpetyState
                then AddNode (tree, extended);
        return tree;
function ChooseTarget (goal:state): state
        var p:real;
        var i:integer:
        p = UniformRandom in [0.0 … 1.0];
        i = UniformRandom in [0 … NumWayPoints-1];
        if 0< p < GoalProb
            then return goal;
        else if GoalProb <p < GoalProb + WayPointProb then
            return WayPointCache[i];
        else if GoalProb + WayPointProb < p < 1 then
            return RandomState();
function Nearest (tree:rrt-tree, target:state):state
        var nearest:state;
        nearest := EmptyState;
        foreach state s in current-tree
                if  Distance (s,target) < Distance (nearest,target)
                    then nearest := s;
        return nearest;
function Extend (env:environment,initial:state,goal:state):state;
function Distance (current: state, target: state): real;
function RandomState (): state;
```

Figure 33. ERRT Algorithm/14/

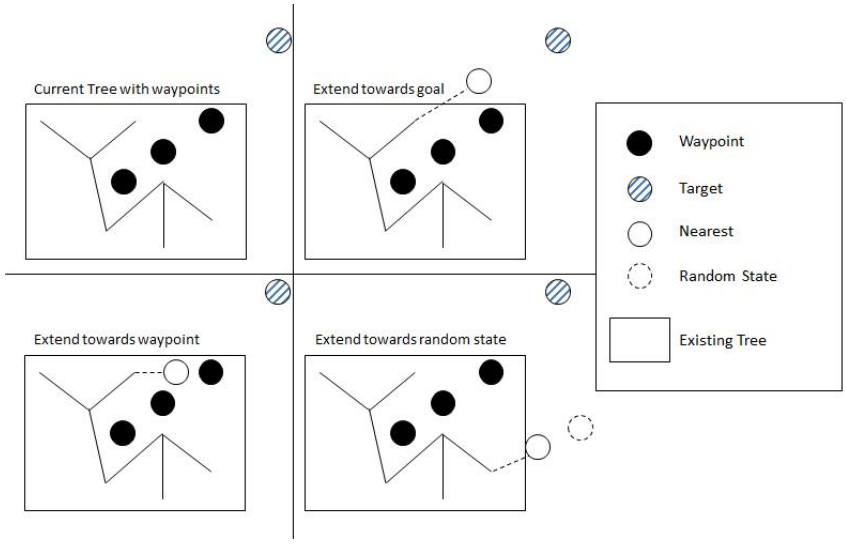

Figure 34. Illustration of ERRT /14/

There are some others variants algorithm of RRT, such as RRT* (RRT star). Similarly, the main difference is target choose method. Different from ERRT, RRT* can surely converge to an optimal solution, by the approaches of committed trajectories and branch-and-bound tree adaptation. /37/

4.2.2 Simulation

To verify the performance of this algorithm, we simulate its procedure as experiments. **Figure 35** shows one of the simulation results for ERRT algorithm. The blue point, coordinate (50, 10), is the initial state, while the red point, coordinate (750, 550) is the goal state. The simulated field is a rectangle whose width and length is 800 and 600 respectively while several obstacles scattered between the initial state and the goal state. The probability is 0.1 for P[goal], 0.6 for p[waypoint] and 0.3 for random generate respectively and the extend distance is 4 every time. After simulation, the best path plan is shown in bold.

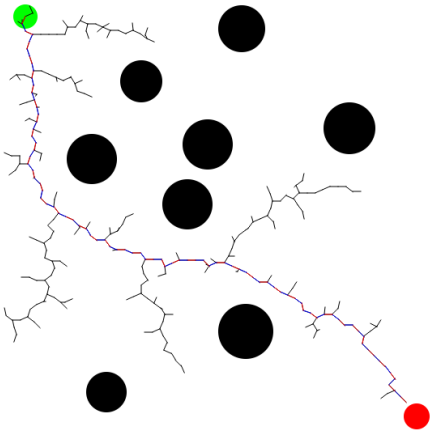

Figure 35. ERRT Algorithm Simulation/20/

Similarly, the following figure also illustrates the simulation results of ERRT and RRT*. It can be seen from the figure that the path length of RRT* is smaller, so it is a better path plan compared to ERRT.

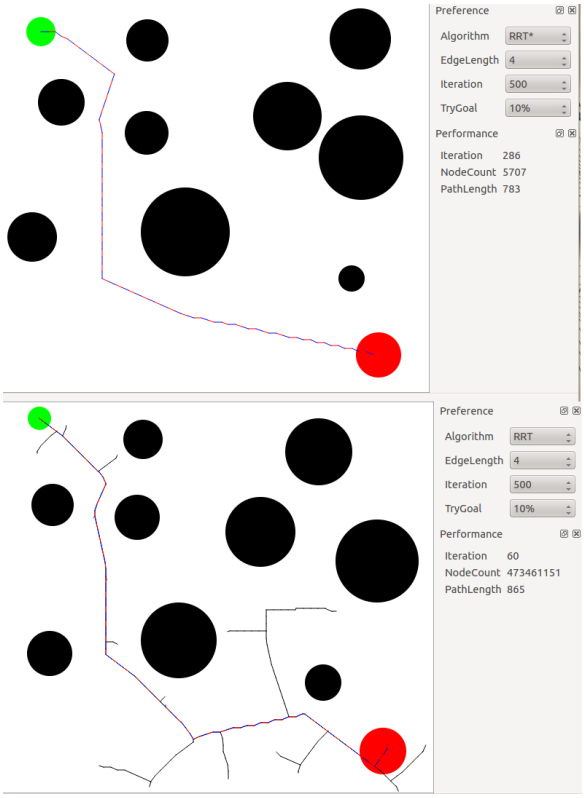

Figure 36. Simulation Result of ERRT(down) and RRT*(up)

4.3 Safe Navigation

4.3.1 Theory

Safe navigation is applied to avoid collision with obstacles thereby guaranteeing the safety of robots. *Field_world_obstacle.h* defines class *obstacle* to describe the obstacles in the field, shown in Appendix. Clearly, **Figure 37**, for the sake of simplicity, defines three types of obstacles with three macros respectively to represents all the obstacles, OBS_RECTANGLE represents the working area in the field according to the rules, such as defense field, OBS_CIRCLE represents the robots and balls in the field while OBS_HALF_PANE represents the field border. Correspondingly, class *obstacle* has a private integer *type* to save the obstacle type. Also, it has other data to save, for example, the location and speed, and method to check collision. The file also defines another class *obstacles* (see Appendix) to represent all the obstacles in the field for a given robot as well as some methods to detect collision.

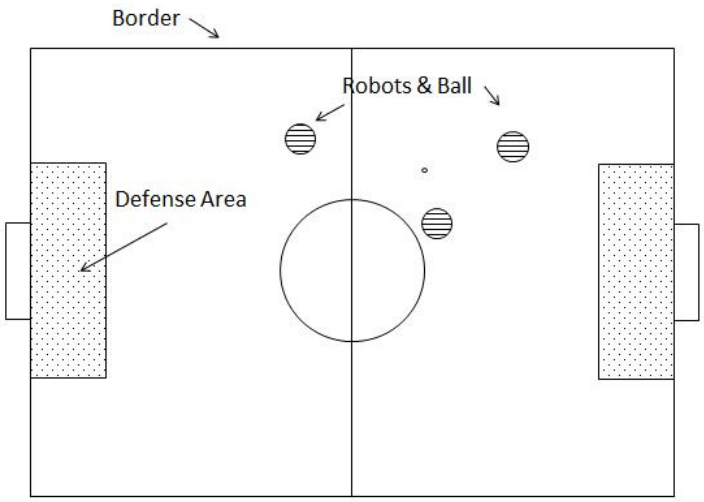

Figure 37. Field Obstacles

For the method to check collision, **Figure 38** illustrates different situations. Mark NO.2, NO.3, NO.4 respectively represent the robot collision check with OBS_CIRCLE, OBS_HALF_PANE and OBS_RECTANGLE type of obstacles. In addition, according to the ERRT algorithm, the program should check collision at every tree extension step (the reason will be explained in the next section), so it

concerns another program, which is the collision check for the robot moving on a specific segment. Mark NO.1, NO.3, and NO.5 respectively represents the collision check with three types of obstacles.

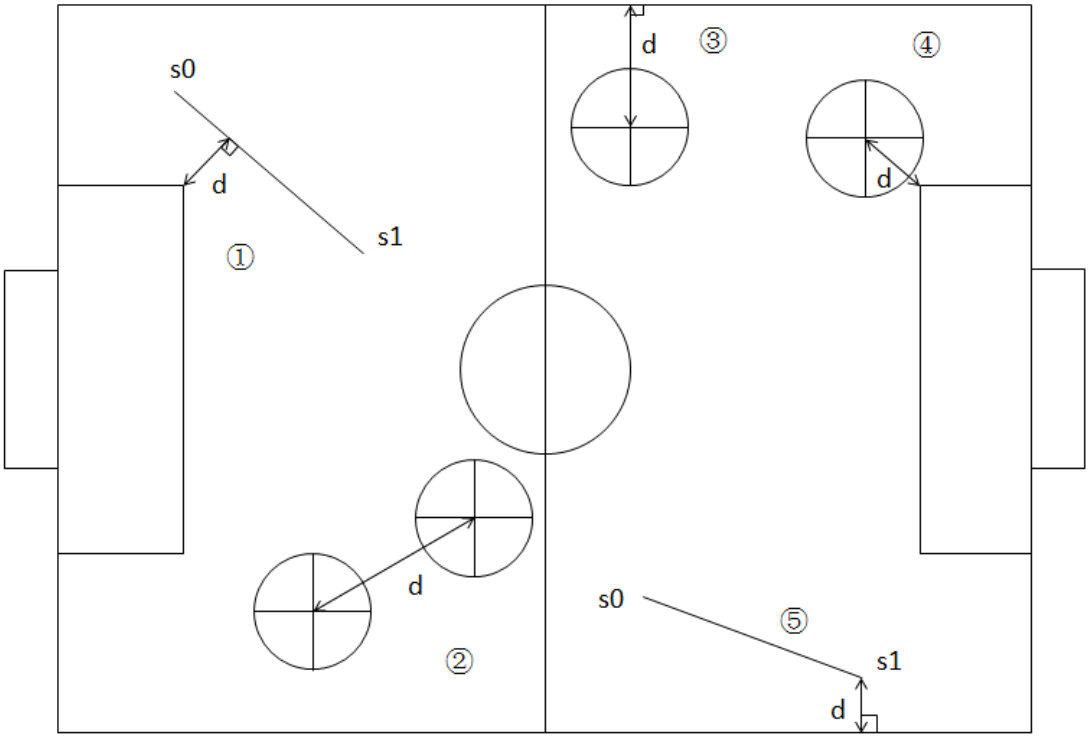

Figure 38. Illustration of Collision Check

4.3.2 Example

The key point of collision check is to calculate the shortest distance, d, between two objects. Take Mark NO.5 as example. This is the collision check for the robot moving on a specific segment and a half plane, which usually is to check whether the robot will be out of border. **Figure 39** shows the collision check calculation. Obviously, the shortest distance is either the perpendicular line to plane *a* through two endpoints,

$$b=(\overrightarrow{OS} - \overrightarrow{OC}) \cdot \overrightarrow{p} \quad (4)$$

$$c=(\overrightarrow{OP} - \overrightarrow{OC}) \cdot \overrightarrow{p} \quad (5)$$

where

$\overrightarrow{p}$ is the direction vector of plane a,

$\overrightarrow{OC}$ is the center position vector

Therefore, if both b and c are longer than the radius of the robot, there is no collision.

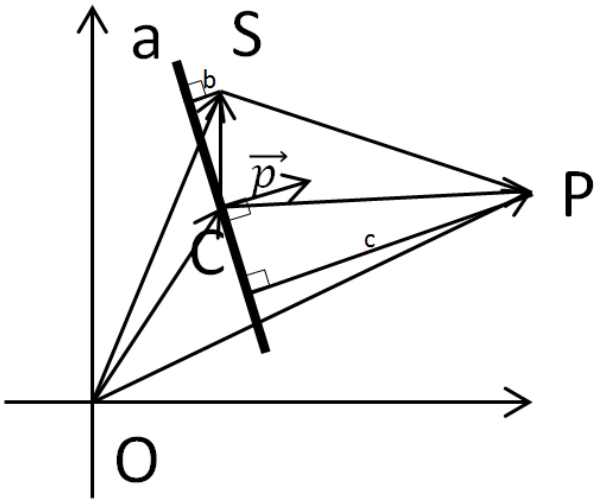

Figure 39. Collision Check Example

4.4 Combination of Path Planning and Safe Navigation

4.4.1 Theory

To reduce the time complexity, the program of this module combines the path planning and safe navigation together, thus safe staff is simultaneously considered when planning the path. More precisely, to avoid a collision with obstacles, before extending the tree to a generated new state in function *extend*, the program will apply the method in the previous section to check whether there is a collision. **Figure 40** clarifies the mechanism. It should be noted that, when there is collision, the program will firstly try to avoid the collision by rotating this step vector. If the collision still exists, the tree will not extend to the generated new state.

```
function Extend (env:environment,initial:state,goal:state):state
        var extended:state;
        var step:vector;
        var d:double;
    step = goal.pos – initial.pos;
        d = step.length();
        if d < step_size
                then return 0;
        step *= step_size/d;
        extended = initial.pos + step;
        if Check (extended)
                then return extended;
        else if Check(extended.rotate())
                then return extended;
        else
                then return 0;
```

Figure 40. Function extend with Safe Navigation

4.4.2 Simulation

The following figure illustrates the execution of this algorithm, implemented by https://www.theseus.fi/handle/10024/59312. The upper figure shows the path searching situation. The green circle is the initial state while the red circle is the target state. The black circle is the obstacle. For the upper figure, the yellow rectangles are the safe distance between the obstacle and the initial state to avoid a collision.

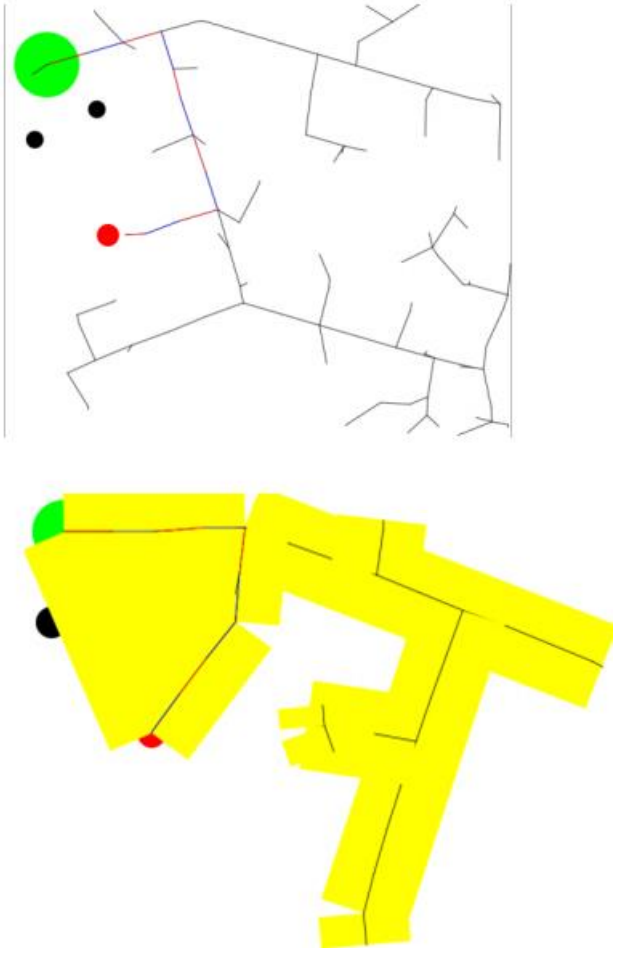

Figure 41. Safe ERRT Simulation /5/

The following figure shows the situation in the real game field. Obviously, our robot successfully goes through the obstacles without any collision.

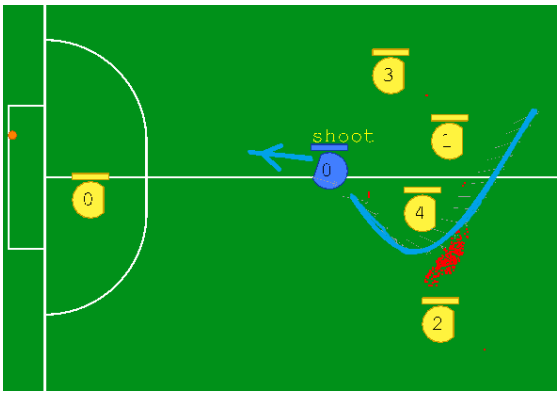

Figure 42. Real Situation in Game Field

5 STP ARCHITECTURE FOR MULTI-ROBOT CONTROL

Based on the discussion in the previous chapters, this chapter will particularly introduce the STP architecture, including the skills, tactics and plays, and how they coordinate for a multi-robot system.

5.1 Introduction

The STP architecture is designed by the researchers of Carnegie Mellon University, to implement the aims of responsive and adversarial team control./4/ The key point of this architecture is that it divides the behavior of single robot and whole team, thus making it much easier and clearer to develop and maintain the system.

- Play

In STP architecture, play, *P*, is defined as a fixed plan, consisting of applicability and termination conditions, *N* participants. Each participant has a sequence of tactics T^1, T^2 etc. and corresponding parameters. After the role assignment, each robot *i* has its tactic T_i from current step.

- Tactic

Tactic, *T*, contains behavior of a single robot. Each robot *i* executes its own set of tactics belonging to the current play *P*. A specific T_i with corresponding parameters decides its own SSM_i (skill state machine) which will be executed by the robot *i*. Tactic T_i can set parameters $SParams_i$ for its containing skills S_i.

- Skill

Skill, *S*, is a fundamental robot activity, such as kicking the ball. Each skill, S, belongs to one or more *SSM* and decides the next transiting skill S', based on current world state, continuous executing time, etc. To generate appropriate decisions, skills and tactics will evaluate the current world state.

Therefore, in such hierarchy, the team is directly controlled by plays through tactics which consists of its own SSM with sequences of skills. **Figure 43** and **Figure 44** show the STP execution respectively. /4/

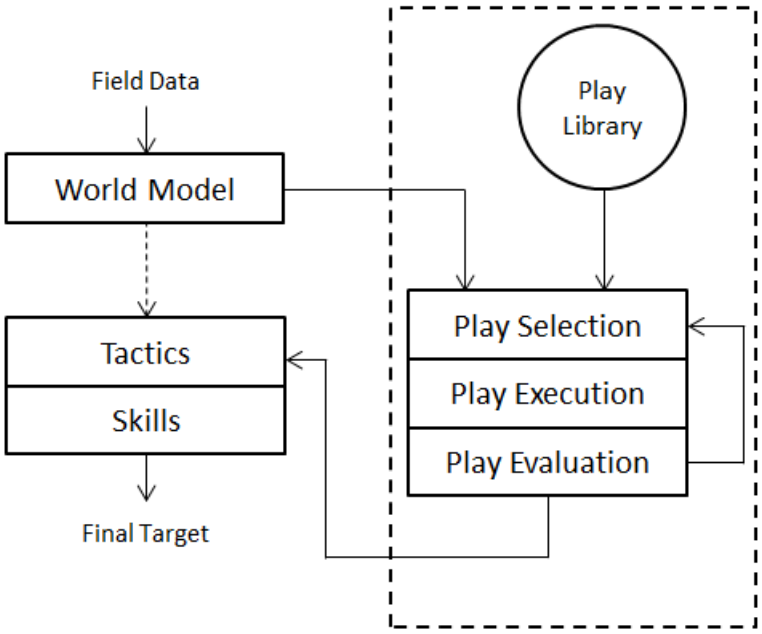

Figure 43. Illustration of STP Architecture

```
Process STP Execution

UpdateWorldModel()
P ← ExecutePlayEngine()

for each robot i ∈ {1, . . . , N}
    (Ti, TParamsi) ← GetTactic(P, i)
    (SSMi, SParamsi) ← ExecuteTactic (Ti, TParamsi)
    if NewTactic(Ti)
        then Si ← SSMi(0);
    (commandi,S'i) ← ExecuteStateMachine(SSMi,Si,SParamsi);
    robot_commandi ← ExecuteRobotControl(commandi);
    SendCommand(i, robot_commandi);
```

Figure 44. Pseudo code of STP Architecture

5.2 Tactics and Skills for Single Robot Control

In the STP architecture, the mechanism to control a single robot needs *tactics* and *skills*. As mentioned above, through *plays* the former provides interface for whole team control while the latter forms SSM to achieve complex behavior for *tactics*. The following part will firstly introduce tactics and skills, followed by skills and

evaluation module. The next section will take passing as an example to introduce the evaluation module in detail. /4/

5.2.1 Tactics

In the STP architecture, *Tactics* is at the top level of single robot control, which contains behavior of a single robot. Different parameters for each tactic in a relatively smaller set can result in a wider range of behavior. **Figure 45** lists some of the tactic class we have developed now. It should be intuitive to know the function of each tactic through its name. /4/

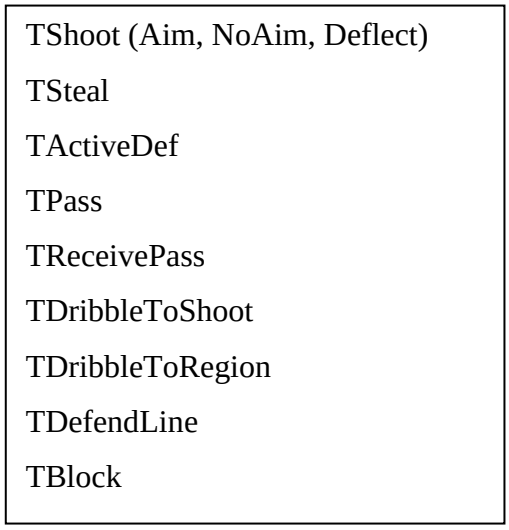

Figure 45. List of Tactics

During the game, each robot owns one tactic with corresponding parameters, determined by the current play. The tactic will not terminate until the play transits to the next one of the tactic sequence. As mentioned in the introduction above, each tactic determines and sets parameters for the SSM to execute. Those parameters, generated by the evaluation process which will be introduced later, include a target position with specific direction and velocity, target points to shoot at with given kicking power. Because of the different parameters, although some different tactics contains many of the same skills, the tactics goals can also be quite different. Take *TShoot* and *TPass* as an example, shown in the **Figure 42**. The skills in the corresponding SSM are quite similar, but the behaviors are much different due

to different parameter. Inevitably, to achieve satisfying behaviors, according to the actual need, the tactic may have local parameter. /4/

Figure 46 clarifies the pseudo code of algorithm for the *TPass* tactic class. In this example, the program firstly evaluates the current situation to ensure the target position, the ball target position shoot at and the shoot angle tolerance. After the evaluation, the tactic assigns corresponding parameters to the skills of SSM. The parameters *MoveBall* and *BallShotPass* belong to the command parameter set for SSM which will be introduced in the next section.

```
Tactic TPass(i)
{
(ball_target, target, angle_tolerance) ← evaluation.aim(position,
position(i), ball_position,error);
SParami ← setCommand(MoveBall, target, ball_target, angle_tolerance,
BallShotPass)
}
```

Figure 46. Pseudo Code of TPass Tactic Class

5.2.2 Skills

Different combinations of skills form different SSM for tactics, whose state machine actual execution sequence obviously depends on the current world state. For example, if a robot attempts to dribble the ball to the opponent's defense zone, it should possibly (a) go to the ball, (b) put the ball onto the dribbler, (c) push the ball with dribbler to the target position. Also, if the ball is closed to the border, the robot may firstly spin to get the ball which would lead to a different execution sequence. That is how the current world state influences the actual execution sequence of SSM. **Figure 47** illustrates the tactics, skills and SSM in the STP architecture. As mentioned above, each tactic with different command parameter can lead to a specific SSM.

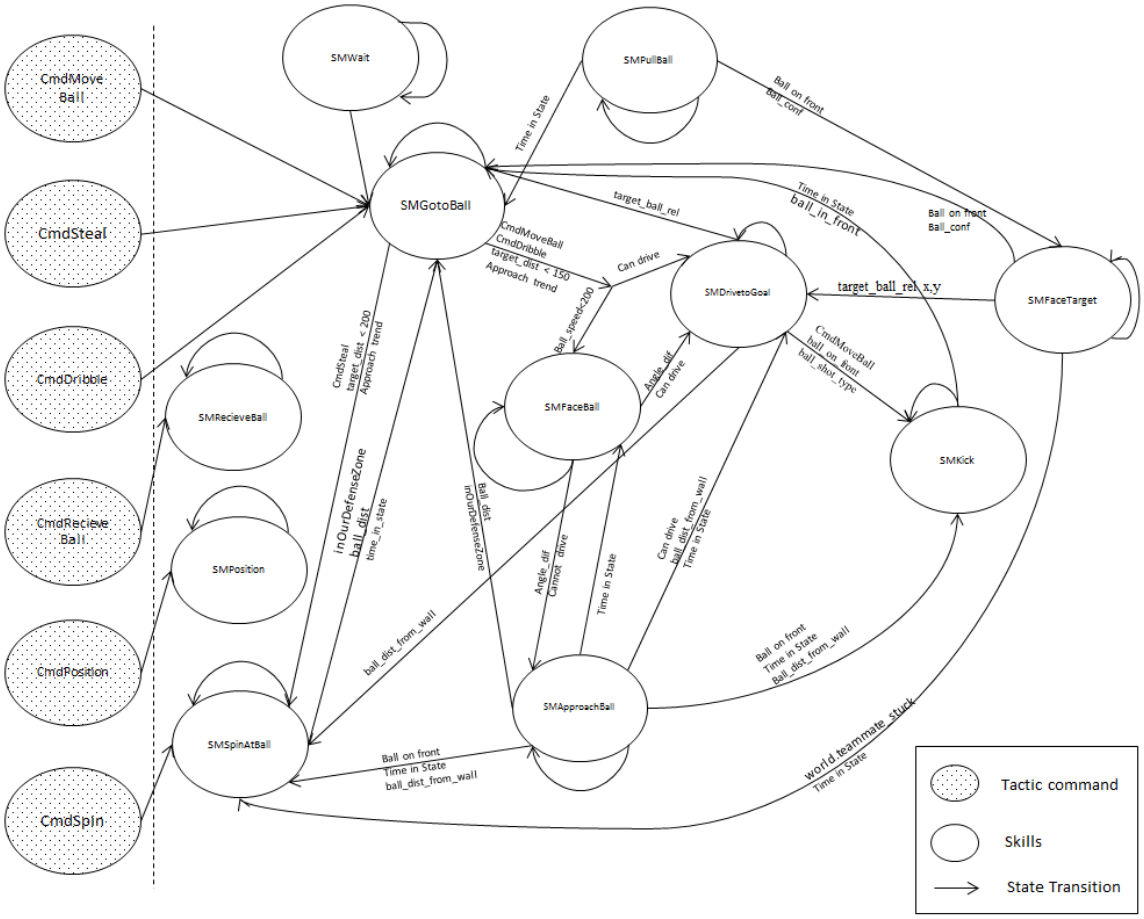

Figure 47. Tactics, Skills and SSM in STP Architecture

In our system, each skill is an independent behavior, forming a unique state in the SSM. Unlike the tactic, which will not terminate until the play transitions to the next tactic, at every time step, each skill will transit to itself or to another skill in the SSM based on the current world state.

Three key components form each skill, evaluation, command generation and state transitions. The evaluation step aims to utilize or generate necessary predicates according to the current world model. As mentioned in the previous chapter, in the world modelling, every time after the vision update is complete, the program will accordingly update commonly used predicates, which can avoid redundant calculation duplication waste. The command generation step intends to generate proper parameters to control the robot for the path planning and safe navigation or speed planning, which has been introduced in previous chapter. The state transition step allows the current state to transit to the next proper skill in the SSM which can be

another skill or itself. The arrays in the **Figure 47** show the transition conditions, set by the tactics or SSM, such as the continuous total time in one state, the position of executing robot, etc. That is why the same skill can be repeatedly applied in different tactics or SSM and also in different condition for the same tactic or SSM. Obviously, this mechanism reduces the code duplication. /4/ /19/

Figure 48 shows the algorithm for *FaceBall* skill, which is the *SMFaceBall* in **Figure 47**. *FaceBall* skill allows the robot to face to the target ball. The skill firstly evaluates current situation and generates advanced necessary predicates, such as can_drive, which is derived from the data in the world modelling step after the vision data processing. After the evaluation, the skill transits to DriveToGoal (SMDriveToGoal) or ApproachBall (SMApproachBall) based on the previous evaluation results. If no condition satisfies, it then generates an essential command and transits to itself.

```
Skill Execution SMFaceBall(i)
    if (direction_difference<THRESHOLD) then
        if (can_drive)
        then Transition(SMDriveToGoal)
        else if (NOT can_drive)
          then Transition(SMApproachBall)
    else then
      command_i.navigation.direct <- true
      command_i.navigation.velocity <- calculateVelocity()
      Transition(SMFaceBall)
```

Figure 48. FaceBall Skill Algorithm

5.2.3 Evaluation Module

According to **Figure 46**, the function *evaluation.aim* is part of the Evaluation Module, which intends to promote the aim process. In summary, the Evaluation Module helps to evaluate different alternatives to make optimal decisions throughout the execution of STP architecture. Similarly, to reduce code redundan-

cy, this system combines these evaluations to the Evaluation Module and divides them to three types of evaluations, aim, defense and target positions. /4/

- Aim

This type of evaluation calculates the optimal angle for the robot to kick the ball while trying to avoid a collision.

- Defense

This type of evaluation provides the optimal defense strategy, such as point defense, line defense and block defense, etc.

- Target Position

The last type of evaluation determines the optimal target position to complete a given task. Take *TReceivePass* as an example. The Evaluation Module here helps to find an optimal position for the robot to receive a pass.

Figure 49 illustrates those evaluations. Mark NO.1, NO.2 and NO.3 respectively show three different evaluations at different situations.

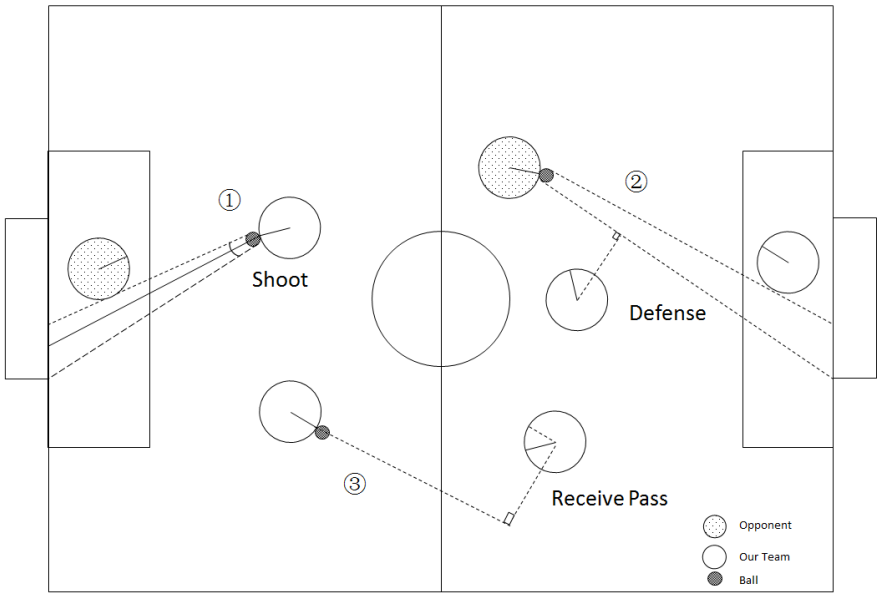

Figure 49. Illustration for Evaluation Module

5.3 Passing Evaluation

Passing is a very common tactic in RoboCup, when, for example, an offense robot with the ball is blocked by the front defense robots of opponent team. In such example case, to increase the score probability, the robot will firstly evaluate the current situation, select a teammate with higher advantage, then choose a pass location and finally pass. The process described above is a part of evaluation module and this section will take it as an example to introduce the passing algorithm and then simulate the process.

5.3.1 Pass-ahead Location Selection

To establish a mathematical model, the passing robot is defined as **P** and the receiving robot is defined as **R**. For the opponent team, the defense robot is defined as **D** and the goalie is defined as G.

The passing tactic here is called direct chip deflection ‘header’, or called pass-ahead. In such a tactic, P utilizes chip pass over *d* at a height midway up the field and then R deflects the ball directly into the goal directly into the goal. /6/

Figure 50 shows the initial situation before passing, simulated by Matlab. The yellow robot with the ball is P and the other yellow robot is R, while the red robot in the half-circle, defense zone, is **G** and the other is **D**.

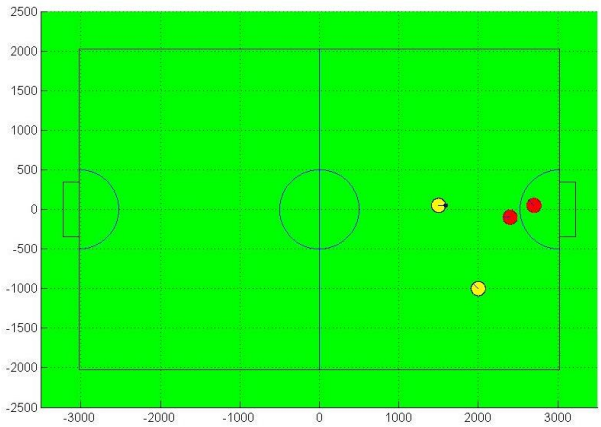

Figure 50. Initial situation before passing

The goal of passing is based on the situation that the current offense robot with the ball has a relatively low score probability. So passing tactic intends to pass the ball to another teammate at position **x***, and maximize its score probability. Definitely, the positions, **x***, of all robots should

$$\mathbf{x}^* \in \mathbf{R}^2 \tag{6}$$

Where $\mathbf{R}^2$ is the set of all positions in the game field domain.

For this target, we define

$$\mathbf{x}^* \equiv \underset{x \in \boldsymbol{R}^2}{arg\ max}[P(goal|\mathbf{x})] \tag{7}$$

As the passing process can be divided into two main parts, passing-receiving (R successfully receives the ball) and shooting (R successfully scores a goal), that means the probability is determined by two probabilities:

$$\mathbf{x}^* \equiv \underset{x \in \boldsymbol{R}^2}{arg\ max}[P(receive|\mathbf{x})P(goal|receive,\mathbf{x})] \tag{8}$$

As our RoboCup game field is very complicated and highly dynamic, it is impossible to exactly calculate the probability. Therefore, we can only try to approximate the real situation as close as possible, by defining several main conditions c_i in the execution process at position **x**. Thus the approximating probability of successful receiving pass is defined as:

$$\hat{P}(receive|x) \equiv \prod_{\boldsymbol{i}} \hat{P}(c_i|\boldsymbol{x}) \tag{9}$$

Therefore, at a specific position, given **x**, the probability for R to successfully receiving a pass is determines by $\hat{P}(c_i|\boldsymbol{x})$, the approximating probability for each condition, c_i.

To calculate each probability, we assume that, in such situation in Figure 51, the velocity of all objects is **0** before passing. The position of **R** is $\mathbf{x_R}$ while the position of **D** is $\mathbf{x_D}$. In addition, due to the physical limitations of robot and SSL rules,

- Robot has a maximum acceleration value $\mathbf{a_{Rmax}}$,

- Robot has a maximum velocity $\mathbf{v_{Rmax}}$,
- Ball has a maximum velocity $\mathbf{v_{Bmax}}$.

The main conditions c_i are:

- c_1: No opponent is able to be faster than R to arrive at **x**. $\hat{P}(c_1|\boldsymbol{x})$~0 if at least one opponent can be faster than R to reach **x**; otherwise $\hat{P}(c_1|\boldsymbol{x})$~1.

  We can compare the navigation time for each robot respectively. For D, to reach **x** as soon as possible, directly towards **x**, it should (a) accelerate with maximum acceleration until $\mathbf{v_{Rmax}}$. If in such case, it has not reached **x**, D will (b) keep $\mathbf{v_{Rmax}}$ until **x**. The navigation time is t_d, so for (a)
  if

$$\frac{1}{2}a_{Rmax}t_d^2 = |\boldsymbol{x} - \boldsymbol{x_D}| \tag{10}$$

  Therefore,

$$t_d = \sqrt{2|\boldsymbol{x} - \boldsymbol{x_D}|/a_{Rmax}} \tag{11}$$

  if current speed of D, v_D is smaller than the maximum velocity $\mathbf{v_{Rmax}}$,

$$a_{Rmax} * t_d \leq |\mathbf{v_{Rmax}}| \tag{12}$$

  Else, for (b), if v_D is greater than the maximum velocity $\mathbf{v_{Rmax}}$,

$$a_{Rmax} * t_d \geq \mathbf{v_{Rmax}} \tag{13}$$

  After the acceleration, D runs in uniform linear motion for t'. The acceleration distance S_a and time *t* is

$$2a_{Rmax}S_a = \mathbf{v_{Rmax}}^2 \tag{14}$$

$$S_a = \frac{1}{2}a_{Rmax}t^2 \tag{15}$$

  So the uniform linear motion time,

$$t' = (|\boldsymbol{x} - \boldsymbol{x_D}| - S_a)/|\mathbf{v_{Rmax}}| \tag{16}$$

  Thus, the total navigation time for case (b) is

$$t_d = \mathrm{t} + t' = |\mathbf{v_{Rmax}}|/a_{Rmax} + (|\boldsymbol{x} - \boldsymbol{x_D}| - \mathbf{v_{Rmax}}^2/2a_{Rmax})/|\mathbf{v_{Rmax}}| \tag{17}$$

  So the navigation time t_d is

$$\begin{cases} \sqrt{2|\boldsymbol{x}-\boldsymbol{x_D}|/a_{Rmax}}\,,\ 2a_{Rmax}|\boldsymbol{x}-\boldsymbol{x_D}| < \mathbf{v_{Rmax}}^2 \\ |\mathbf{v_{Rmax}}|/a_{Rmax} + (|\boldsymbol{x}-\boldsymbol{x_D}| - \mathbf{v_{Rmax}}^2/2a_{Rmax})/\,|\mathbf{v_{Rmax}}|\,,\ 2a_{Rmax}|\boldsymbol{x}-\boldsymbol{x_D}| \geq \mathbf{v_{Rmax}}^2 \end{cases} \tag{18}$$

Similarly, for R, we can get the total navigation time t_r,

$$\begin{cases} \sqrt{2|\boldsymbol{x}-\boldsymbol{x_R}|/a_{Rmax}}\,,\ 2a_{Rmax}|\boldsymbol{x}-\boldsymbol{x_R}| < \mathbf{v_{Rmax}}^2 \\ |\mathbf{v_{Rmax}}|/a_{Rmax} + (|\boldsymbol{x}-\boldsymbol{x_R}| - \mathbf{v_{Rmax}}^2/2a_{Rmax})/\,|\mathbf{v_{Rmax}}|\,,\ 2a_{Rmax}|\boldsymbol{x}-\boldsymbol{x_R}| \geq \mathbf{v_{Rmax}}^2 \end{cases} \tag{19}$$

Obviously, the $\hat{P}(c_1|\boldsymbol{x})$ is

$$\hat{P}(c_1|\boldsymbol{x}) = \begin{cases} 1, & \mathrm{t}_r < \mathrm{t}_d \\ 0, & \mathrm{t}_r \geq \mathrm{t}_d \end{cases} \tag{20}$$

- c_2: No opponent can intercept the pass. $\hat{P}(c_2|\boldsymbol{x})$~0 if **D** can navigate to a position along the line between the original pass position $\boldsymbol{x_0}$ and destination **x** and then intercept the ball; otherwise $\hat{P}(c_4|\boldsymbol{x})$~1.

Figure 51 illustrates the ball passing trajectory in 2D. To intercept the pass, D must move to the position between $f(O_1)$ and $f(O_2)$, the center of two critical robots respectively, where f(x) is the function that converts the 2D position to 3D position.

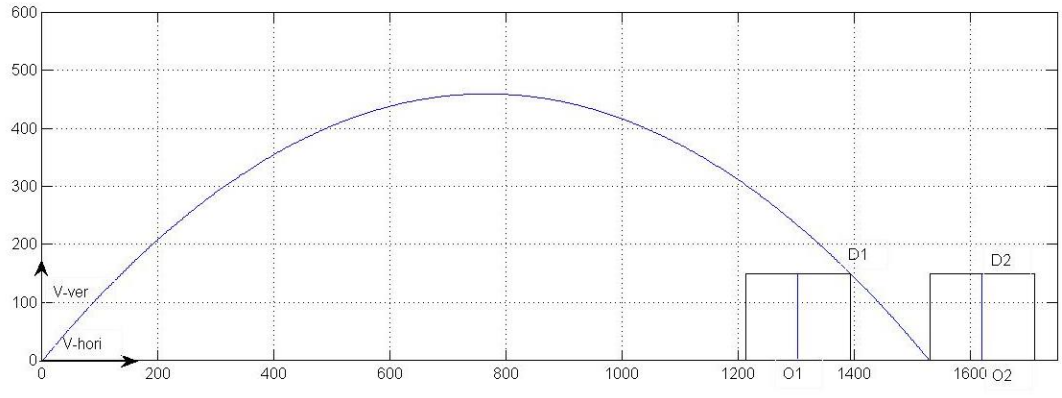

Figure 51. Ball Passing Trajectory Illustration

Assume that the vertical velocity of the ball is **V_ver** while the horizontal velocity is **V_hori** and time to specific position is $\mathbf{t_b}$. The height and distance of

ball is $\boldsymbol{H}$ and $\boldsymbol{S}$, the gravity acceleration is $\boldsymbol{g}$, the height and radius of robot is H_r and R_r, respectively. So the kinematics equations on both vertical and horizontal direction are,

Horizontal: V_hori* t_b = S (21)

Vertical: H = V_ver* t_b - $\frac{1}{2}gt_b^2$ (22)

$$H \in [0, H_r]$$

$$S \in [0 \ , \ |\boldsymbol{x} - \boldsymbol{x_D}|]$$

So the time for the ball to specific position t is

$$t_b = \frac{V\_ver + \sqrt{V\_ver^2 - 2gH}}{g} = S/V\_hori \quad (23)$$

Assume that the intercept position of D is O,

$$\mathbf{O} \in [\boldsymbol{f}(\boldsymbol{O_1}), \boldsymbol{f}(\boldsymbol{O_2})]$$

For $\boldsymbol{O_1}$, in equation (22) and (23), set H as R_r and use equation (24),

$$\boldsymbol{O_1} = V\_hori\frac{2V\_ver}{g} - R_r \quad (24)$$

For $\boldsymbol{O_2}$, in equation (22) and (23), set H as 0 and use equation (24),

$$\boldsymbol{O_2} = V\_hori\frac{V\_ver + \sqrt{V\_ver^2 - 2gH_r}}{g} + R_r \quad (25)$$

So, according to the equation (19), the navigation time t_d of D is,

$$\begin{cases} \sqrt{2|\boldsymbol{O} - \boldsymbol{x_D}|/a_{Rmax}}, & 2a_{Rmax}|\boldsymbol{O} - \boldsymbol{x_D}| < \mathbf{v_{Rmax}}^2 \\ |\mathbf{v_{Rmax}}|/a_{Rmax} + (|\boldsymbol{O} - \boldsymbol{x_D}| - \mathbf{v_{Rmax}}^2/2a_{Rmax})/|\mathbf{v_{Rmax}}|, & 2a_{Rmax}|\boldsymbol{O} - \boldsymbol{x_D}| \geq \mathbf{v_{Rmax}}^2 \end{cases}$$

After time t_d of passing, using equation (23), set t_b as t_d, the height of the ball H is,

H = V_ver* t_d - $\frac{1}{2}gt_d^2$ (26)

Obviously, the $\hat{P}(c_2|\boldsymbol{x})$ is

$$\hat{P}(c_2|\boldsymbol{x}) = \begin{cases} 1, & H < Hr \\ 0, & H \geq Hr \end{cases} \quad (27)$$

- c_3: R has enough time to receive the pass. $\hat{P}(c_3|\boldsymbol{x})$~0 when the ball passing time to **x** is longer than a minimum reaction time t_{min} of R; otherwise $\hat{P}(c_3|\boldsymbol{x})$~1.

  In our frame, the vision frame is about 60 bps, thus the least reaction for a robot is 1/60 s. Assume in the maximum velocity, the distance is much smaller than the safe distance which will be introduced in c_4,

  $$\mathbf{v_{Rmax}}*1/60 << 2*\text{robot\_radius} \quad (28)$$

  So the

  $$\hat{P}(c_3|\boldsymbol{x})=1$$

- c_4: The receiving location **x** is suitable for pass receiving. $\hat{P}(c_4|\boldsymbol{x})$~0 if the x is too close to the opponent defense area, which is forbidden by the rules, or if R reaches x, this can cause a collision with other robot or cause out-of-bounds; otherwise $\hat{P}(c_4|\boldsymbol{x})$~1.

  The positions of **P**, **R**, **G** and **D** are respectively $\mathbf{x_0}$, **x**, $\mathbf{x_G}$ and $\mathbf{x_D}$ and the position set of defense zone is **F**. So $\hat{P}(c_4|\boldsymbol{x})$ is

  $$\hat{P}(c_4|\boldsymbol{x}) = \begin{cases} 1, & |\boldsymbol{x}-\boldsymbol{x_0}| > 2*\text{robot\_radius AND} \\ & |\boldsymbol{x}-\boldsymbol{x_G}| > 2*\text{robot\_radius AND} \\ & |\boldsymbol{x}-\boldsymbol{x_D}| > 2*\text{robot\_radius AND} \\ & |\boldsymbol{x}-\boldsymbol{x_F}| > THREAHOLD\ (\boldsymbol{x_F} \in \text{F}) \\ 0, & \text{otherwise} \end{cases} \quad (29)$$

- c_5: The passing ball exactly or nearly after R reaches **x**. $\hat{P}(c_5|\boldsymbol{x})$~0 when the passing ball reaches relatively much later after R or earlier before R reaches; otherwise $\hat{P}(c_5|\boldsymbol{x})$~1.

  Based on the conditions that c_1 and c_2 are both true, let us assume that the vertical velocity of the ball is **V_ver** while the horizontal velocity is **V_hori** and time to specific position is $\mathbf{t_b}$. The time from $\mathbf{x_R}$ to **x** is $\mathbf{t_r}$. According to the equation (20) and (24), $\mathbf{t_r}$ and $\mathbf{t_b}$ is respectively,

  $$\begin{cases} \sqrt{2|\boldsymbol{x}-\boldsymbol{x_R}|/a_{Rmax}}, & 2a_{Rmax}|\boldsymbol{x}-\boldsymbol{x_R}| < \mathbf{v_{Rmax}}^2 \\ |\mathbf{v_{Rmax}}|/a_{Rmax} + (|\boldsymbol{x}-\boldsymbol{x_R}| - \mathbf{v_{Rmax}}^2/2a_{Rmax})/|\mathbf{v_{Rmax}}|, & 2a_{Rmax}|\boldsymbol{x}-\boldsymbol{x_R}| \geq \mathbf{v_{Rmax}}^2 \end{cases}$$

  $$t_b = \frac{\text{V\_ver}+\sqrt{\text{V\_ver}^2-2gH}}{g} = S/\text{V\_hori}$$

  So, obviously

$$\hat{P}(c_5|\boldsymbol{x}) = \begin{cases} 1, & t_r \approx THRESHOLD * t_b \; AND \; t_r < t_b \\ 0, & \mathrm{H} \geq \mathrm{Hr} \end{cases} \tag{30}$$

Based on the theory above, we utilized Matlab to simulate the probability $\hat{P}(receive|x) \equiv \prod_i \hat{P}(c_i|\boldsymbol{x})$ and got the result shown in the following. The probabilities for each position are shown in grayscale where the black is wrong and the white is true. According to the figure, obviously, the position around the R can be the potential candidate positions. The next section will evaluate the shooting process after passing.

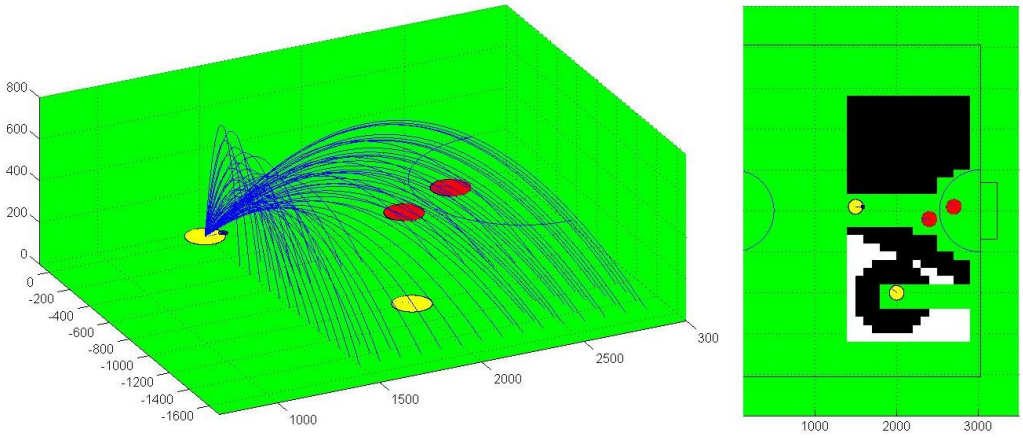

Figure 52. Pass-ahead Probability Distribution

Similarly, $P(goal|receive, \mathbf{x})$ can also be defined as,

$$\hat{P}(goal|receive, \boldsymbol{x}) \equiv \prod_i \hat{P}(c_i'|\boldsymbol{x}) \tag{31}$$

For simplicity, two conditions is defined here,

- c_1': which is the open shooting θ_g angle from **x** to the opposing goal. $\hat{P}(c_1'|\boldsymbol{x})$~0, when there is not shooting angle at all. $\hat{P}(c_1'|\boldsymbol{x}) \rightarrow 1$, when the $\theta_g \rightarrow \theta_{max}$. When $\theta_g > \theta_{max}$, $\hat{P}(c_1'|\boldsymbol{x}) = 1$, meaning that beyond such certain threshold, the θ_g has no influence on the shooting probability.

  The following figure shows an example of shooting angle at the specific position **O**. Usually, the shooting angle is the included angle between the lines to the goal border or tangent lines to block robots. Angles **a** and **b** are the two

shooting angle at this position, so to calculate the shooting probability, set the bigger one, a, as θ_g. The calculation equation is,

$$\hat{P}(c_1'|\boldsymbol{x}) = \begin{cases} \frac{\theta_g}{\theta_{max}}, \theta_g \geq \theta_{max} \\ 1, \ \theta_g \geq \theta_{max} \end{cases} \quad (32)$$

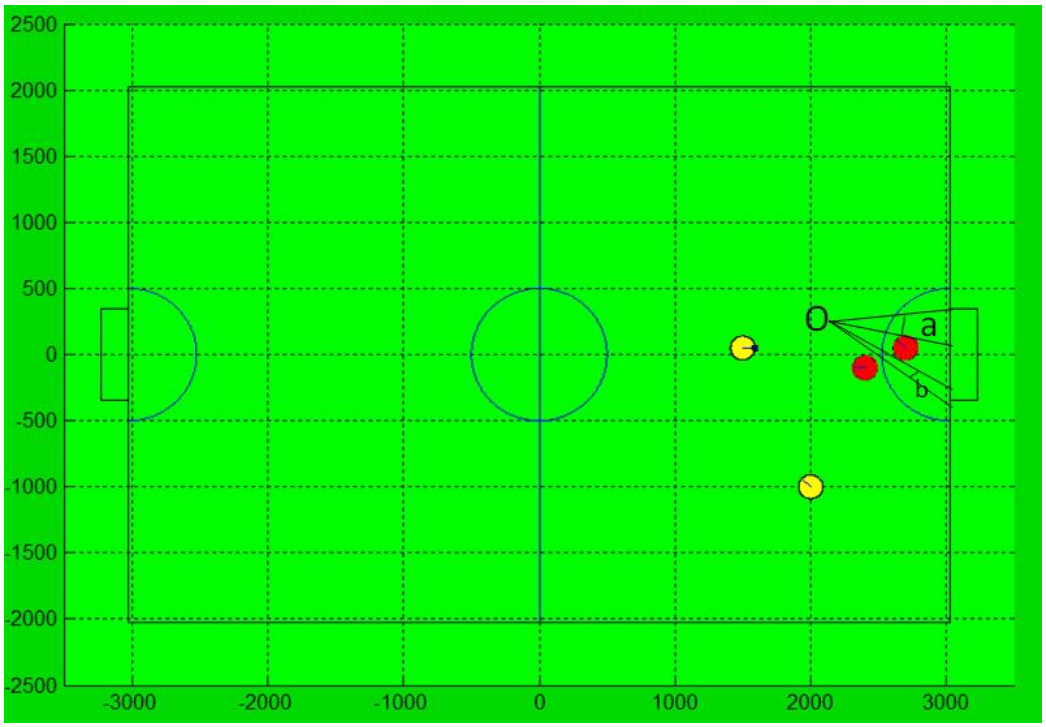

Figure 53. Shooting Angle Example

- c_2': The shooting location **x** is suitable for shooting. This is the same as c_4. So the equation is the same as equation (30). /23/

Based on the theory above, we utilized Matlab to simulate the probability $\widehat{\mathrm{P}}(\mathrm{goal}|\mathrm{receive}, \mathrm{x}) \equiv \prod_{\mathrm{i}} \widehat{\mathrm{P}}(\mathrm{c_i'}|\mathrm{x})$ and got the result shown in the following figure. Also, the probabilities for each position are shown in grayscale where the black is 0 and the white is 1. According to the figure, the positions marked with pink circle have a high probability and thus easier to score. Probability less than 10% are not drawn.

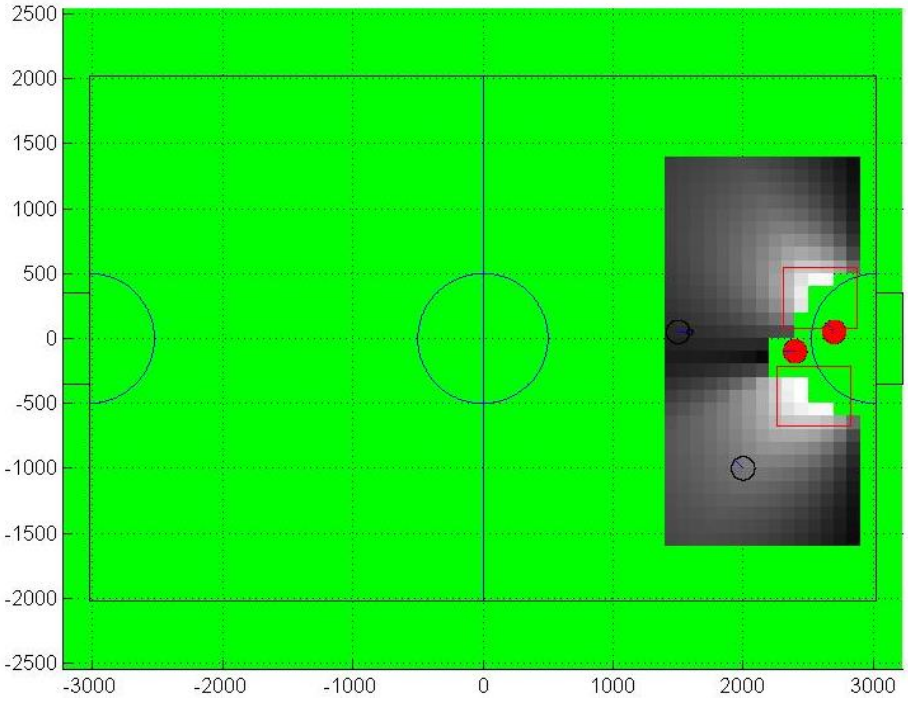

Figure 54. Shooting Probability Distribution

Based on the theory above, we can multiply the two main probabilities together to evaluate the overall pass-ahead tactic,

$$\hat{P}(goal|\mathbf{x}) = \hat{P}(receive|\mathbf{x})\hat{P}(goal|receive,\mathbf{x}) \tag{33}$$

$$\hat{P}(receive|x) \equiv \prod_i \hat{P}(c_i|\boldsymbol{x}) \tag{34}$$

$$\hat{P}(goal|receive,\boldsymbol{x}) \equiv \prod_i \hat{P}(c_i'|\boldsymbol{x}) \tag{35}$$

Similarly, we utilized Matlab to simulate the probability$\hat{P}(goal|\mathbf{x})$. The highest probability,

$$\hat{P}(goal|\mathbf{x}) = 72.43\%$$

$$\mathbf{x^*} = \underset{x \in \boldsymbol{R}^2}{\arg\max}[P(receive|\mathbf{x})P(goal|receive,\mathbf{x})] \tag{36}$$

$$= (2300, -600)$$

pointed by the array, is the optimal position for pass-ahead tactic.

With the theory of shooting evaluation, we can also calculate the goal probability in **x**,

$$\hat{P}(goal|\mathbf{x}) = 15.17\%$$

Significantly, our evaluation module provides a much better choice to goal.

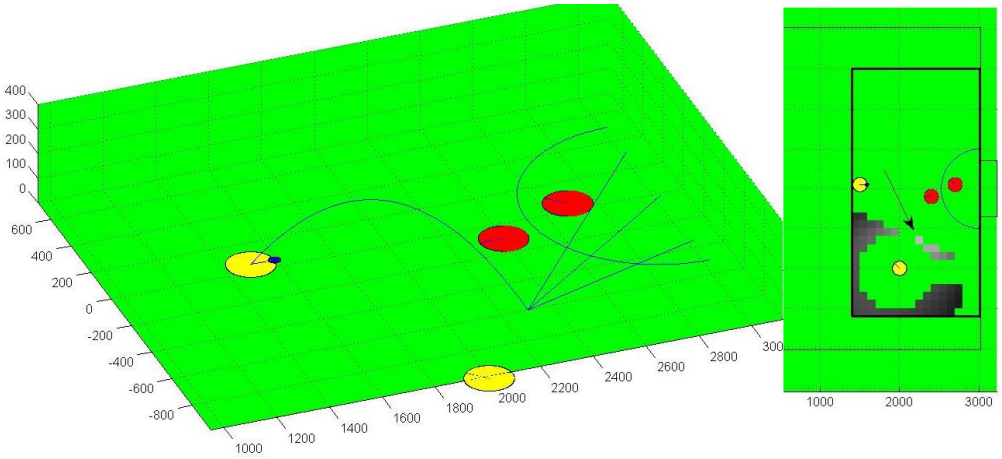

Figure 55. Passing Coordination Probability Distribution

5.3.2 Pass-ahead Coordination

After the evaluation, the rest of the work is the coordination among our robots according to the evaluation result. The coordination aims to minimize the wait time of the receiver robot while ensuring the relatively high shooting probability. The following figure illustrates the algorithm. /23/

Pass-ahead coordination algorithm
$\mathbf{T_r} \leftarrow$ receiver navigation time to receive location $\mathbf{x_r} \approx \mathbf{x}$
$\mathbf{T_s} \leftarrow$ passer navigation time to kicking location $\mathbf{x_p} \approx \mathbf{x_0}$
$\mathbf{T_b} \leftarrow$ minimum ball traversal time from source $\mathbf{x_0}$ to $\mathbf{x}$
$\mathbf{T_p} \leftarrow \mathbf{T_s} + \mathbf{T_b}$

if $T_p \geq T_r$ **then**
  P starts moving to x_p
end if

if $T_r \geq T_p$ **then**
  R starts moving to x_r
end if

if P is at $\mathbf{x_p}$ and $\mathbf{Tr} \approx \mathbf{T_p}$ **then**
  P shoots ball to **x**
end if

Figure 56. Pass-ahead Coordination Algorithm/23/

The reliability of the tactic depends on the accurate estimation of robot navigation time for a specific position and velocity, ball travel time to a given position. Combined with the overall strategy for the whole team, which will be introduced in the next section, such tactics are very useful in the game.

5.4 Plays for Multi-Robot Control

Plays are on the top level of the STP architecture, providing an overall strategy for the whole team. The possible issues concern dynamic role assignment, tactics execution process, etc. In our system, *play* is defined as a team plan, while *playbook* is defined as a set of team plans. The following part will firstly introduce the target of the whole system and then discusses how plays and playbooks solve achieve the target.

5.4.1 Targets

The criteria to judge our team strategy is the performance. The following six targets are the main points for overall performance,

1. Coordinated team behavior,

2. Temporary action extension,

3. Inclusion of special purpose behavior for certain situations,

4. Convenience for human design,

To achieve the first target to the fourth target, the system needs to generate coordinated and sequenced behavior among the team. Also, because of modularization, the play design is separate with the lower layer. Therefore, the design language should be human readable so that one even without much programming skills can design a good play. This also requires the system to be able to execute what the designer describes.

The rest of the targets are advanced targets, which are,

5. Ability to grab opportunities, and

6. Dynamic adaptation to different opponents

Obviously, as the play designer cannot possibly consider all the situations in the play design, this requires the system to automatically grab possible opportunities in the game. The final target requires the system to adapt to different opponents./4/

The following part will describe the play execution system and then will show how our playbook chooses appropriate alternative strategy according to current situations.

5.4.2 Play

As mentioned above, a play is a team plan, which consists of four main parts,

1. Applicability conditions,

2. Termination conditions,

3. Participant roles with tactics, and

4. Execution details.

The applicability conditions refer to in what condition a play can be selected and executed. The termination conditions means in what condition a play can be terminated. The participant roles refer to the participants in a play. The execution details contain a set of extra information, such as tactics parameters, etc.

The play is written in the script language and it is easy to design the strategy. **Figure 57** shows a play script example. Taking this play script as an example, the following will introduce each part individually.

```
PLAY Two Attackers, Shoot 1
APPLICABLE offense
DONE aborted !offense
TIMEOUT 10
ROLE 1
        shoot A
        none
ROLE 2
        block 320 900 -1
        none
ROLE 3
        position_for_pass { R { 1000 0 } { 700 0 } 500 }
        none
ROLE 4
        defend_line { -3025 2025 } { -3025 -2025 } 1375 3025
        none
```

Figure 57. Play Script Example

- Applicability conditions

The applicability condition is triggered based on the current world state. The keyword in the script is *APPLICABLE*, followed by a sequence of logical formula of predicates which is updated in the world modelling module. In Figure 45, the ap-

plicability condition is *offense*, which means that this play will be selected if *offense* predicate is true. **Figure 58** shows some of the available state predicate keywords in our system. The meaning of the predicate should be clear from the name.

offense	our_kickoff
defense	their_kickoff
special	our_freekick
our_ball	their_freekick
their_ball	our_penalty
loose_ball	their_penalty
our_side	ball_x_gt
their_side	ball_x_lt
midfield	ball_absy_gt

Figure 58. Some of Available State Predicate Keywords

With applicability conditions, the strategy designer can restrict a play used for specific condition. For example, in **Figure 57**, only when the current condition is the offense state, will this play be possibly selected and executed.

- Termination conditions

Similarly, the termination condition is triggered based on the current world state, which determines the stop of this play in play script. The keyword of the termination conditions is *DONE*, followed with a *result* of each outcome and logical formula of possible outcome predicates of this play.

The possible results are, *aborted*, *failed*, *succeeded* and *completed*. Those keywords are utilized to evaluate the execution status of current play, which can be used for play selection later. More precisely, succeeded and failed results means whether our team scored or not, by normal offense, or penalty shot, etc. Completed result refers to the condition when the whole play execution finishes. For example, if an offense play completes a shot, no matter if the team scores, the status is considered completed. The aborted result describes the condition when the cur-

rent play stops and does not complete. For example, when the system executes an offense play, if the ball is stolen during the process, the status is considered failed.

The outcome predicates can also be the condition in applicability conditions. In Figure 45, the only termination condition is *!offense*, which means that the play must end when the current status is not offense. Like common logical formulas, '!' signifies negation.

Besides termination conditions after keyword DONE, in our system, two conditions can also result in the current play to terminate. The first condition is triggered when the current status is *completed*, which means our team scores. The second condition is triggered when the current play continues to execute too long time, longer than a threshold (usually 20 seconds), and no other termination conditions are triggered. In this case, the current status changes to *aborted*. This mechanism restricts the maximum execution time of a play to increase the feasibility of the system.

- Participant roles with tactics

Inevitably, the main components of the plays are roles. In our current system, our play supports four roles and the rest one acts as goalie. Figure 57 shows another play script, but different from **Figure 57**, each role has multiple tactics in a sequence. For a robot, it will transit to the next after current tactic completes.

The coordination is a crucial issue of the multi-agent system which requires all the roles to transit at the same time through the sequence of behaviors. For example, in **Figure 59**, the role NO.1 executes *spin_to_region* tactic, which spins the ball to the specific region. Meanwhile, role NO.3 executes *position_for_loose_ball* tactic, which obtains the loose ball spinned by the role NO.1 in the same region. After that, role NO.3 transmits to *shoot* tactic while role NO.1 waits in a specific region for the possible ball.

PLAY Two Attackers, Corner Spin 1

APPLICABLE offense in_their_corner

DONE aborted !offense

TIMEOUT 15

ROLE 1

spin_to_region { R { B 1100 800 } { B 700 800 } 300}

position_for_pass { R { 1000 0 } { 700 0 } 500 }

none

ROLE 2

block 320 900 -1

none

ROLE 3

position_for_loose_ball { R { B 1100 800 } { B 700 800 } 300}

shoot A

none

ROLE 4

defend_line { -3025 2025} {-3025 -2025} 1275 3025

none

Figure 59. Play Script with Multiple Tactics for Each Role

For system flexibility, the roles assignment can be dynamic based on the current situation. This requires the system to scientifically evaluate the situation and generate an appropriate assignment plan. Moreover, the parameters format of tactics is determined by the tactic class. Take tactic class *defend_line* as an example, according to the parameter parsing function and construct function, the parameter should be two point positions to determine a segment and a range with a minimal and maximal value, which is

defend_line $\{x_1,y_1\}$ $\{x_2,y_2\}$ min max

The possible parameters can be a specific point position, a segment, or a rectangle region. This, in another aspect, enables the tactic to adapt to different situations, thus increase the flexibility.

- Execution details

The execution details refer to the rest of the play script. For example, the name of the play follows the keyword *PLAY*. *TIMEOUT* specifies the timeout length of this play, overriding the default timeout in the system. For the detail of script example, please refer to the appendix.

5.4.3 Play Execution

The Play Execution Module aims to assign correctly the tactics of play into corresponding robots and ensure their execution. After parsing and interpreting the play script, this module then assigns roles, switches different roles, distributes tactics, grabs opportunities, and termination.

As mentioned above, the role assignment can be dynamic, which, in other words, means that the role assignment is tactic-specific. Because of this, this module needs role switching based on the actual situations. Tactic distribution intends to transit the tactics according to the play script and situations. When the tactic status is *succeeded*, the play transits to the next tactic for each role. Opportunity grabbing means this module will monitor the situation and immediately react to those valuable opportunities. For example, when the ball is much closed to one of our role and the opponent defense zone, this role will be quickly assigned with a shoot tactic and execute. The termination step happens when the current status fulfills the play script, or the two situations mentioned above. Besides those, the command from the referee, like penalty declaration, can also terminate the game.

5.4.4 Playbook

As mentioned above, the playbook is a set of all plays, providing all possible team behavior. Though it is impossible to design a play which suits all kinds of situations, our system can keep improving by adding a new play for different situations.

Therefore, there exist plenty of plays in the playbook and the key target is to select an appropriate play. For example, if the current status is both *our_ball* and *offense*, which play with accordant *applicability conditions* should be selected?

To solve this problem, in the playbook, our system sets a weight for every play. When p_1 p_2 p_3 …p_k are the plays whose weights are w_1 w_2 w_3 w_k respectively and also satisfy current status. So the probability for a certain play p_j to be selected is

$$\Pr(p_j|\mathrm{w}) = \frac{w_j}{\sum_{i=1}^{k} w_i} \tag{37}$$

Our system just directly signifies the weights in the playbook, but some algorithm like Random Weighted Majority and Exp3 can dynamically adjust the weight according to different opponents. /21/

5.4.5 Implementation

The main implementation files are under folder *strategy_extutor*. The following figure illustrates the overall file structure of STP. Clearly, Skill, Tactic and Play are implemented in the corresponding folder respectively. As the skill is the bottom-level action, most of the function is implemented together with tactic in the *tactic* folder. The CPP (C Plus Plus) file *evaluation.cpp* with its header file implements the evaluation module. The execution and evaluation of the play is implemented in *strategy.cpp* with its header file, which calls the method and variable of others files.

Figure 60. File Structure of STP

- Tactic and skill

As mentioned in Chapter 3, with polymorphism, the tactics are implemented under the *tactic* folder. The following figure shows some of the tactic files, which evaluate the current situation and generate tactic commands for single robots.

Figure 61. Example of Tactic and Skill Files

The SSM (skill state machine), introduced earlier in Chapter 5.2, is implemented in *field_world_robot.cpp* with its header file of knowledge_base/ database/ world_state folder. It defines a class Robot, representing the robot in real world, including the parameters like position, teammates, obstacles, commands, and some methods like go to specific point, etc. The relationship between its methods and the SSM is shown in the following figure. Combined with **Figure 47**, the tactic commands are defined as an enumeration, *CommandType*. Also, the skill states are defined as another enumeration, *SMState*, while the corresponding methods are listed on the right.

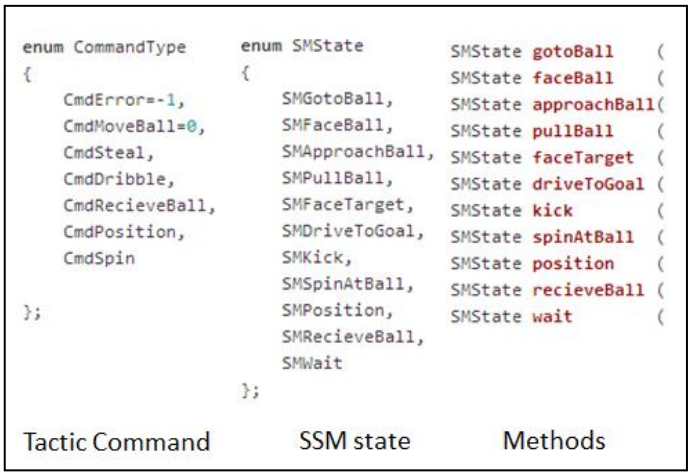

Figure 62. The relationship between methods in class Robot and SSM

- Evaluation

This part implements the function of evaluation module, such as the aim before shooting, defense position planning, etc. The file evaluation.cpp with its header file provide class Evaluation and class EvaluationPosition for evaluation purposes.

- Play

The implementation of Play can be divided into two parts. The first part is the play script *.ply* file, which is located in folder *config/plays/*, and another part is the parser for the script, which is implemented in *play.cpp* and its header file. The relationship between each class, defined in that file, is illustrated in the following figure.

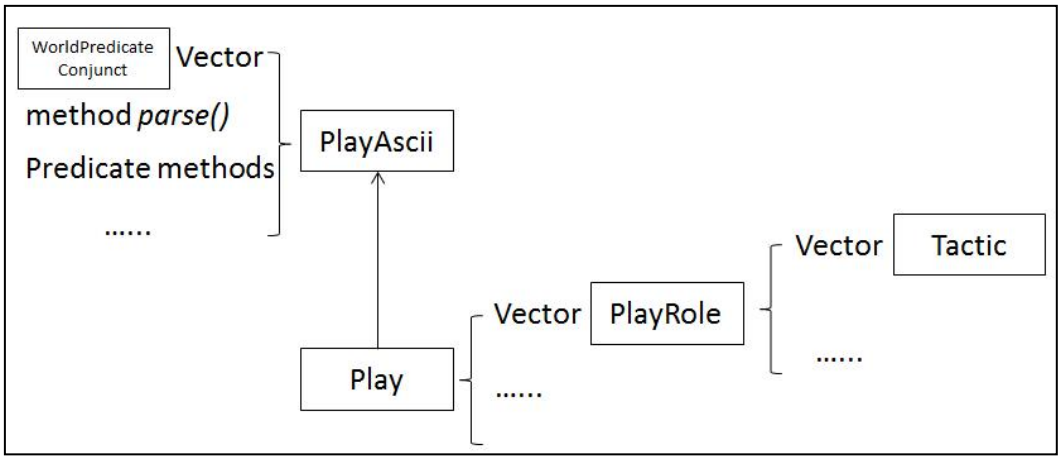

Figure 63. Class Relationship in play.h

As it can be seen the figure, class PlayRole describes the specific role of a robot, including the tactic, represented by class Tactic. Class Play is a set of PlayRole for a team, which is inherited by class PlayAscii. PlayAscii has two main functions, parse and predicate based on current situations. As the play script is programmed not in standard advanced language, but our customized script language, it needs to be parsed by our program, which is class PlayAscii. The pseudo code of parsing algorithm is shown below. The key of this algorithm is the parameter parsing after detection of corresponding keywords.

For the predicate function of class PlayAscii, it provides many methods to predicate the status according to the world model, such as if the ball is on our side, or whether it is in offense now, which determines the applicability and termination conditions.

```
Play Script Parsing algorithm
S ← Load the specific play script
word ← S next word
S ← S - word
while word is end of file
        if word == APPLICABLE
                Applicability_conditions_list=parse(S)
        else
        if word == DONE
                Termination_conditions_list=parse(S)
        else
        if word == FIEXEDROLES
                Fixedroles_list=parse(S)
        else
        if word == TIMEOUT
                Timeout=parse(S)
        else
        if word == ROLE
                Role_Tactic_list = parse(S)
        end if

        word ← S next word
        S ← S - word
end while
```

Figure 64. Parsing algorithm of class PlayAscii

- Strategy

File *strategy.h* defines four classes, PlayExecutor, PlayBook, Warmup and Strategy, whose relationship is shown in the following figure. Class PlayBook, represented the concept of playbook mentioned before, stores all the play, their result and other parameters, like the play name, play weights, etc. Class Executor is responsible for evaluation and execution, given the play in class PlayBook. Also, as introduced above, this class also implements the opportunity grabbing function to grab a possible goal chance thereby increasing strategy feasibility. Class Warmup defines a tactic for the team to warm up before the start of the game. For the Class Strategy, it combines the mentioned class together and executes whole play, the highest level of STP architecture.

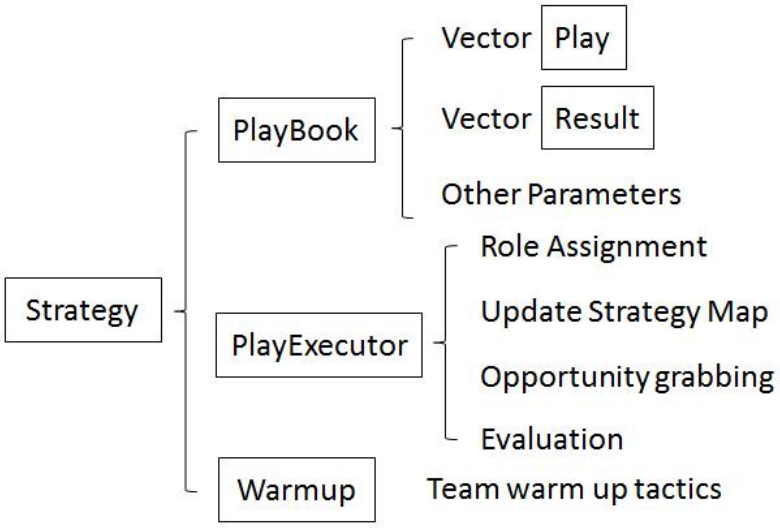

Figure 65. Class Relationship in strategy.h

6 OVERVIEW OF FUTURE RESEARCH

As our Botnia RoboCup SSL project is a huge system, there are many places to improve and complete. Also, we have to make progress to avoid falling behind other teams. Finally, with the development of our society, various theory and approaches are being created, thus it is highly necessary to keep our team with the world. Therefore, this chapter plans an overview of to-do lists for the future work in several aspects, based on current situations.

6.1 Strategy Server

6.1.1 GUI

As the interaction interface, GUI module influences the performance of our system. A user-friendly GUI can provide numerous data for analysis and debug. The figure below shows an example of good GUI.

1. After clicking a robot, a window jumps out showing the current situation of this robot, including the real-time position, speed, electric quality and so on in data form and figure form. Also, the previous and possible future trajectories can also be shown in the field for analysis.

2. A specific game process can be recorded and replayed. Usually our game executes in high speed, it is essential to analyze the process after the match. So game recording is very important.

3. The real-time skills, tactics and plays can also be listed and recorded.

Figure 66. To-Do Lists of GUI Module

6.1.2 STP

- Play Script Language

As our current play script language is customized, it has some limitations, such as optimization, error checking, etc. In the future, we can develop or utilize a robust language and compiler thereby increasing the overall performance and separating a new module, play design module. Many popular programing languages for AI (Artificial Intelligence) can be a good example, like IPL, Lisp, Prolog, STRIPS and so on. In such case, our development can be divided into system development team and strategy development team.

- Offense

Although, with method of opportunity grabbing, our system has some feasibility, the current offense strategy mainly depends on the logic of the developer of the play script. For the future, to obtain high feasibility, Machine Learning can be uti-

lized in this system so that the system can think itself to plan the defense strategy. /24/

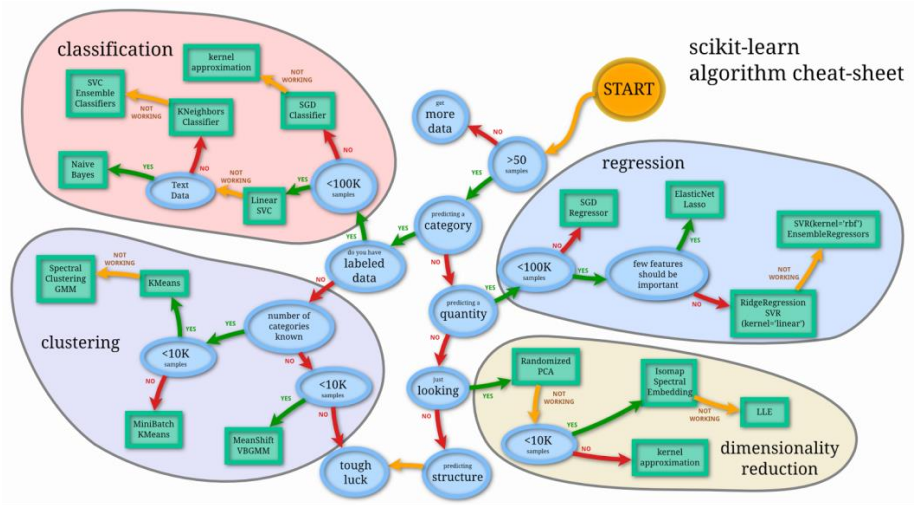

Figure 67. Example of Machine Learning Algorithm

- Defense

If the offense strategy is called active strategy, the defense strategy can be called passive strategy which should predict the possible trend of the game. In our current system, the predication is based on pure theoretical analysis. For example, to calculate the probability of an opponent defense robot, our system will analyze the closest robot to plan the defense strategy. In fact, this mechanism establishes a universal model for all opponents. But as every team strategy is different, it is unnecessary to predict all possible situations for one certain team. Instead, if our system can make prediction based on the characteristics of a specific team, the efficiency will be improved tremendously. To solve this issue, we can apply game training system for our team which builds a certain game parameter for each opponent. While competing with those opponents, the system will update the parameter and adjust the defense strategy./24/

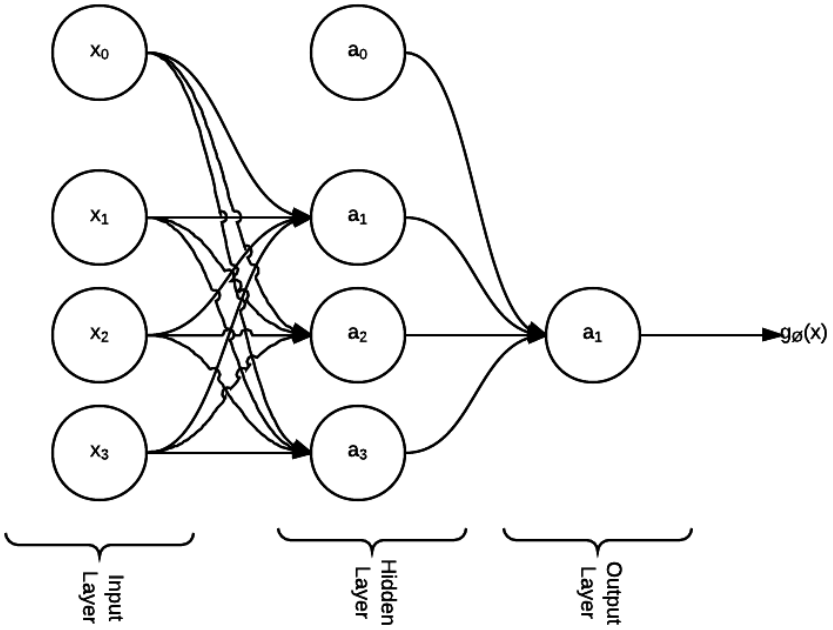

Figure 68. Game Training Model

6.1.3 Vision Data Processing

The main algorithm of vision data processing utilized in our vision server is EKBF (Extended Kalman-Bucy Filters). Although it highly improves the stability of vision data, because of some errors like linearization, the result cannot keep stable, sometimes not even work. Therefore, this algorithm needs optimize by using, e.g. Levenberg-Marquardt approach in iteration step. /25/

6.2 Robot Motion Control

6.2.1 Robot Body Simulation

Robots, as the final executor of the whole system in the game field, must accurately execute the commands given by the strategy system, so a robust embedded motion control algorithm is highly necessary. One of the popular approaches is to utilize Maltab Simulink for the simulation and algorithm parameter generation. In the future, our team can simulate the robot as seen in the following figure for further research.

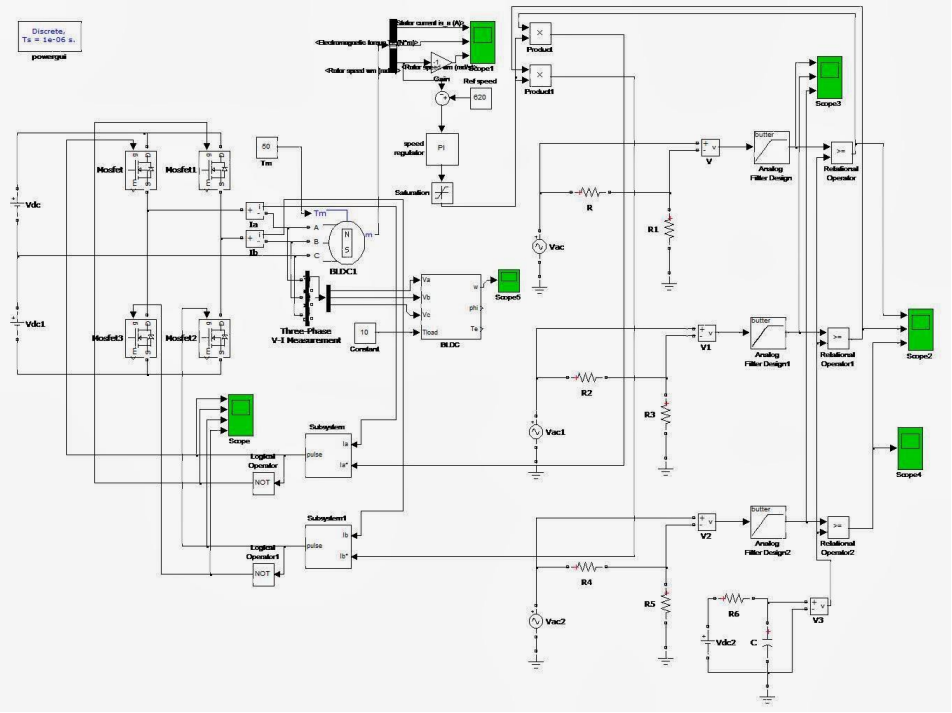

Figure 69. Simulation of Brushless Motor/26/

6.2.2 Intercommunication

Currently, the real-time data of the game field totally comes from the vision server and it is not enough, as our system needs the data from the robot directly, such as the electric quantity, the motor speed, etc. In our current architecture, we have considered this extension in the strategy server, so the rest of the work is in the embedded system of the robot.

7 SUMMARY

This thesis introduces the development of team strategy in detail for Botnia RoboCup Small Size League (SSL) Robot Team.

By the optimization of strategy software and the development of STP architecture for our team strategy, the behavior of the team including overall and single robot improved a lot. It can be concluded that the STP architecture is a very powerful tool which divides complex team control into separate layers and also solves the uncertainty problem by state machine. Moreover, an appropriate mathematical model needs to be established to evaluate the actual situation for decision making. In this thesis, based on the mathematical model, physical motion equations and the probability theory are utilized, which helps to evaluate the running trend and make strategy decision in the field. Last but not least, at the implementation stage, designing robust framework with clear structure and high expansibility is important. Therefore, the key points of this thesis are the process division approach and strict mathematic theory support as well as implementation skills.

There is no success without hard work. To work on such a huge project, you have to keep self-studying by reading academic articles, taking advanced courses, consulting experts, etc. For the students who are interested in this project, this thesis recommends them firstly thoroughly figure out the whole structure of the system. Based on that, the students should then decide the research direction and start work. The direction should not necessarily be very difficult but challenging in some level. During the process of research, students can improve much.

We wish all the students working on this project the best for their future study and career.

8 REFERENCES

/1/ RoboCup. Accessed 26.2.2014. http://www.robocup.org/

/2/ RoboCup Small Size Robot League. Accessed 26.2.2014. http://robocupssl.cpe.ku.ac.th/

/3/ Botnia RoboCup SSL Robot Team. Accessed 26.2.2014. http://robotics.puv.fi/

/4/ Browning, B., Bruce, J., Bowling, M,. & V eloso, Manuela M. 2005. STP: Skills, Tactics and Plays for Multi-Robot Control in Adversarial Environments. IEEE Journal of Control and Systems Engineering 219, 33–52.

/5/ Bin Feng, Yuan Gao. 2013. Development of Strategy Software and Algorithm Simulators for Multi-Agent System in Dynamic Environments. Vaasan ammattikorkeakoulu.

/6/ James, B., Stefan, Z., Mike, L. & Manuela, V. 2008. CMDragons: Dynamic Passing and Strategy on a Champion Robot Soccer Team. 2008 IEEE International Conference on Robotics and Automation Pasadena, vol., no., pp.4074, 4079.

/7/ IEEE Std 1394-2008. 2008. IEEE Standard for a High-Performance Serial Bus. vol., no., pp.1, 954.

/8/ SSL-Vision: The Shared Vision System for the RoboCup Small Size League. Accessed 5.3.2014. http://www.informatik.uni-bremen.de/agebv2/downloads/published/zickler_rs_09.pdf>.

/9/ DECT. Accessed 5.3.2014. http://www.etsi.org/index.php/technologies-clusters/technologies/dect.

/10/ PUV Robotics RoboCup. Accessed 5.3.2014. robotics.puv.fi.

/11/ James, B., Michael, B., Brett, B., and Manuela, V. 2003. Multi-robot team response to a multi-robot opponent team. 2003 IEEE International Conference on Robotics and Automation, vol.2, no., pp.2281,2286 vol.2, 14-19.

/12/ E. Gat. On three-layer architectures. 1997. Artificial Intelligence and Mobile Robots. MIT/AAAI Press.

/13/ R. Simmons, T. Smith, M. B. Dias, D. Goldberg, D. Hershberger, A. Stentz, and R. Zlot. 2002. A layered architecture for coordination of mobile robots. In Multi-Robot Systems: From Swarms to Intelligent Automata. Kluwer.

/14/ James, B., and Manuela V. Real-time randomized path planning for robot navigation. 2002. Intelligent Robots and Systems, 2002, vol.3, no., pp.2383,2388 vol.3.

/15/ Steven M. LaValle. 1998. Rapidly-exploring random trees: A new tool for path planning. In Technical Report No. 98-11.

/16/ Qt Project. Accessed 10.3.2014. http://qt-project.org/

/17/ Bjarne Stroustrup's C++ Glossary. Accessed 12.3.2014. http://www.stroustrup.com/glossary.html

/18/ QThread Class Reference. Accessed 13.3.2014. http://qt-project.org/doc/qt-4.8/qthread.html

/19/ Scott, L., James, B., and Manuela, V. 2002. A modular hierarchical behavior-based architecture. RoboCup-2001: The Fifth RoboCup Competitions and Conferences. Springer Verlag./20/ James, B. 2006. Real-Time Motion Planning and Safe Navigation in Dynamic Multi-Robot Environments. Carnegie Mellon University, School of Computer Science.

/21/ Michael, B., Brett, B., and Manuela V. 2004. Plays as effective multiagent plans enabling opponent-adaptive play selection. In Proceedings of International Conference on Automated Planning and Scheduling (ICAPS'04).

/22/ LaValle, S., and James, J. 2001. Randomized kinodynamic planning. International Journal of Robotics Research, Vol. 20, No. 5, pages 378-400.

/23/ Joydeep, B., Juan, P., and Danny, Z. 2014. Opponent-Driven Planning and Execution for Pass, Attack, and Defense in a Multi-Robot Soccer Team. International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS).

/24/ Mehryar, M., Afshin, R., and Ameet, T. 2012. Foundations of Machine Learning, The MIT Press.

/25/ Madsen, K., and Nielsen H. 2004. METHODS FOR NON-LINEAR LEAST SQUARES PROBLEMS. Technical University of Denmark.

/26/ Akhila, R., and Nikhil, S. 2012. A comparative study of sensor and sensor less control of four-switch Inverter fed Permanent Magnet Brushless DC motor. Power, Signals, Controls and Computation (EPSCICON), 2012 International Conference on , vol., no., pp.1,6, 3-6.

/27/ George, L. 2012. Path Planning Algorithms for Autonomous Border Patrol Vehicles. University of Toronto.

/28/ Vera, B. 2010. Robot Motion Planning. Max-Planck-Institut Informatik.

/29/ Steven, M. 2006. Path Planning. Cambridge University Press.

/30/ Path Planning. Accessed 8.4.2014.
http://people.scs.carleton.ca/~lanthier/teaching/COMP4807/Notes/7%20-%20PathPlanning.pdf

/31/ Visibility Graphs. Accessed 8.4.2014.
http://www.win.tue.nl/~awolff/teaching/2009/2IL55/pdf/v15.pdf

/32/ Robotic Motion Planning: Cell Decompositions. Accessed 8.4.2014.
http://www.cs.cmu.edu/~motionplanning/lecture/Chap6-CellDecomp_howie.pdf

/33/ Introduction to Multi-Agent Programming. Accessed 8.4.2014.
http://www.informatik.uni-freiburg.de/~ki/teaching/ws1011/imap/05_SearchAlgorithmsAndPathPlanningMAS_PartB.pdf

/34/ Motion Planning. Accessed 8.4.2014.
http://msl.cs.uiuc.edu/~lavalle/icra12/slides2.pdf

/35/ Motion Planning 2D Cell Decomposition. Accessed 8.4.2014.
http://robotics.unibg.it/teaching/robotics/pdf/09_Motion_Planning_2D.pdf

/36/ Geraerts, R., Overmars, H. 2002, A comparative study of probabilistic roadmap planners. Proc. Workshop on the Algorithmic Foundations of Robotics (WAFR'02), pp. 43–57.

/37/ Karaman, S. 2011. Anytime Motion Planning using the RRT*. IEEE International Conference on Robotics and Automation (ICRA).

DEFINITION OF TACTIC CLASS TSHOOT

```
class TShoot : public RobotTactic
{
public:
        enum Type { Aim, NoAim, Deflect };
        virtual const char *name() const;
        static double eval_fn(World &world, const MyVector2d p,
                        int obs_flags, double &a);
private:
        Type type;
        int deflect_target;
        MyVector2d prev_target;
        bool prev_target_set;
        EvaluationPosition eval;
public:
        TShoot(Type _shot_type = Aim, int _deflect_target = -1);
        virtual ~TShoot() { }
        void LoadConfig();
        static Tactic *parser(const char *param_string);
        virtual Tactic *clone() const;
        virtual int selectRobot(World &world, bool candidates[], double bi-
as[]);
        virtual void command(World &world, int me, Robot::RobotCommand
&command,bool debug);
        double successProb(World &world);
};
```

DEFINITION OF CLASS OBSTACLE AND OBSTACLES

Field_world_obstacle.h

```
#define OBS_RECTANGLE  0
#define OBS_CIRCLE     1
#define OBS_HALF_PLANE 2

class obstacle
{
private:
      //mask is the flag whether the obstacle is masked
      int type,mask; // type of obstacle, enable mask
      vector2f pos;  // location of center
      //circle: rad.x save rad
      //rectangle: save width and height
      //plane: save the direction and side
      vector2f rad;  // (x,y) radii perpendicular
      vector2f vel;  // object velocity
public:
      double margin(state s);//Calculate the distance between robot and the obstacle
      vector2f closest_point(state s);//Find the closest point of the obstacle to the robot
      bool check(state s);//check whether the robot collides with the obstacle
                          //true: not collide, false: collide
      bool check(state s0,state s1);//check the collision with the straight line
                                    //determined by state s0 and state s1
      vector2f repulse(state s);
};
class obstacles
{
public:
      obstacle obs[MAX_OBSTACLES];
      int num,current_mask;
public:
      obstacles();
      void clear();
      void add_rectangle(float cx,float cy,float w,float h,int mask);
      void add_circle(float x,float y,float radius,
                  float vx,float vy,int mask);
      void add_half_plane(float x,float y,float nx,float ny,int mask);
      void set_mask(int mask);
      bool check(MyVector2d p);
      bool check(MyVector2d p,int &id);
      bool check(state s);
      bool check(state s,int &id);
      bool check(state s0,state s1);
```

```
    bool check(state s0,state s1,int &id);
    vector2f repulse(state s);
}
```

PLAY SCRIPT GRAMMAR

PLAY playname

[**APPLICABLE** [!]<predicate> [!]<predicate> ...]

... (each line is combined with an or operator)

[**APPLICABLE** [!]<predicate> [!]<predicate> ...]

[**DONE** {failure | success} [!]<predicate> ...]

... (each line is combined with an or operator)

[**DONE** {failure | success} [!]<predicate> ...]

[**FIXEDROLES** 1 3 4 2]

ROLE 1

tactic parameters

none

ROLE 2

tactic parameters

none

...

Parameters:

Tcoordinates

[{] [[origin type] side type [d]] coordx coordy [}]

origin type:

none : world origin

A : absolute

B : ball

side type:

null : absolute

B : ball (direction of ball is positive)

A : Absolute

S : strong side of opponent team (side they have most robots)

O : ball side unless ball is in middle then it is S side

optional

d : dynamic: gets recomputed each frame

TRegion:

[{] {R {Tcoord1 Tcoord2 width} | C {Tcoord_center radius} }

R : rectangle

width is width of rectangle where length is along Tcoord1 to Tcoord2