VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Dong Liu

IMPROVING BOTNIA2009 ROBOSOCCER STRATEGY

Information Technology

2011

PREFACE

The coding part of the thesis had been finished at 2010. However, due to my grad-school application, I started writing this document at August of 2011, which makes memorizing became the most difficult part of the thesis. Fortunately I have overcome this problem at last.

I would like to express my thankfulness to my supervisor, Mr. Yang Liu, not only for his help and guidance all through the process, but also for giving me this invaluable opportunity to work for this project.

I would also to thank my parents and my wife. You are the most important spiritual props of me, supporting a self-abased boy, to become a strong man.

Much gratitude to Mr. Kalevi Ylinen, Dr. Moghadampour and those who give me a hand when I encountering different kinds of problems.

Sincerely thanks to all the staff in the VAMK IT Department, for maintaining a excellent studying environment for our international students.

Vaasa, 21.08.2011

Dong Liu

Keyword: Botnia, Robotics, Robosoccer, Vision Filter

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Dong Liu
Title	Improving Botnia2009 Robosoccer Strategy
Year	2011
Language	English
Pages	80 + 1 Appendices
Name of Supervisor	Yang Liu

The old robosoccer strategy application of VAMK is based on the code of Cornell's Big Red Robosoccer team, which is released many years ago.

Nowadays, along with the development of the Robosoccer competition, more and more limitations and defects had been revealed from the old application.

The most critical matter is the locating problem, which is considered as the root of many other troubles. Since the noise might influence the coordinate of the object, nothing can be done if the precise location of required object could not be retrieved.

The purpose of the thesis and the related project is to implement the missing features and address some important issues.

All the important improvements and algorithms have been described in this thesis. All the codes can be checked from the SVN server of our code-base.

The changes made in this project are tested and working quite well in both real world and the simulator (MSBF is tested only in the simulator).

Keyword: Botnia, Robotics, Robosoccer, Vision Filter

CONTENTS

LIST OF ABBREVIATIONS

(1) Botnia2009: The strategy application of Botnia robosoccer SSL software system

(2) SSL: Small Size League

(3) FM: Frequency modulation

(4) GTK+: GTK+ (GIMP Toolkit) is a cross-platform widget toolkit for creating graphical user interfaces.

(5) UDP: User Datagram Protocol

(6) VS 2010 / 2008: Visual Studio 2010 / 2008

1. INTRODUCTION

1.1 About Robosoccer SSL

Robosoccer SSL [1] is a competition league for small sized robot that involved many technologies, for example, robotics, AI and etc. Each competition team will make use of the official system where the data will be provided, make decisions based on the situation, and then let the robots execute the commands for specialized actions.

The official system is consisted by two applications: Vision server and Referee box [2]. The vision server, running SSL-Vision [3], will collect the captured pictures from the camera, and calculate the positions of the robots and the ball on the field, then encapsulate them via Google Protocol Buffer [4] and transmit the package via a UDP multicast address to the subscribing clients.

The Referee box is an application which acts as a referee, controlling the behavior of the competing teams. It will send commands like “stop”, “start”, “free kick”, etc. to the teams at the proper time or when corresponding event is captured. Currently most of its operations are done by human controlling. An automated referee box is under developing and might be used in the future.

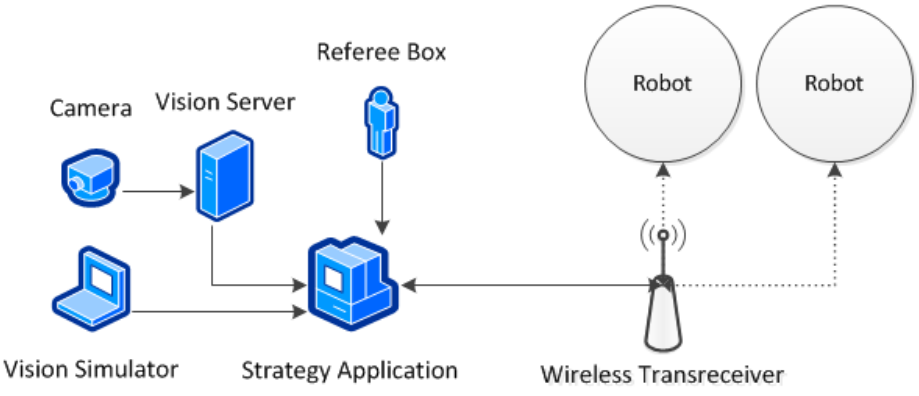

Figure: 1-1: Overall Robosoccer SSL system structure

1.2 Botnia Dragon Knights

Botnia Dragon Knights [5] is a joint international robosoccer team at Vaasa University of Applied Sciences in Finland and Hubei University of Technology in China. The primary goal of this project is to build a robot soccer team that is capable to compete in the RoboCup tournaments, as well as many additional goals to promote the education and research.

Its software system, named Botnia2009, is consists of a strategy system and a vision simulator (Used in test environment where a vision system is not available). The strategy system will process the incoming data from the official system then transmit the command to our robots wirelessly via HW 86012 DECT Module [6] after analyzing and calculation.

The original Botnia2009 is based on QT [7] 4.5.0 LGPL version and Visual C++ 2008.

2. ENVIRONMENT UPGRADE

2.1 Why Upgrade

As QT 4.5.0 is outdated, a plan is carried out to upgrade the framework to the latest 4.7.3, which has many issues patched, to address some crash/freeze problems in the Botnia2009.

```
#include "../common/wireless/qextserialport.h"
#include "RefBoxReceiveThread.h"
#include <QTime>

//Global Timer
extern QTime HBGlobalTime;
extern int HBGlobalTimer;

//Referee Box
extern REFBOX_CMD RefboxCmd;
extern REFBOX_CMD lastRefboxCmd;
extern int internalRefCount;

//Field Analyse
extern float minacityData[5];

//FreeKick Flags
extern bool placeKickUsGo;
extern bool placeKickUsPre;

//Serial Port
extern QextSerialPort* serialComm;
extern int serMode;
extern QByteArray* serialPacket;

//JoyStick
extern bool internalJoy;
extern int internalJoyvx;
extern int internalJoyvy;
extern int internalJoyva;
extern int internalKickPower;
extern bool internalJoyKick;
extern bool internalJoyChipKick;

//options
extern int overallVelocityLimit;
```

Figure 2-1

On the other hand, since there are lots of testing-purpose-constants stored in a single header file (Figure 2-1) in order to provide convenience for students to man-

age, modifying any of them will result in a recompilation of many source files, which will cost a lot of time.

Fortunately, the multi-processer compilation (/mp parameter) is supported in the Visual C++ 2010. This feature can utilize the power of multi-processor/core system, bringing a huge performance boost to the build process.

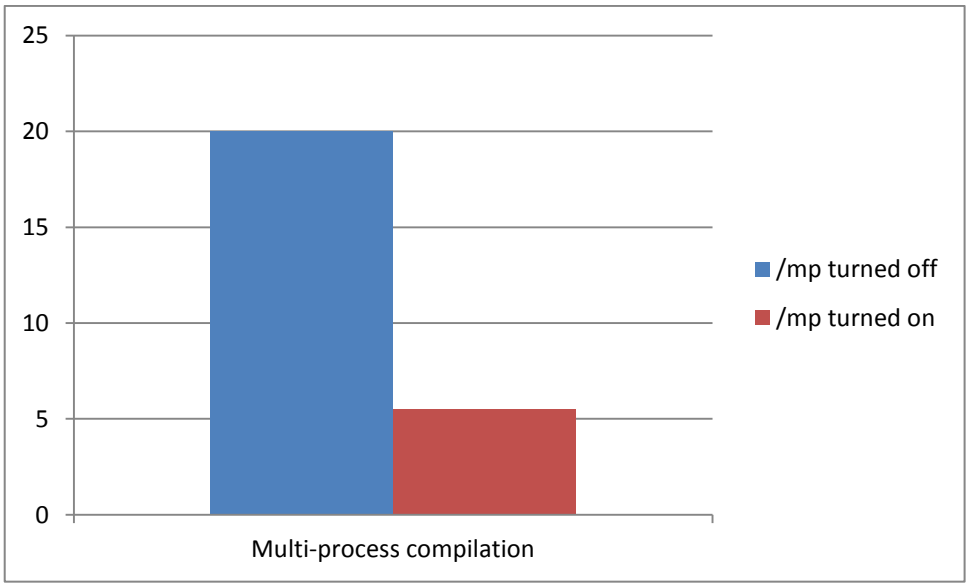

Figure 2-2: Build time (min) on a Core2 Q9300 before/after /mp turned on

Since the Service Pack 1 of the Visual Studio 2010 is released, considering the bug fixes and the improvements to the compiler it might bring, we finally decided to upgrade our environment to QT 4.7.3 + Visual C++ 2010 SP1

2.2 Update the QT framework

This step is pretty easy because installation pack can be directly downloaded from http://qt.nokia.com/downloads. Based on the previous experience, the environment configuration could be done in the following steps:

(1) Download then install the “Qt libraries 4.7.3 for Windows (VS 2008)”

(2) Download then install the “Qt Visual Studio Add-in”

(3) Open our project in Visual Studio and build (The configuration of Visual Studio 2010 will be introduced later)

The old instructions worked well on the build machine. However when we copy the binaries to some other computers, we will encounter the following error in Figure 2-3:

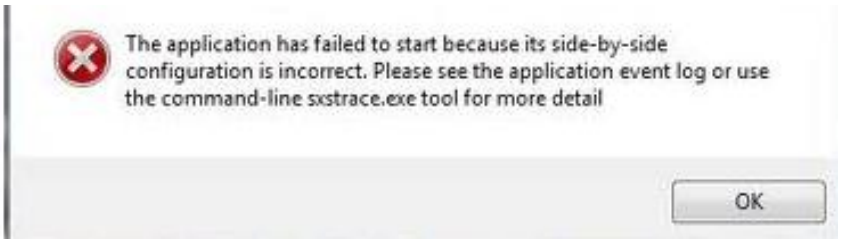

Figure 2-3: Error

To pinpoint this issue, we used dependency walker [8] to analyze the built application.

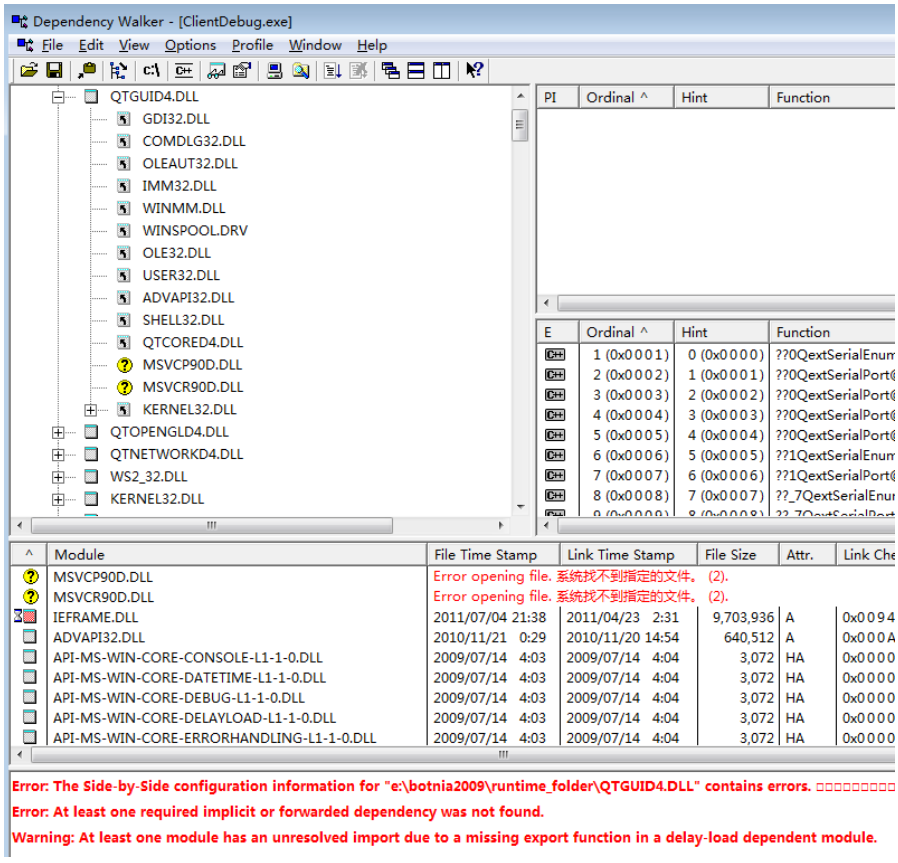

Figure 2-4: Analyze result of dependency walker

As we can see in the Figure 2-4, the QT library depends on some visual C++ runtime libraries, which are lost in some other computers. And that is why we cannot run the application.

To understand this issue, we need to go deeper beneath the error message itself: First of all, we used "Qt libraries 4.7.3 for Windows (VS 2008)" in the old version, which means the QT libraries have been pre-built via Visual Studio 2008. And in VS2008, its linker will use version 9 of Microsoft Visual C++ Runtime Libraries as a sub-library. However, the application is built using Visual Studio 2010, so the version 10 of the libraries above will be also become a requirement. To resolve this problem, we have to install both libraries on the target machine. But even we solve it like this; it is obviously a bad configuration that we want to avoid.

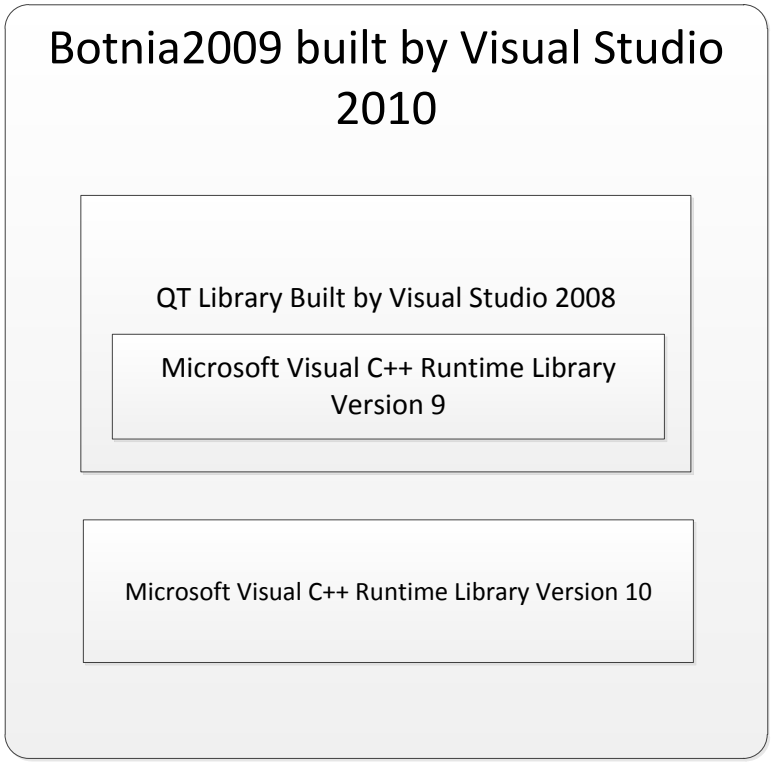

Figure 2-5: The dependency of executable built using the old procedure

The best way to eliminate this problem is to rebuild QT's library in Visual C++ 2010 environment. To achieve this, first we have to download the source code of QT 4.7.3 (http://get.qt.nokia.com/qt/source/qt-everywhere-opensource-src-4.7.3.zip)

Then we unzip the file into a folder. Before building the library, we have to configure how they should be build, which could be done by running the "configure.exe" under the folder with proper parameters. According to the manual, following parameters are used:

(1) –debug-and-release: Both the debug version and release version of QT libraries will be built

(2) –opensource: choose the LGPL license.

(3) –platform win32-msvc2010: Using MSVC++ compiler, version 2010, to build the QT libraries

(4) –mp: This is optional. It means we will use multi-processor compilation in building the libraries. In common situation, the building process will take several hours. With this option, we can dramatically decrease the compilation time.

After this, we can open the Visual Studio Command Prompt (2010)

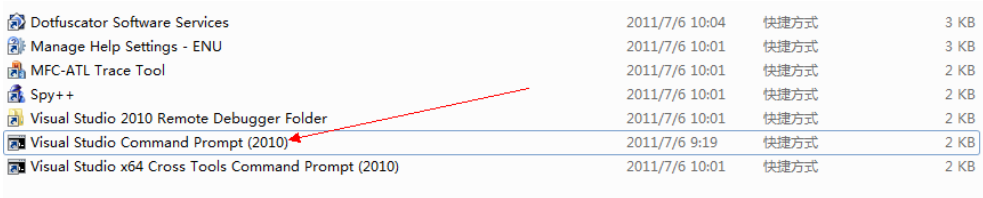

Figure 2-6: Visual Studio Command Prompt (2010)

Then type the command as Figure 2-7:

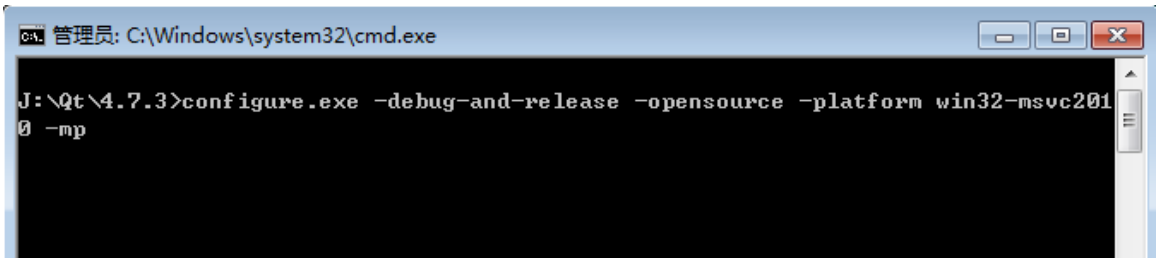

Figure 2-7: Parameters of configure.exe

When configure.exe finished its task, we can run make.exe. After a long time wait, we will get the libraries under its “lib” sub-folder.

Then we copy all the libraries we need to the project’s corresponding folder:

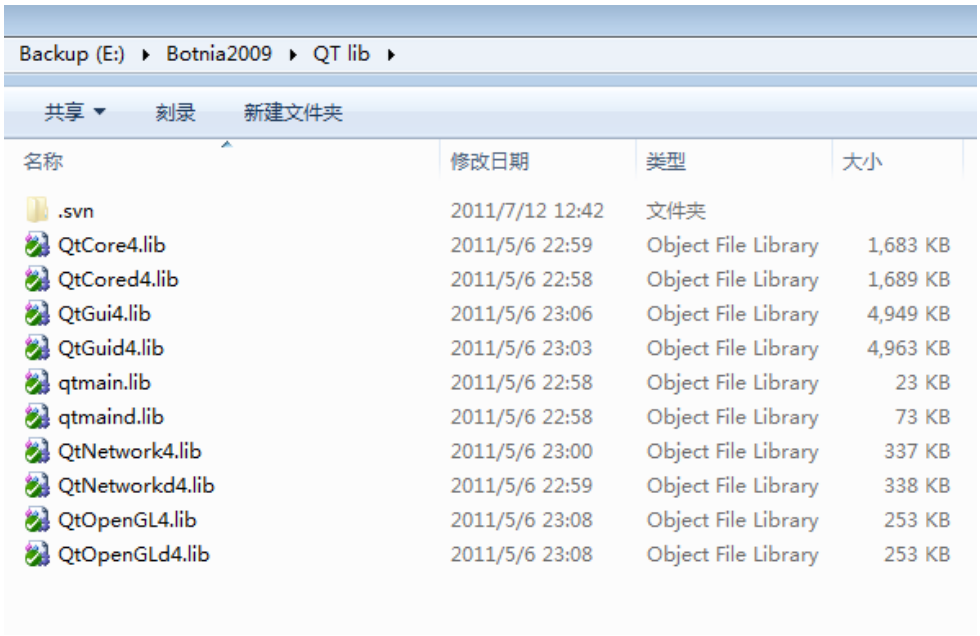

Figure 2-8: Copy the libraries to the project’s corresponding folder

Now we have everything we need to build the project.

Note: Although we need to build the libraries by ourselves, however, the “Qt libraries 4.7.3 for Windows (VS 2008)” package is still needed because we need the tool chain with its installation.

2.3 Configure the Visual Studio 2010

The project upgrade wizard will convert Visual Studio 2008 project to Visual Studio 2010 project automatically and most of the options will be imported and converted.

However, there are still some options should be specified manually.

(1) Set the “Working Directory” as Figure 2-9. Otherwise we will not be able to debug the application inside the Visual Studio since the configuration files of Botnia2009 are stored in a fixed place.

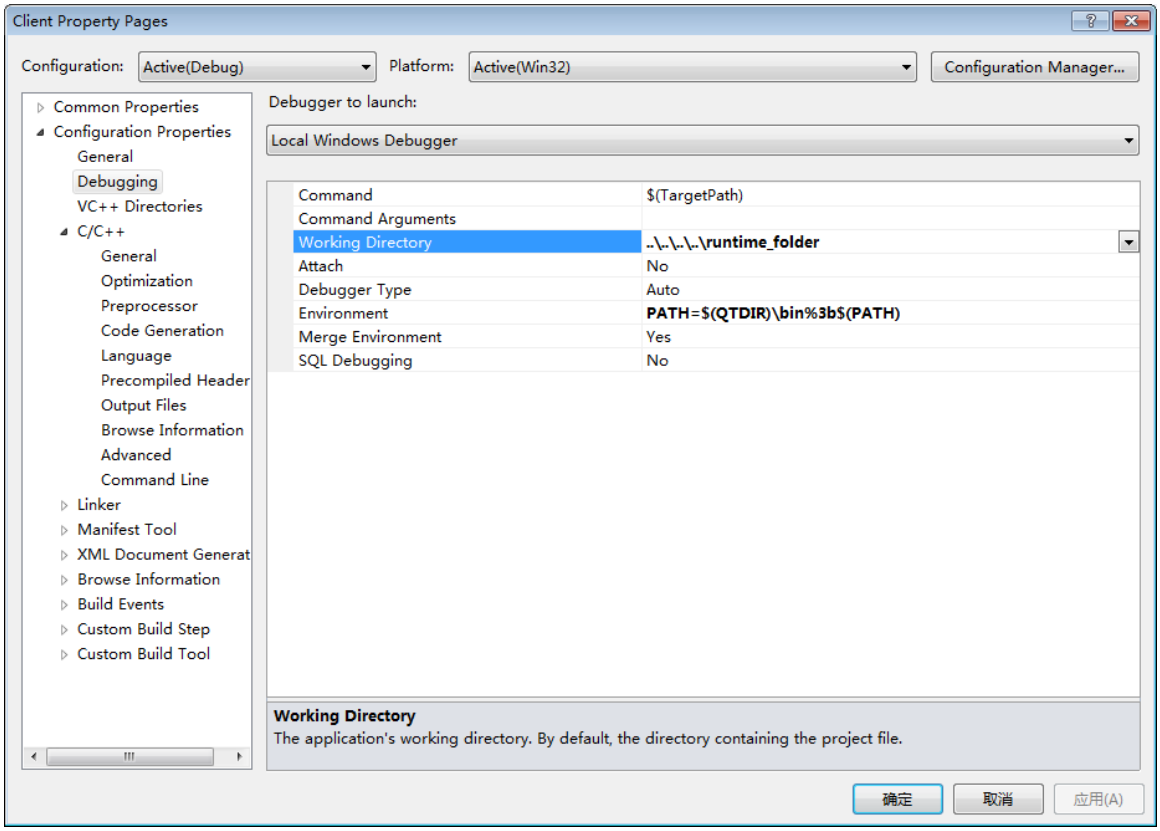

Figure 2-9

(2) Enable /mp as Figure 2-10

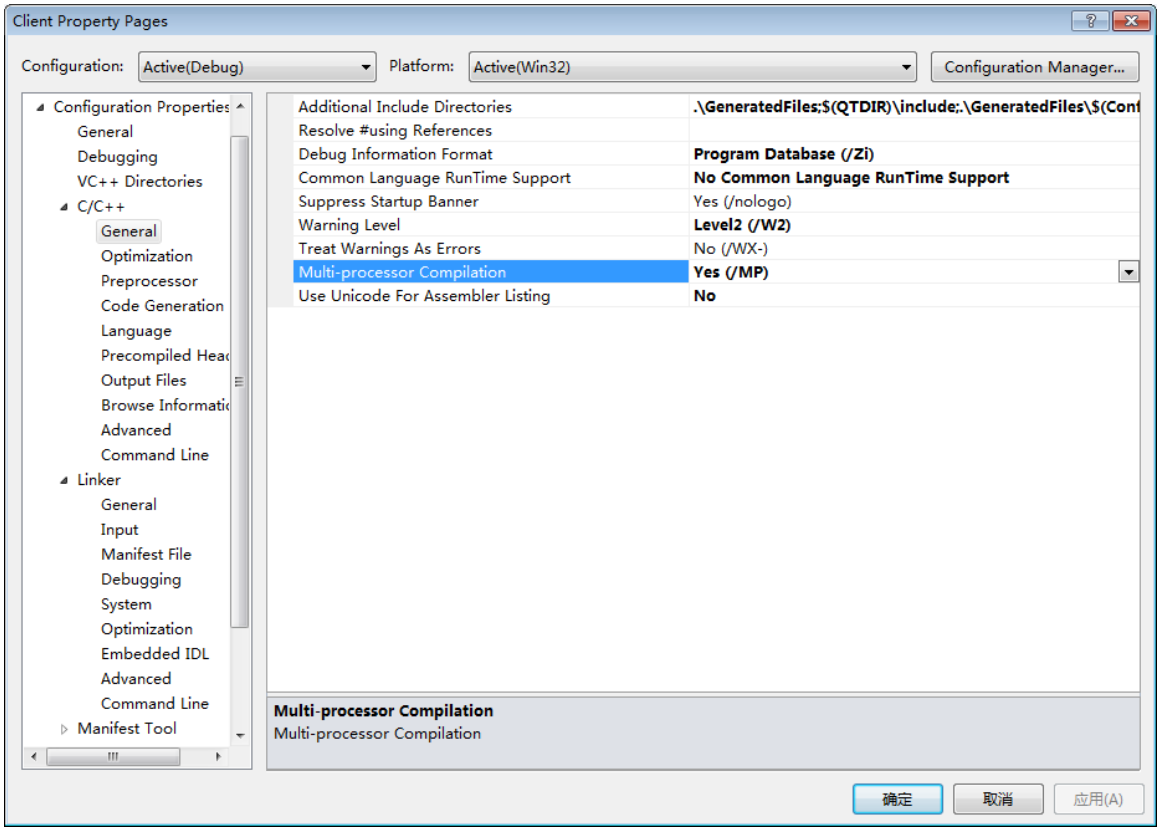

Figure 2-10: Enable Multi-processor Compilation

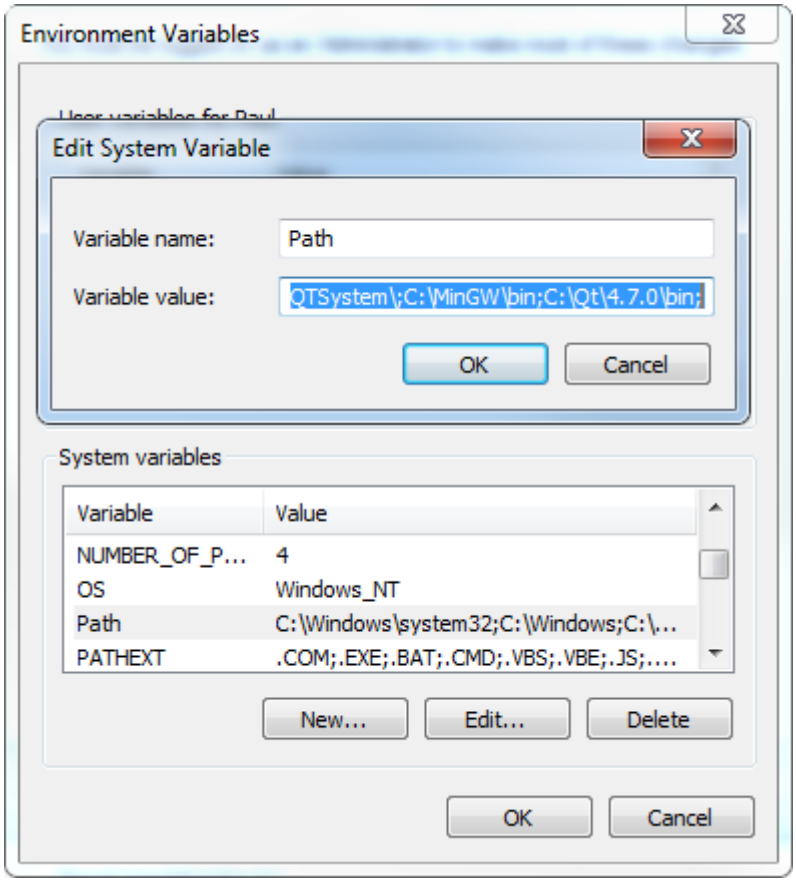

Figure 2-11: Check the PATH

Before we build, we should check the directories that are listed under PATH variables, if we find any old version of cl.exe, we should rename or remove it. Otherwise it will cause a compilation error (Figure 2-12). This is caused by a known bug of Visual Studio 2010 SP1, since it will try to find the compiler using the PATH system variables

```
1>------ Build started: Project: test, Configuration: Debug Win32 ------
1> Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 12.00.8168 for 80x86
1> Copyright (C) Microsoft Corp 1984-1998. All rights reserved.
1>
1> cl    ?
1>
1>CL : Command line warning D4024: unrecognized source file type '    ?', object file
assumed
1> Microsoft (R) Incremental Linker Version 10.00.40219.01
1> Copyright (C) Microsoft Corporation. All rights reserved.
1>
1> /out:
1> ?.exe
1>LINK : fatal error LNK1146: no argument specified with option '/out:'
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
```

Figure 2-12: Possible error during the compilation if we did not deal with that old version of cl.exe under the PATH directories

After the steps above, we can start building the project.

If we use the dependency walker to check the built application, we can see that all the libraries and the executable itself are depending on the version 10 of MSVC Library (Figure 2-13)

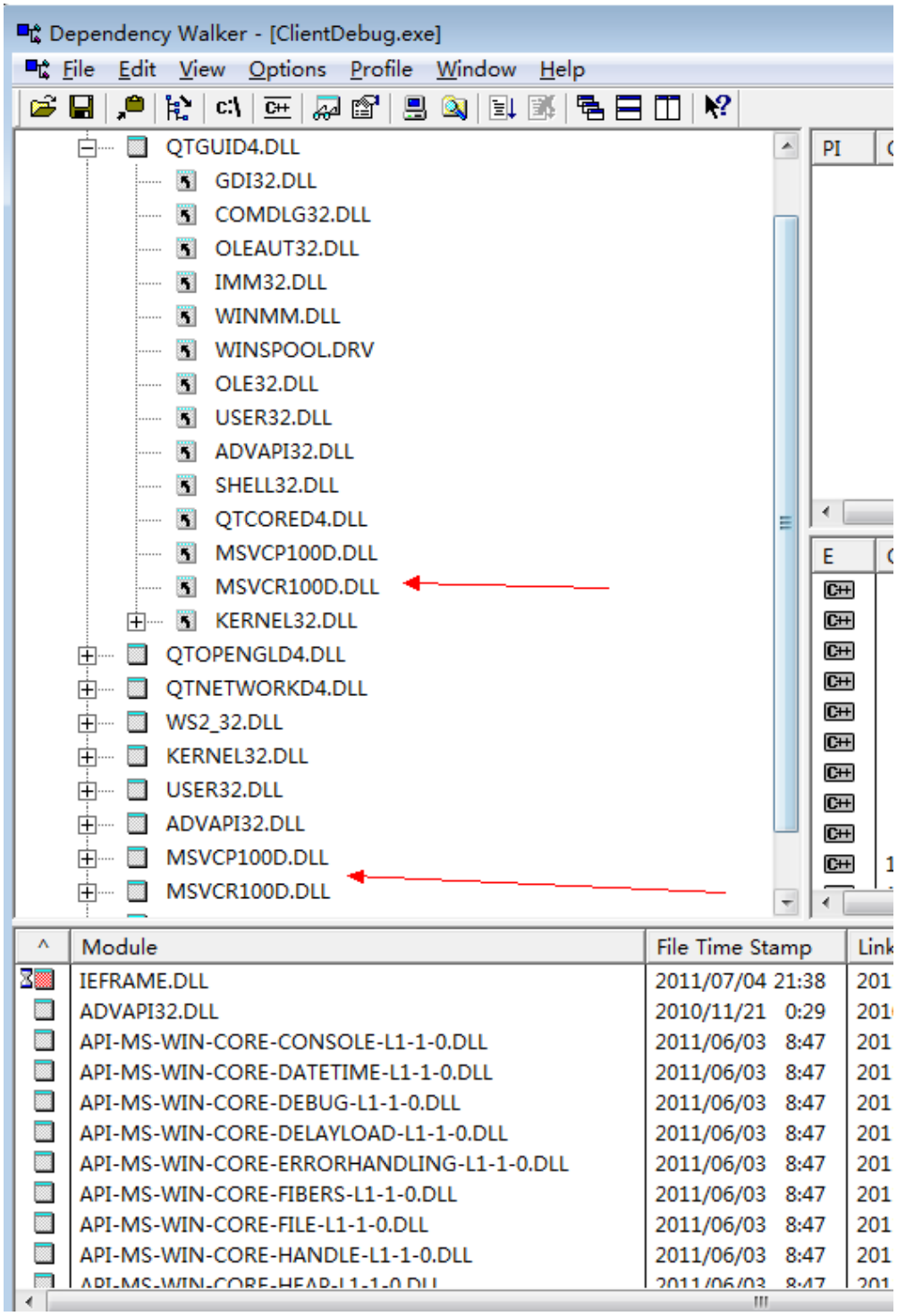

Figure 2-13

2.4 Update Google protobuf

Currently there are 4 protobuf source files in the SSL-Vision project, which should be compiled and included in the Botnia2009 project.

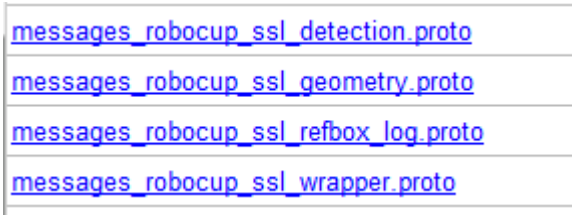

Figure 2-14: protobuf source files of SSL Vision

By using the new version of the "protoc" compiler on the Google protobuf's official website to compile the sources, we can get the new version of source files.

```
#if GOOGLE_PROTOBUF_VERSION < 2003000
#error This file was generated by a newer version of protoc
which is
#error incompatible with your Protocol Buffer headers.
Please update
#error your headers.
#endif
#if 2003000 < GOOGLE_PROTOBUF_MIN_PROTOC_VERSION
#error This file was generated by an older version of protoc
which is
#error incompatible with your Protocol Buffer headers.
Please
#error regenerate this file with a newer version of protoc.
#endif
```

From the code we can notice a number "2003000", this means the output C++ header and its corresponding source file are built by the "protoc" compiler version 2.3.0

However, there are several files in the project (Figure 2-15), who's "proto" source cannot be found in the latest version of SSL-Vision (Listed in Figure 2-14)

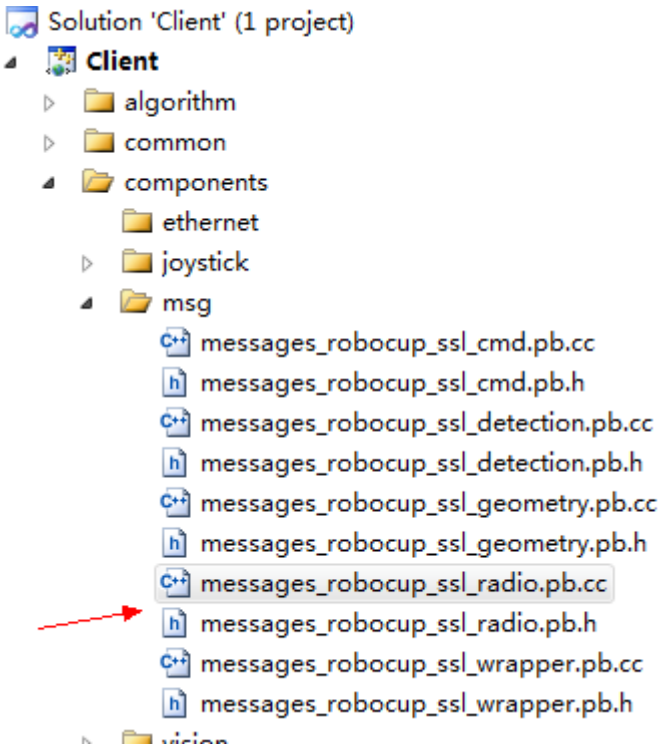

Figure 2-15

Since there is no tool to reverse these files, we cannot convert them directly. However, by reading the compiled C++ sources carefully, we can find the "proto" sources are generated as comments in the headers (Figure 2-26)

Also, the name after the "message" keyword in the ".proto" is the same as the class name in the source (Figure 2-17). After we find this information, we can recover the original protobuf source file by ourselves.

Thus, what we do is copy them to a plain text file and rename their extensions to ".proto", then using the protobuf compiler to compile them like below:

```
Protoc (filepath)(filename) --cpp_out=(Outdir)
```

For example, if your proto source is robot.proto, you can switch to its containing folder under console, and type:

```
Protoc .\robot.proto --cpp_out=.
```

And the compiled C++ files will be placed under the same folder.

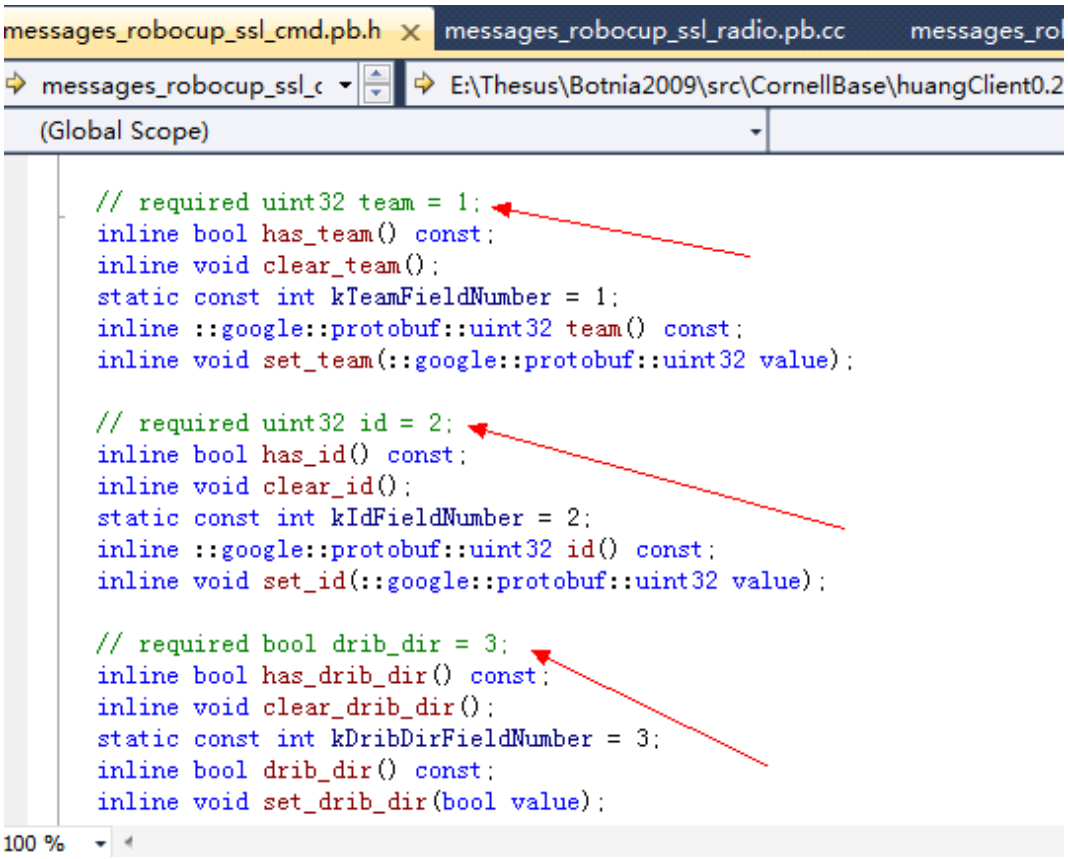

Figure 2-16

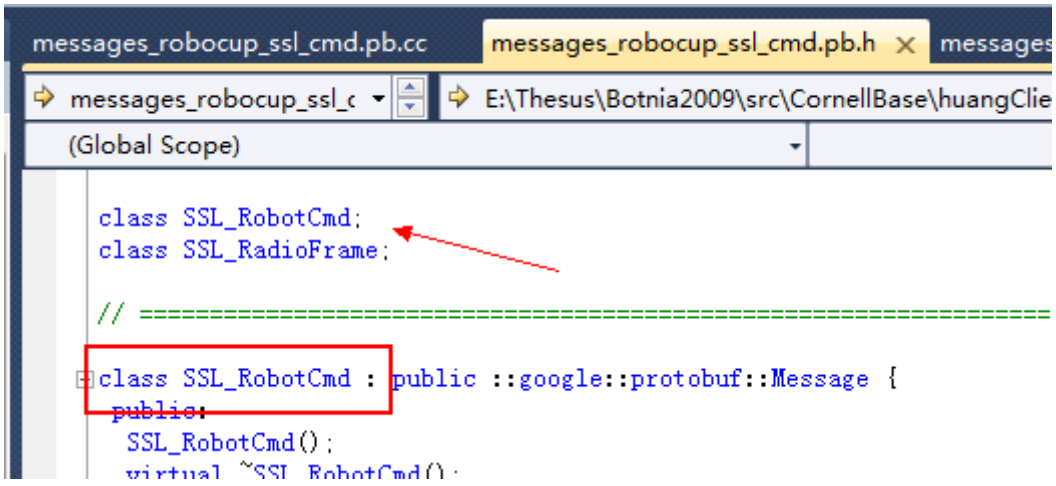

Figure 2-17

After updating these C++ files, we should also update the library files of protobuf. Because they had been previously included in the project, we can just checkout

the source of the latest protobuf, open its solution file under “vsprojects” subfolder.

Although there are 9 projects in the solution, we only need to build the main project (libprotobuf, Figure 2-18). After the build process we can get the library file in the output folder

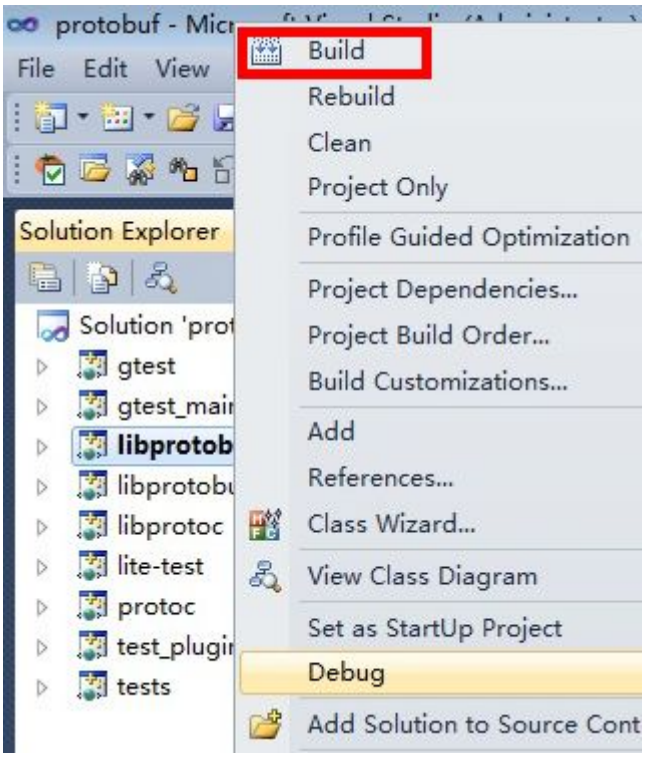

Figure 2-18

Then we can overwrite the original protobuf libraries (DEBUG version and RELEASE version) file in the project with the new one. Note that we should also build the DEBUG and RELEASE version of the library using the different configuration provided by the solution, which means we should not replace the original DEBUG version of the library with the newly built RELEASE version, vice versa.

The last step in this selection is to replace all the old Google protobuf headers in our Botnia2009 project. This step is quite simple since what we have to do is only:

(1) Delete everything under:

"Botnia2009\src\CornellBase\huangClient0.23.1.1\Client\google" in our project folder

(2) Copy everything under " src\google" of the recently checked-out Google protobuf source to the folder in the step (1) (Figure 2-19)

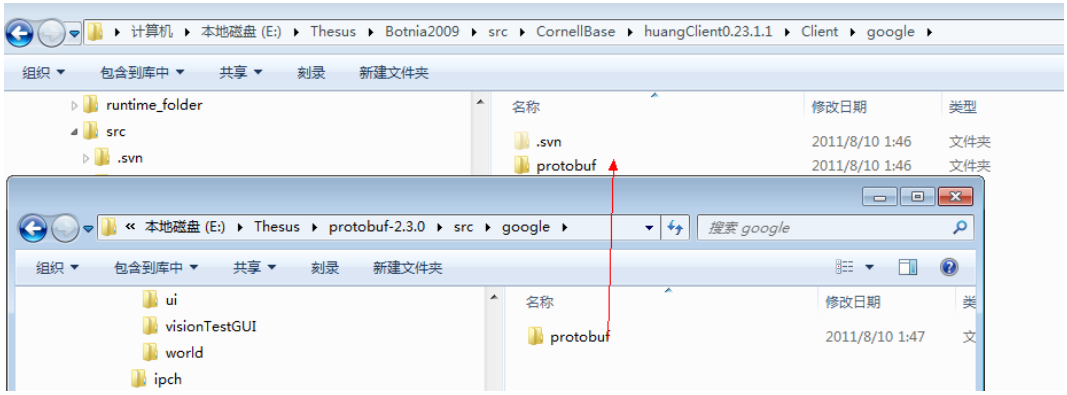

Figure 2-19: Copy the new Google protobuf's headers

3. INTERFACE IMPROVEMENT

3.1 Requirements Analyses

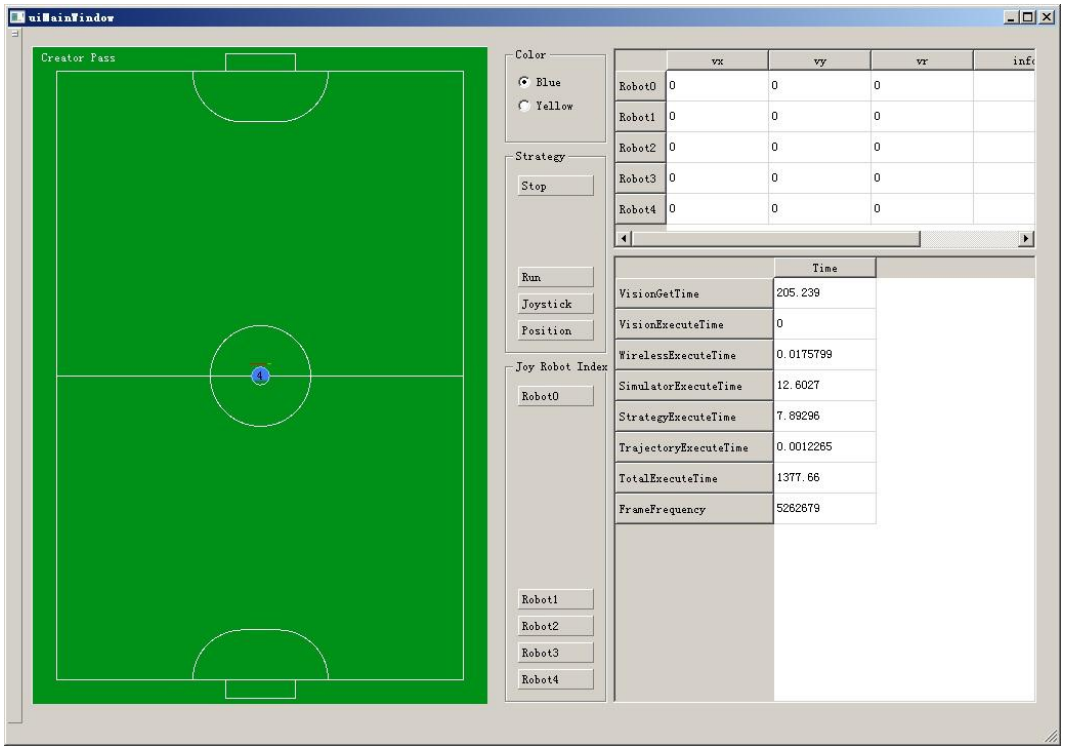

Figure 3-1: Original Botnia2009 Strategy application

Figure 3-1 shown the user interface of the original strategy system, which is made up by three modules:

(1) Field View: Shows the field, the location of each robots and ball, in 2-dimension.

(2) Control Panel: Located at the middle of the UI, which is used to control the system

(3) Status Panel: Shows the external and internal values of the whole system

The defect of the original system is obviously. First of all, some external applications must be pre-configured before running this system. For example, the referee system should be installed with GTK runtime beforehand; otherwise the strategy

system will not work. On the other hand, the joystick function cannot be tested if no joysticks connected to the host PC.

Secondly, there are too few controls available in the control panel. What we can only adjust is the status of the system, the robot which will be controlled by the joystick, and the side we are currently acts. We cannot adjust the kicking power, change the output port number, and test different strategies.

What's more, as the configurations are stored in the TXT files, when some parameters are needed to be changed, we have to edit them by a text editor, and modify them in a specified format, which is not convenient at all.

Last but not least, it is not possible to get any debug information via the UI --- the amount of status value is so few that we cannot observe some hidden errors, tracking the outgoing packets and individual events.

Thus, the main goals in this selection are decided as below:

(1) An internal referee system simulator has to be implemented.

(2) A visualized joy-pad has to be implemented on the UI.

(3) Since we have to use serial ports to send the commands, the port properties should be configurable on the UI.

(4) More information should be shown on the UI

(5) If we want to modify the application parameters, we have to use an external text editor. An internal parameter editor should be implemented.

3.2 Internal referee system simulator

Although there are so many commands supported in the official referee system, only half of them will directly influence the action of our robots and be must needs in the emulator, which includes:

(1) Halt

(2) Stop Game

(3) Force Start

(4) Normal Start

(5) Kick-off Yellow

(6) Penalty Yellow

(7) Free-Kick Yellow

(8) Indirect free-kick Yellow

(9) Kick-off Blue

(10) Penalty Blue

(11) Free-Kick Blue

(12) Indirect free-kick Blue

Inside Botnia2009, the data structure of referee system is designed like this:

```
struct REFBOX_CMD
{
      char Cmd; //Command
      char Count;
      char GoalsYellow;
      char GoalsBlue;
      unsigned short TimeLeft;
      Pair placeKickPos; //2D coordinate class
};

extern REFBOX_CMD RefboxCmd;
extern REFBOX_CMD lastRefboxCmd;
```

A thread called “RefBoxReceiveThread” is used to listening UDP packets from a multicast address, which is registered by the referee system. When new command is available, it will store the new information into the global variable “Refbox-Cmd”, while backup the old command to “lastRefboxCmd” so we can check if duplicate messages are received.

The idea of the internal referee system simulator is to add controls, which can make the application to bypass the packet capture process, while modify the “Ref-boxCmd” and “lastRefboxCmd” in case we need. This design will provide both maximum compatible with the official referee system during the normal competi-tion and the flexibility when we want to test strategy in certain cases, for example: free-kick defense.

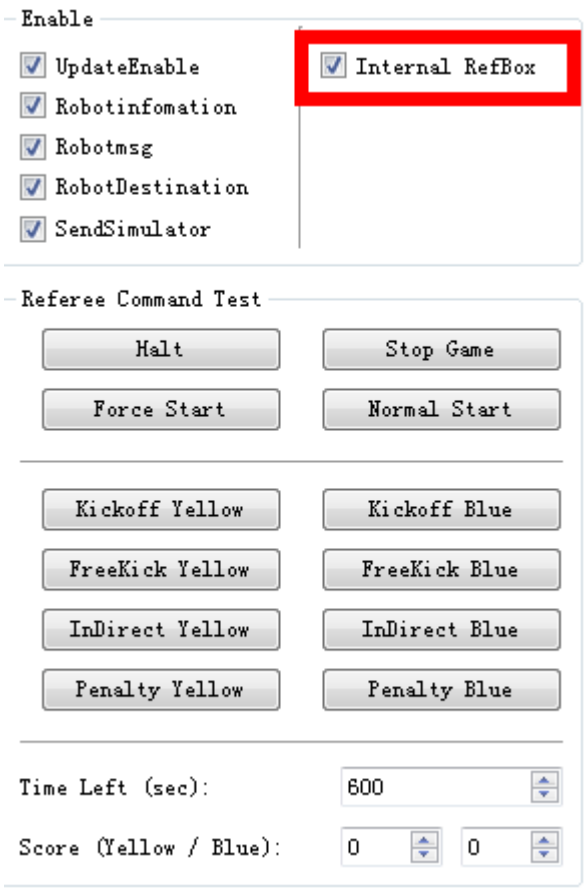

Figure 3-2: Internal referee system simulator

Figure 3-2 shows the UI implementation of this simulator. The marked checkbox (Internal RefBox) controls the availability of the simulator. When it is unchecked, the “Referee Command Test” group-box will be disabled:

```
void WInformation::slot_chkboxEnableIntRefChanged(int state)
{
	switch(state)
	{
	case Qt::Checked:
		internalRefCount = 255;
		ui.gBoxIntRef->setEnabled(true);
		break;
	case Qt::Unchecked:
		internalRefCount = 0;
		ui.gBoxIntRef->setEnabled(false);
		break;
	}
}
```

And the application will listen to the address of the external referee system, vice versa.

```
if (!ui. checkBoxInteralRef->isChecked())
{
	//Listen for external referee system messages
	count=client.receive (buf, bufsize);
}
```

The commands are sending by clicking the relevant button inside the “Referee Command Test” group-box. Below is a sample event handler when we want to send “Halt (H)” command:

```
void WInformation::slot_btnRefHalt()
{
	if (internalRefCount == 0) return;

	RefboxCmd.Cmd = 'H';
	RefboxCmd.Count = internalRefCount;
	RefboxCmd.GoalsBlue = ui.rctBlueScore->value();
	RefboxCmd.GoalsYellow = ui.rctYellowScore->value();
	RefboxCmd.TimeLeft = ui.rctTimeLeft->value();

	internalRefCount--;
}
```

The code above will copy the specified data to the “RefboxCmd” structure

Figure 3-3 shows a test case of indirect free-kick defending when we send a command via the referee system simulator:

Figure 3-3: Working referee system simulator test case

3.3 Serial port configurator

The commands to the robots are transmitted via serial ports. The original system uses the following settings as default:

(1) COM1

(2) 115200bps

(3) No parity

(4) 8 data bits

(5) 1 stop bit

Based on these unchangeable settings, many problems would be introduced when some settings cannot be achieved on some PC. For example, the COM ports: If COM1 have already been occupied, we have no choice but to rebuild the system. Thus, the serial settings must be changeable via the UI of our system.

Since Qt does not have native serial port support, an external library, named "QExtSerialPort", had been introduced. QextSerialPort provides needed interfaces to old fashioned serial ports for Qt-based applications.

By including this library into the project, we can create a serial port object in the code and operate it with convenience.

QExtSerialPort also contains another component, which acts as an enumerator, which can be called almost anytime during the program execution. This component can export the available serial ports to a list, which can be imported to the system. Thus, the problems related to the serial port operation can be fully solved.

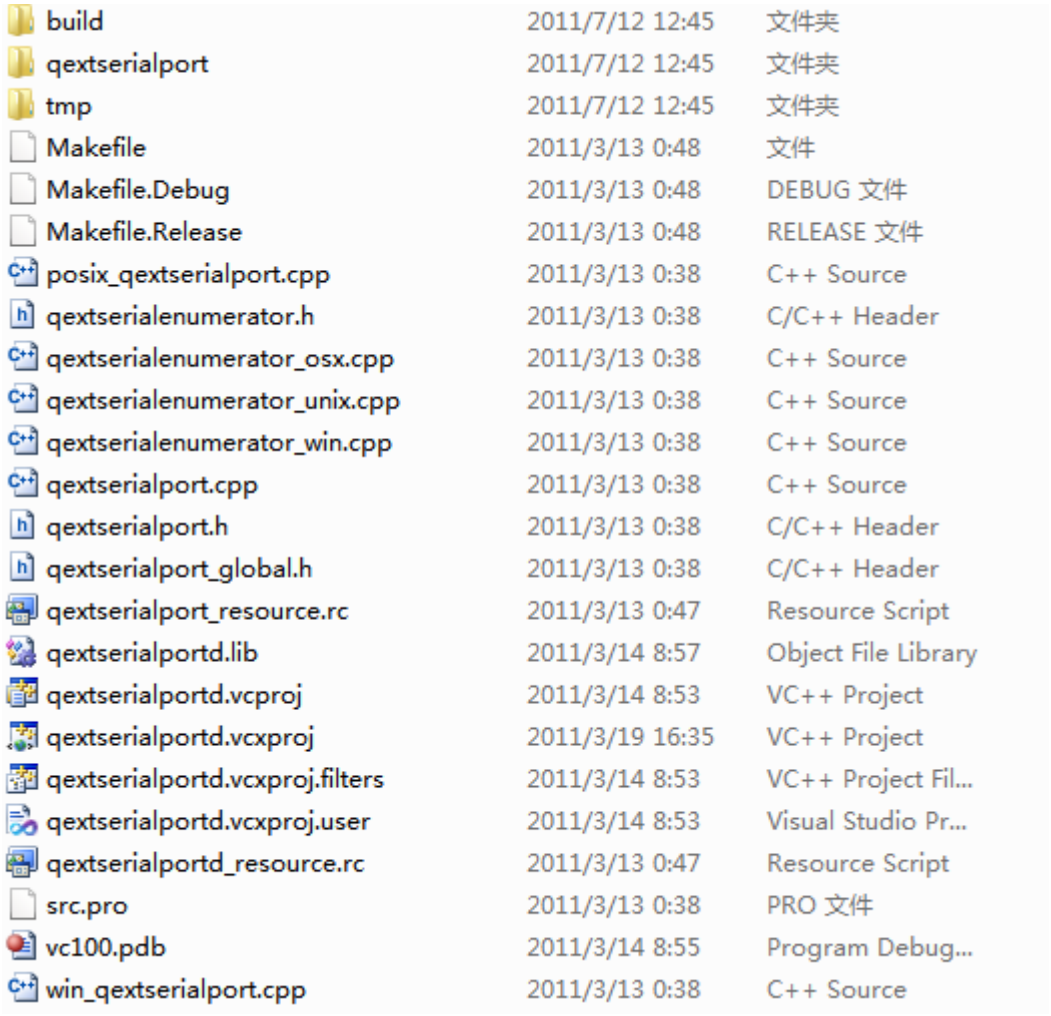

Figure 3-4: QExtSerialPort codebase

After we checked the code out, we can use VS2010 to open its solution files, which is conducted by VS 2008. It will be upgraded by the wizard of VS2010.

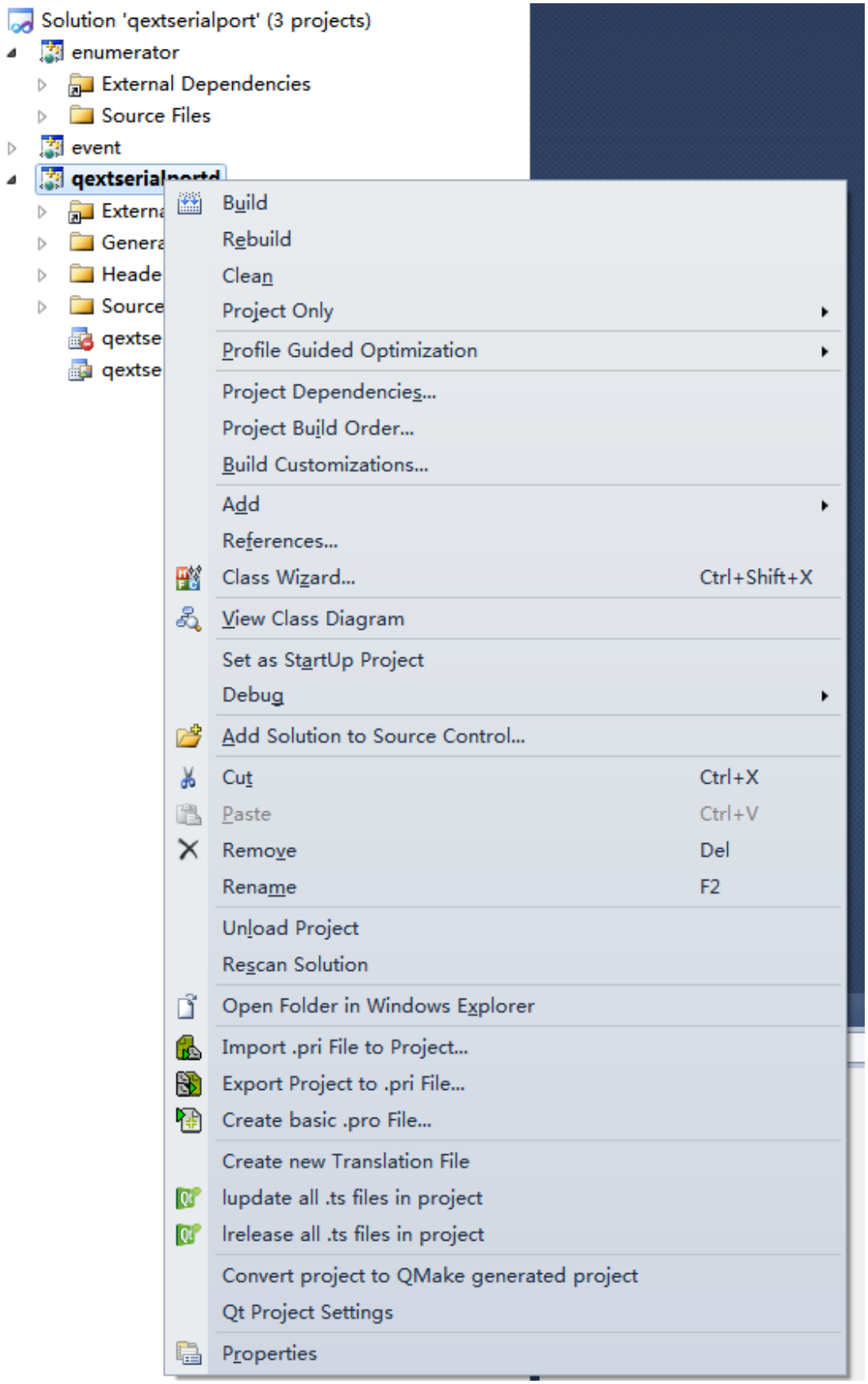

Figure 3-5: QExtSerialPort solution

There are three projects included in this solution; first of all we should adjust the properties of the main project (qextserialportd, as Figure 3-5).

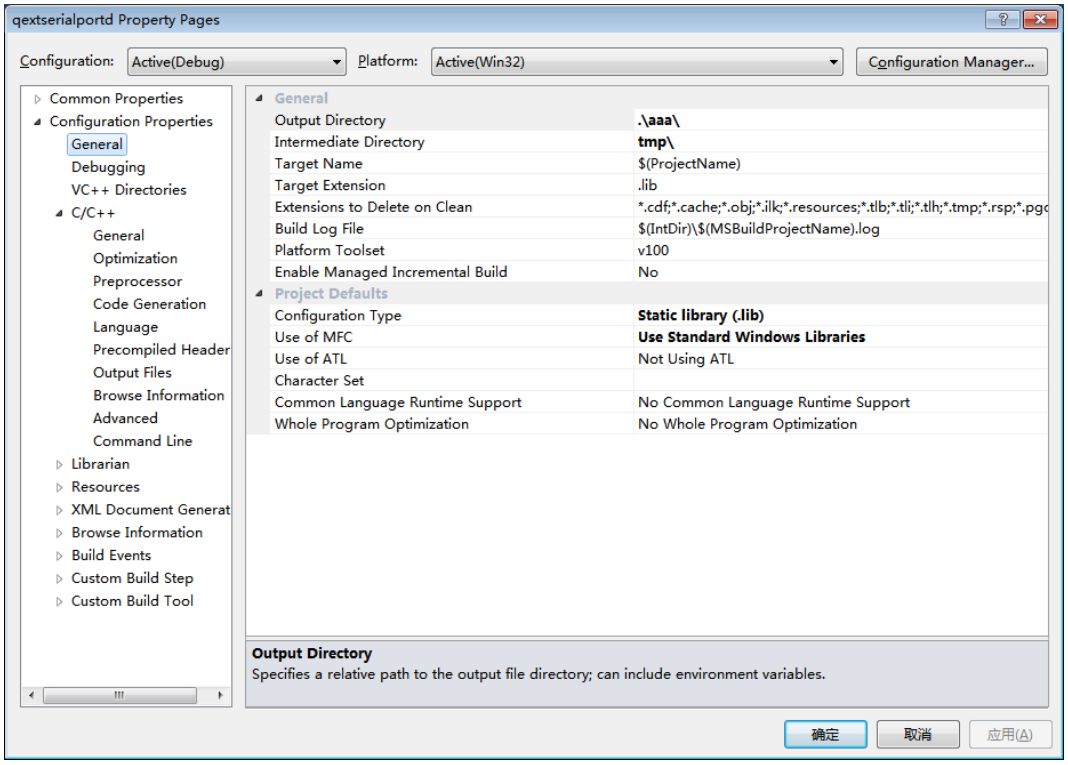

Figure 3-6

Since we need to set the output as a static library, the “Configuration Type” should be set to “Static library (.lib)” (Figure 3-6)

In the DEBUG configuration, we should set “Debug Information Format” to “/Z7” to integrate all the information to the program debug database (pdb), otherwise there might have warnings in the link step, which indicates that the debug information is missing. (Figure 3-7)

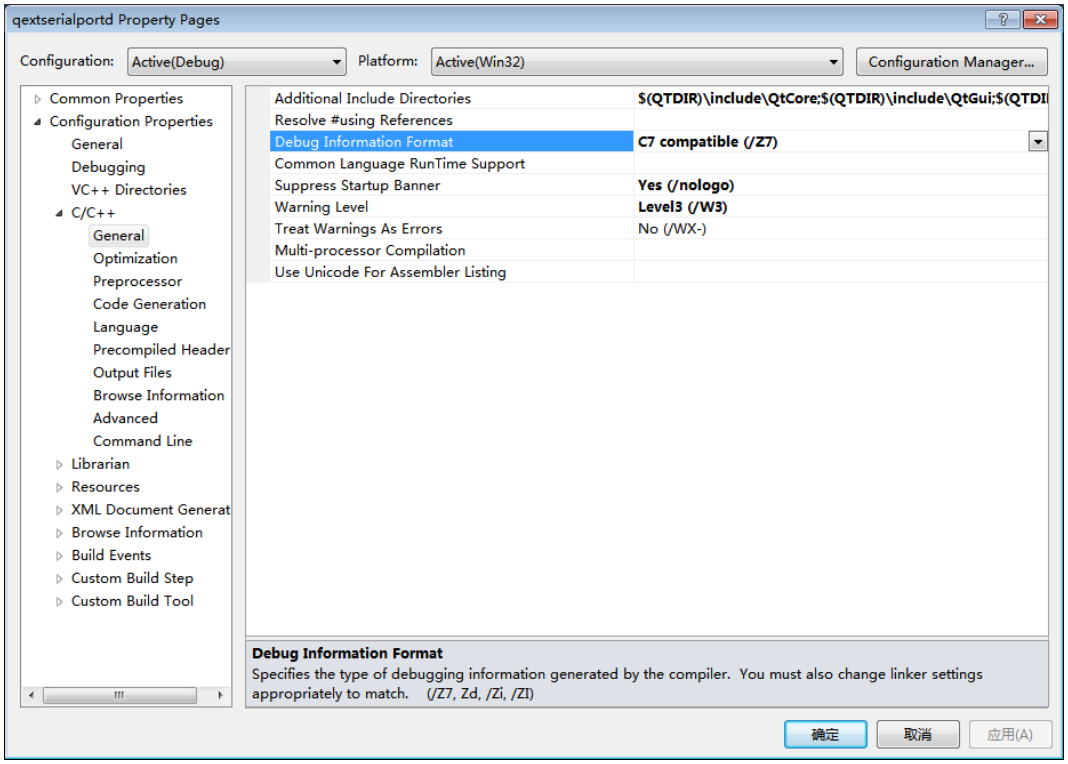

Figure 3-7

After this step, we build the debug version of the library. Then we open the output folder and the built library is there:

Figure 3-8: Built QExtSerialPort Library

Before we build the release build for the release version of Botnia2009, we should rename the library, or it will be overwritten by the latest one.

After we got the libraries, we should copy it into its library folder of Botnia2009 (\Botnia2009\src\ThirdParty\qextserial\). Before we can use it, we should include its path and name in the Botnia2009 project. (Figure 3-9, 3-10)

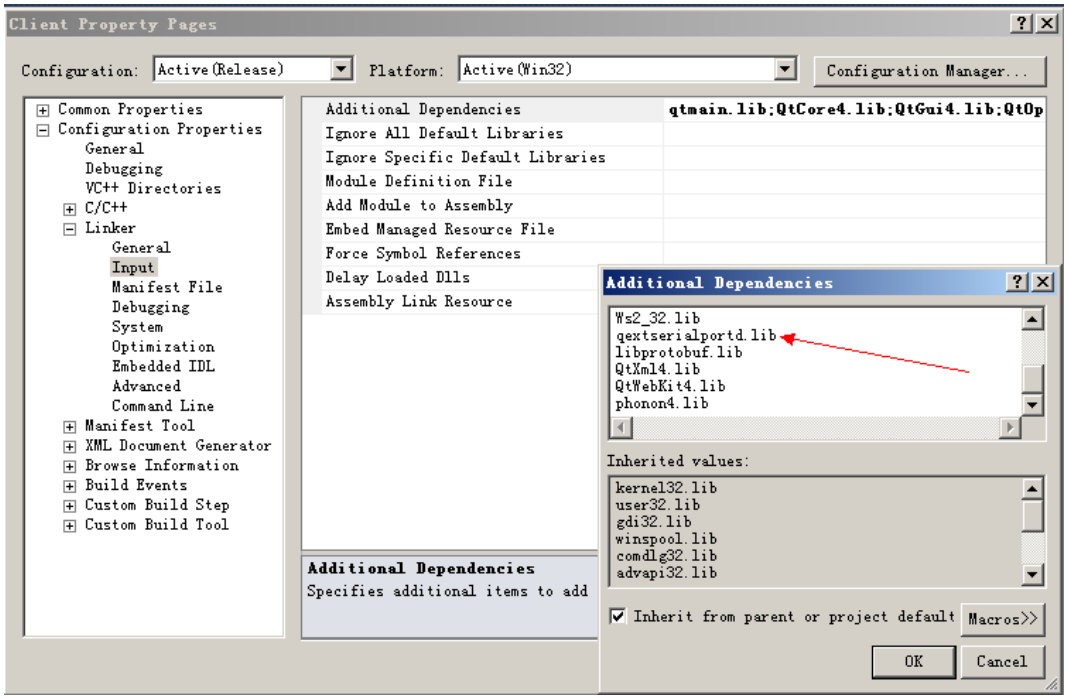

Figure 3-9: Add QExtSerialPort library to the linker settings of Botnia2009

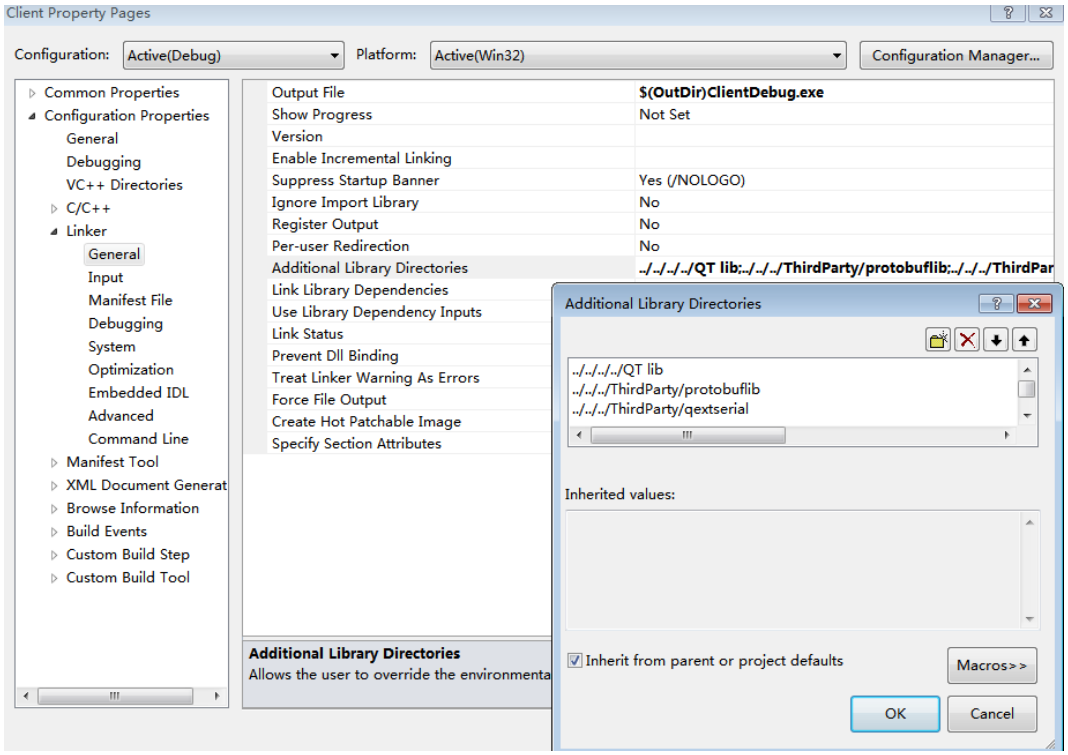

Figure 3-10: Add QExtSerialPort library to the linker settings of Botnia2009

Also, its header files should be copied to the project's folder and included in the solution explorer (Figure 3-11)

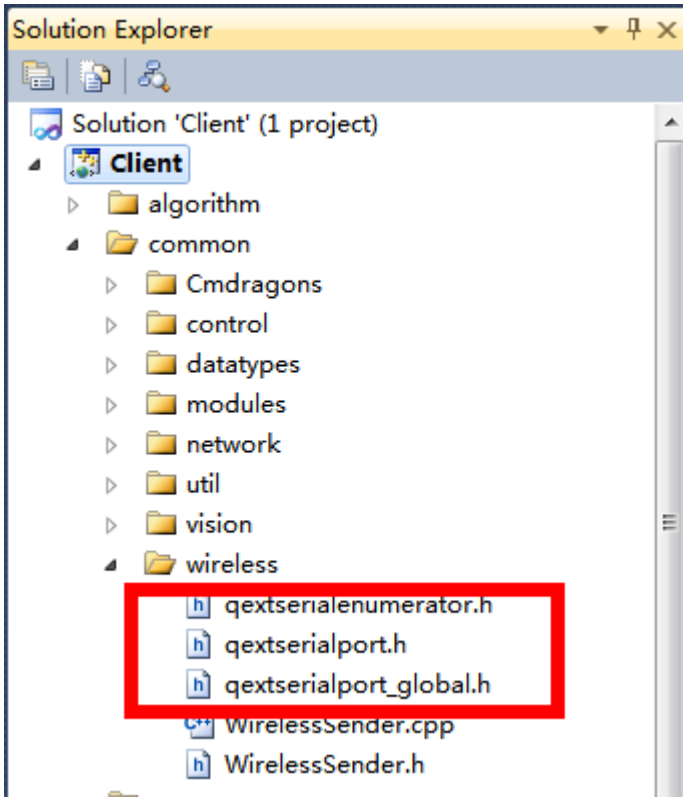

Figure 3-11

And the library is ready to use.

We can enumerate all the useable serial ports as below:

```
QStringList cbStrList;
cbStrList.clear();
QList<QextPortInfo> existPorts =
QextSerialEnumerator::getPorts();
      while (!existPorts.isEmpty()) {
            cbStrList << existPorts.first().portName;
            existPorts.pop_front();
      }
ui.cbSerPort->clear();
ui.cbSerPort->addItems(cbStrList);
ui.cbSerPort->setCurrentIndex(0);
```

We can create a serial port object (QExtSerialPort) as below:

```
QextSerialPort* serialComm = new QextSerialPort();
serialComm->setQueryMode(QextSerialPort::EventDriven);
serialComm->setDataBits(DATA_8); //8 bits data
serialComm->setStopBits(STOP_1); //1 stop bit
serialComm->setFlowControl(FLOW_OFF);
serialComm->setParity(PAR_NONE); //No parity
serialComm->setBaudRate(BAUD115200); //115200 bps
```

We can use QT's designer to add controls as Figure 3-12

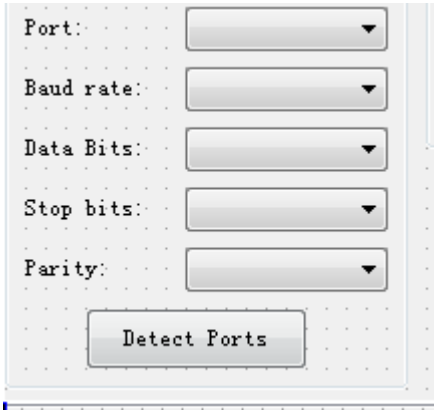

Figure 3-12: Design the serial port settings

Then we can add entities into the combo-boxes using the code below:

```
cbStrList.clear();
cbStrList << "300" << "600" << "1200" << "2400" << "4800" <<
"9600" << "19200" << "38400" << "56000" << "57600" <<
"115200";

ui.cbSerBaud->addItems(cbStrList);
ui.cbSerBaud->setCurrentIndex(10);

cbStrList.clear();
cbStrList << "6" << "7" << "8";
ui.cbSerDataBits->addItems(cbStrList);
ui.cbSerDataBits->setCurrentIndex(2);

cbStrList.clear();
cbStrList << "1" << "2";
ui.cbSerStopBits->addItems(cbStrList);
ui.cbSerStopBits->setCurrentIndex(0);

cbStrList.clear();
cbStrList << "NONE" << "ODD" << "EVEN";
ui.cbSerParity->addItems(cbStrList);
ui.cbSerParity->setCurrentIndex(0);
```

And it is finally done as Figure 3-13:

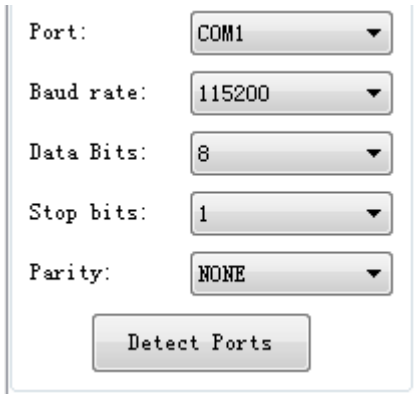

Figure 3-13

3.4 Parameter editor

The required parameters, which had been stored in plain text format, are put inside different text files under “Botnia2009\runtime_folder\Params”.

Below is a part of a parameter file

```
####################
### Field Params ###
####################

OUR_GOAL_LINE -3.025
THEIR_GOAL_LINE 3.025

# pretty much the same as OUR_GOAL_LINE, but used for goalie
OUR_LEFT_GOAL_WALL -3.025

OUR_RIGHT_GOAL_WALL -3.025
THEIR_LEFT_GOAL_WALL 3.025
THEIR_RIGHT_GOAL_WALL 3.025
RIGHT_SIDE_LINE -2.025
RIGHT_GOALIE_BOX -0.675
RIGHT_GOAL_POST -0.35000
LEFT_GOAL_POST 0.350
LEFT_GOALIE_BOX 0.675
LEFT_SIDE_LINE 2.025
GOAL_WIDTH 0.700000
```

The lines start with “#” and blank lines are considered as comments to be ignored. Each parameter is written in certain format, which can be concluded as:

parameter_name parameter_value

So we can define a structure for parameters as below:

```
struct paramStruct {
      QString pName;
      QString pValue;
};
```

And design the UI elements as in Figure 3-14:

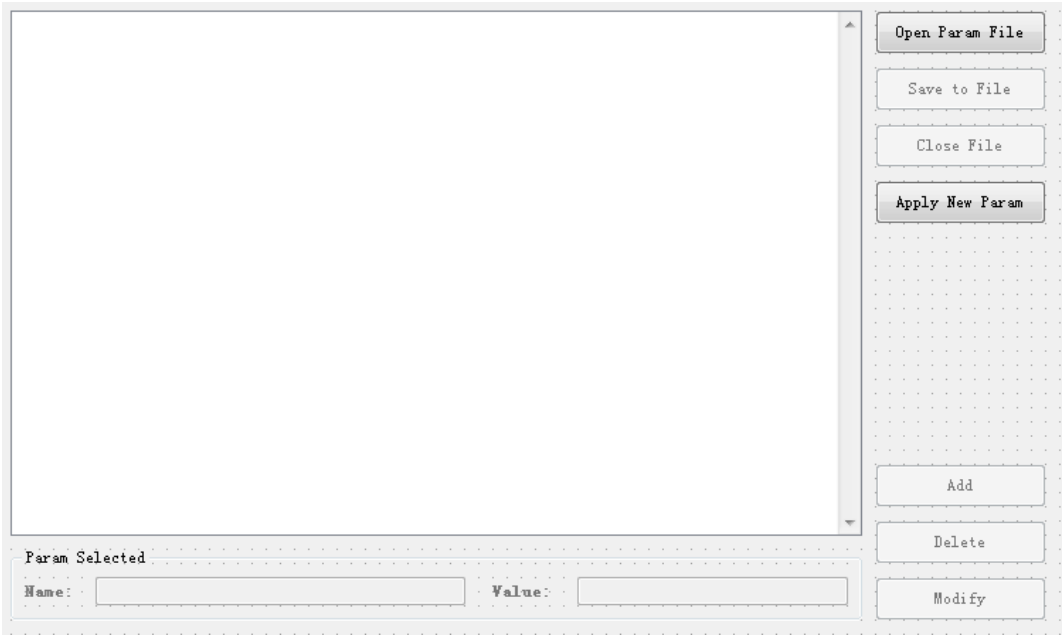

Figure 3-14: UI of parameter editor

To read the parameters into the widget, we have to read the file and put the content into a buffer.

```
paramFile.setFileName(paramFileName);

if(!paramFile.open(QIODevice::ReadOnly|QIODevice::Text))
return;

QTextStream ts(&paramFile);
paramFileBuffer=ts.readAll();

if (paramFileBuffer.simplified().isEmpty()) return;
wrapParam(); //put the content into buffers

//Enable controls
ui.btnParamSave->setEnabled(true);
ui.btnParamClose->setEnabled(true);
ui.gbParamEdit->setEnabled(true);
ui.btnParamInsert->setEnabled(true);
ui.btnParamDelete->setEnabled(true);
ui.btnParamModify->setEnabled(true);
```

The buffer is constructed by a vector. A String-list is used to store the parameters by line.

```
paramStruct ps;
QVector<paramStruct> paramVector;

/*
To ensure that there will be a blank line at the end of
the file
*/
if (!paramFileBuffer.endsWith('\n'))
paramFileBuffer.append('\n');

/*
Split the content of the paramenter file by lines
*/
QStringList psList =
paramFileBuffer.split('\n',QString::SkipEmptyParts);

QString readBuf;
```

Then we read the file to the vector:

```
while (!psList.isEmpty()) {
      readBuf = psList.first();
      psList.removeFirst();

      //Ignore the blank and comments lines
      if (readBuf.at(0)=='\n' || readBuf.at(0)==' ' ||
readBuf.at(0)=='#') continue;

      /* Ignore those lines without space,
      Since it is impossible for them to contain any
      valid paramenters */
      spaceLoc=readBuf.indexOf(' ', tmpLoc);
      if (spaceLoc==-1) continue;

      //Locate the name of the paramenter
      ps.pName = readBuf.mid(tmpLoc, spaceLoc-tmpLoc);
      tmpLoc=spaceLoc+1;
      while (readBuf.at(tmpLoc)==' ' ||
readBuf.at(tmpLoc)=='\t') {tmpLoc++;}

      //Check if there are comments at the end of the line
      spaceLoc=readBuf.indexOf(' ', tmpLoc);
      if (spaceLoc!=-1) {
            ps.pValue = readBuf.mid(tmpLoc, spaceLoc-tmpLoc);
      } else {
            ps.pValue = readBuf.mid(tmpLoc, readBuf.size());
      }

      //Add the unit into the vector
      paramVector << ps;
      spaceLoc=tmpLoc=0;
}
```

At last, we write the contents to the QTabWidget on the UI:

We can observe the results in Figure 3-15.

```
QVector<paramStruct>::iterator i;

i=paramVector.begin();
int rowCount=0;

while (i!=paramVector.end()) {

      //Write the contents to the cells
      ui.tableWgParam->setItem(rowCount,0, new
QTableWidgetItem(i->pName));
      ui.tableWgParam->setItem(rowCount,1, new
QTableWidgetItem(i->pValue));
      i++;
      rowCount++;
}
```

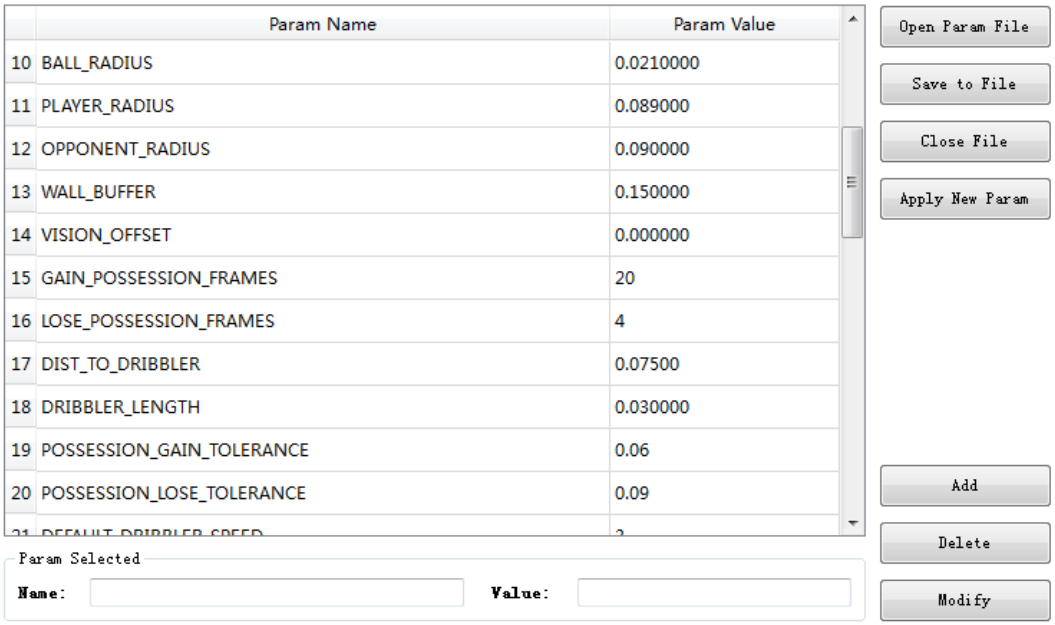

Figure 3-15: Parameter editor

Thanks to the convenience of the QVector container, inserting, modifying and deleting of the elements is very simple --- just use the given APIs to change the vector then reload the UI and everything should be OK.

```
// Insertion of new element
void WInformation::slot_btnParamInsert()
{
      paramFileBuffer.append(ui.leParamName->text() + " " +
ui.leParamValue->text());

      //Update the vector and refresh the widget on the UI
      wrapParam();
}
```

To modifying and deleting the contents in the specified cells, we have to get the location of the elements.

This can be achieved by dealing with the “cellDoubleClicked” signal of QTabWidget. The location of the selected parameter will be given as parameters in its slot.

```
// Modifying
//Connect the signal to the slot
QObject::connect(ui. tableWgParam, SIGNAL(cellDoubleClicked
(int, int)), this, SLOT(slot_tableWgParamDoubleClicked(int,
int)));

void WInformation::slot_tableWgParamDoubleClicked(int row, int
col)
{

      //Replace operations
      ……

      //Update the vector and refresh the widget on the UI
      wrapParam();
}
```

3.5 Visualized Joy-pad (VJP)

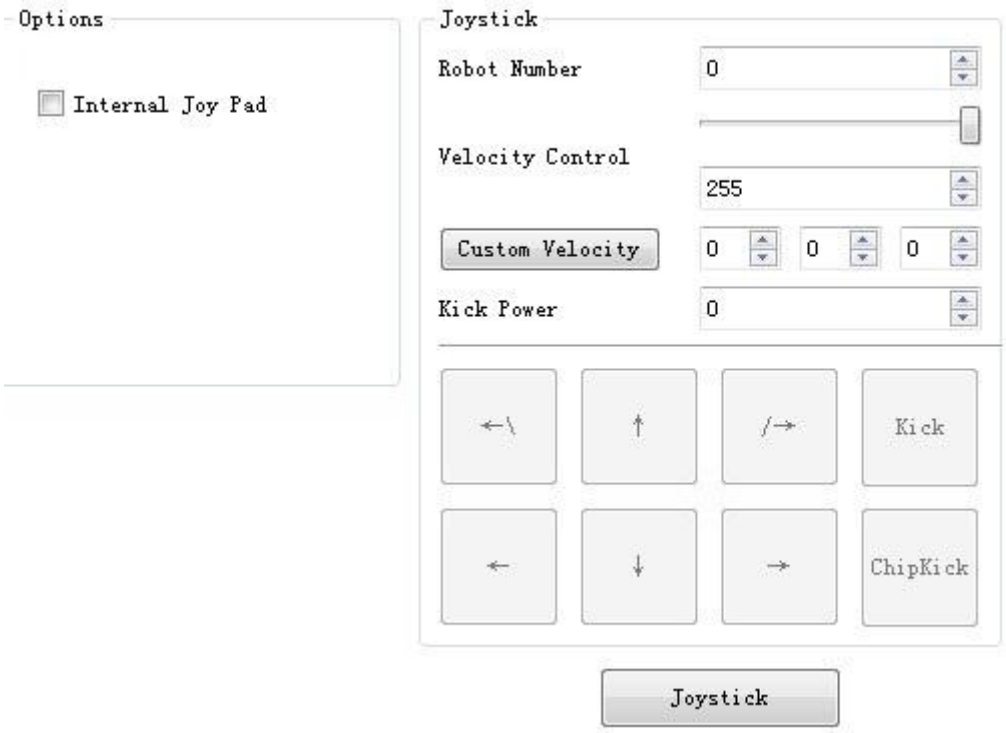

Figure 3-16: UI elements of Visualized Joy-pad

Figure 3-16 illustrates the design of the VJP. Each joy-pad can gain the control of one robot, and the maximum velocity and kick power can be confined below a certain value. The VJP is controlled by a checkbox inside the "Options" group-box. When it is unchecked, the VJP will be disabled.

Also, the VJP can be used by pressing the NUMPAD on the keyboard.

The NUMPAD function can be done by implementing the "keyReleaseEvent" or "keyPressEvent" of the QWidget class. Below is the code for the "keyReleaseEvent", which is used when we release the key (Stop the input). For the "keyPressEvent", we have to change all the "false" to "true". The usage of the BOOL flag will be described later

```
if (!ui.checkBoxInteralJoyPad->isChecked()) {
      return;
}

switch (event->key()) {
       case Qt::Key_Left:
             slot_joyLeftToggled(false);
             break;
       case Qt::Key_Right:
             slot_joyRightToggled(false);
             break;
       case Qt::Key_Down:
             slot_joyDownToggled(false);
             break;
       case Qt::Key_Up:
             slot_joyUpToggled(false);
             break;
       case Qt::Key_Home:
             slot_joyTurnleftToggled(false);
             break;
       case Qt::Key_PageUp:
             slot_joyTurnrightToggled(false);
             break;
       case Qt::Key_End:
             slot_joyKickToggled(false);
             break;
       case Qt::Key_PageDown:
             slot_joyChipkickToggled(false);
             break;
}
```

The “slot_” functions in the code are actually the slot function which handles the corresponding signal triggered by the direction buttons in Figure 3-16.

```
QObject::connect(ui.btnMCleft, SIGNAL(toggled(bool)), this,
SLOT(slot_joyLeftToggled(bool)));
```

Such binding can be done by the QObject::connect function. When a specified event happens, a signal will be emitted, and the designated slot function will be called.

```
//JoyStick in externInit.h
extern bool internalJoy; //Enable bit
extern int internalJoyvx; //Velocity on the X axis
extern int internalJoyvy; //Velocity on the Y axis
extern int internalJoyva; //Rotation speed
extern int internalKickPower; //Kick power
extern bool internalJoyKick; //Normal kick trigger
extern bool internalJoyChipKick; //Chipkick trigger
```

Before introducing the code of the buttons' events, several global variables have been added for the control purposes. The slot function will change those variables directly when needed, to affect the velocity of the target robot.

Besides, those values will be read regularly by the transmitter module, to get the speed data and warp in a packet then send to the robot. The wireless module will be introduced in Chapter 6.

```
void WInformation::slot_joyUpToggled(bool checked)
{
	if (checked) {
		ui.btnMCdown->setChecked(false);
		internalJoyvx = ui.spinBoxMCVLimit->value();
	} else {
		internalJoyvx=0;
	}
}
```

The code above is the implementation of the "Up" button. The "checked" flag will indicate whether the button is pressed or released. As a virtual controller, we cannot change its speed by giving different press power to the input device (key-board), so the limitation value will be applied directly.

Since there is no sense for the button indicates the reverse direction being pressed at the same time, so pressing "Up" will release the "Down" automatically (ui.btnMCdown->setChecked(false)). Other buttons are done in the similar way.

3.6 Status Panels

A good status panel can help us understanding what is happening inside the system. Figure 3-17 is a snapshot of the newly implemented field information panel.

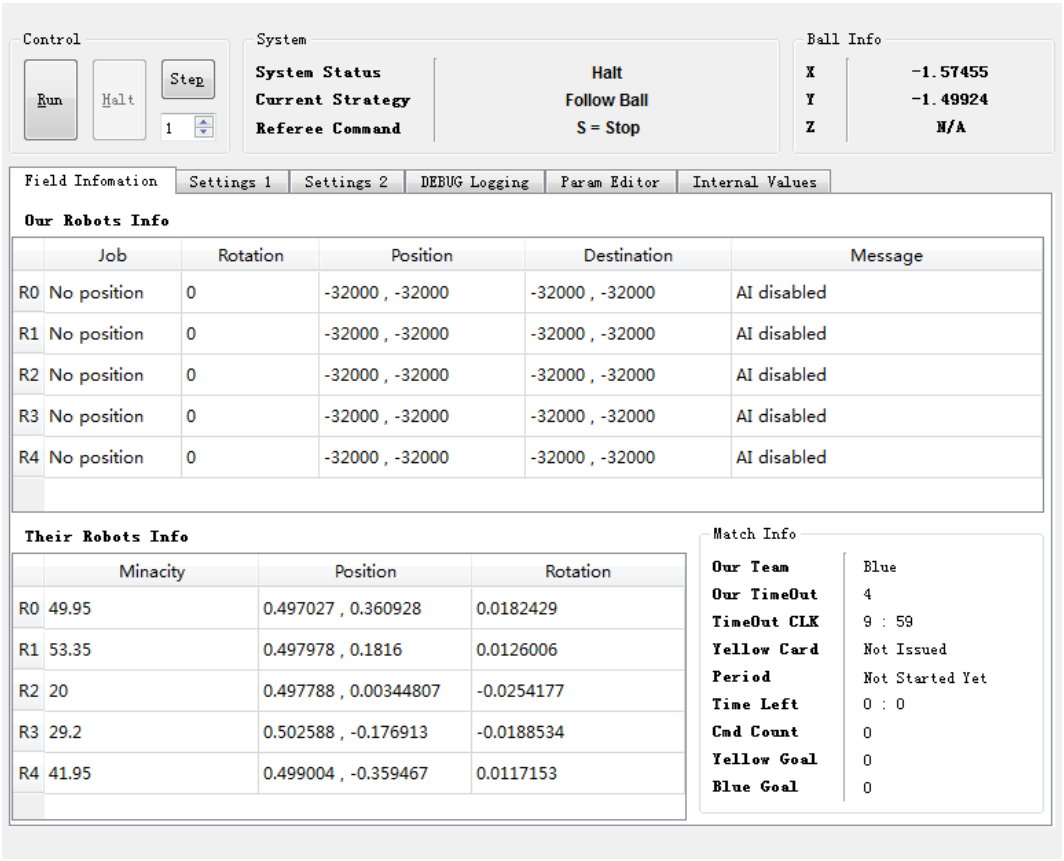

Figure 3-17: Status Panel (Field Information Tab)

The data is retrieved from the functions of "RoboCupModule", which contains the location and rotation data we need to present on the QTableWidget, which are implemented using the same technology as the Parameter Editor.

For example:

```
/* Get the horizontal ordinate of Robot i. Since QTableWidget
 and QLabel control accept only Strings, we convert the number
 to String using QString::number() */
QString::number(sm->GetOurRobotX(i))
```

The challenge came at the “Match Info” part. Since the referee system will not report the timing information of “Timeout” and “Yellow card”. So we need to record them all by the application itself.

When a “Yellow Card” command is received, we first record the time left

```
if ((RefboxCmd.Cmd == 'y' && YELLOW == sm->CurrentTeam() &&
RefboxCmd.Count != cmdCnt) || (RefboxCmd.Cmd == 'Y' && BLUE ==
sm->CurrentTeam() && RefboxCmd.Count != cmdCnt))
{
      YellowCardStart = RefboxCmd.TimeLeft;
}
```

The “YellowCardStart” variable also act as a flag, if it is not 0, we can know that the yellow card countdown is not finished yet.

```
if (YellowCardStart > 0) {
      int YCClock = 120 - (YellowCardStart -
lastRefboxCmd.TimeLeft);

      if (YCClock < 0) {
            YellowCardStart = 0;
      } else {
            ui.lYCCLK->setText(QString::number(YCClock/60) +
      " : " + QString::number(YCClock%60));
      }

      } else {
            ui.lYCCLK->setText("Not Issued");
      }
```

Since the penalty time of yellow card is 120 second, we can use the time elapsed to calculate the remaining penalty time.

The timeout indicator is done in the similar way:

```
if ((RefboxCmd.Cmd == 't' && YELLOW == sm->CurrentTeam() &&
RefboxCmd.Count != cmdCnt) || (RefboxCmd.Cmd == 'T' && BLUE ==
sm->CurrentTeam() && RefboxCmd.Count != cmdCnt))
      {
            TimeOutLeft--;
            TOstart = QDateTime::currentDateTime();
            isTO = true;
      }

if ((lastRefboxCmd.Cmd == 't' && YELLOW == sm->CurrentTeam())
|| (lastRefboxCmd.Cmd == 'T' && BLUE == sm->CurrentTeam()))
      {
            QDateTime TOend = QDateTime::currentDateTime();
            if (TOstart.secsTo(TOend) != 0) {
            if (TimeOutCLK>0 && TimeOutLeft>=0)
                  {TimeOutCLK--;}
                  TOstart = TOend;
            }
      }

      if (TimeOutCLK > 0) {
      ui.lTimeOutCLK->setText(QString::number(TimeOutCLK/60) +
      " : " + QString::number(TimeOutCLK%60));
      } else {
            ui.lTimeOutCLK->setText("Time Used Up")
      }

      if (TimeOutCLK > 0 && TimeOutLeft > 0) {
      ui.lTimeOut->setText(QString::number(TimeOutLeft));
      } else {
      ui.lTimeOut->setText("Not Allowed");
      }
```

When a “timeout” command is received, we will first check if it is caused by our team, if so, the available timeouts will be subtracted by 1.

Then the current time will be recorded, and it will be checked regularly if the time is changed. If so, the available timeout time will be reduced.

3.7 Logger

The logger is consisted by two “QPlainTextEdit” and several checkboxes and buttons. One of the text edit field is used to output logs, another is used to send commands to the robots.

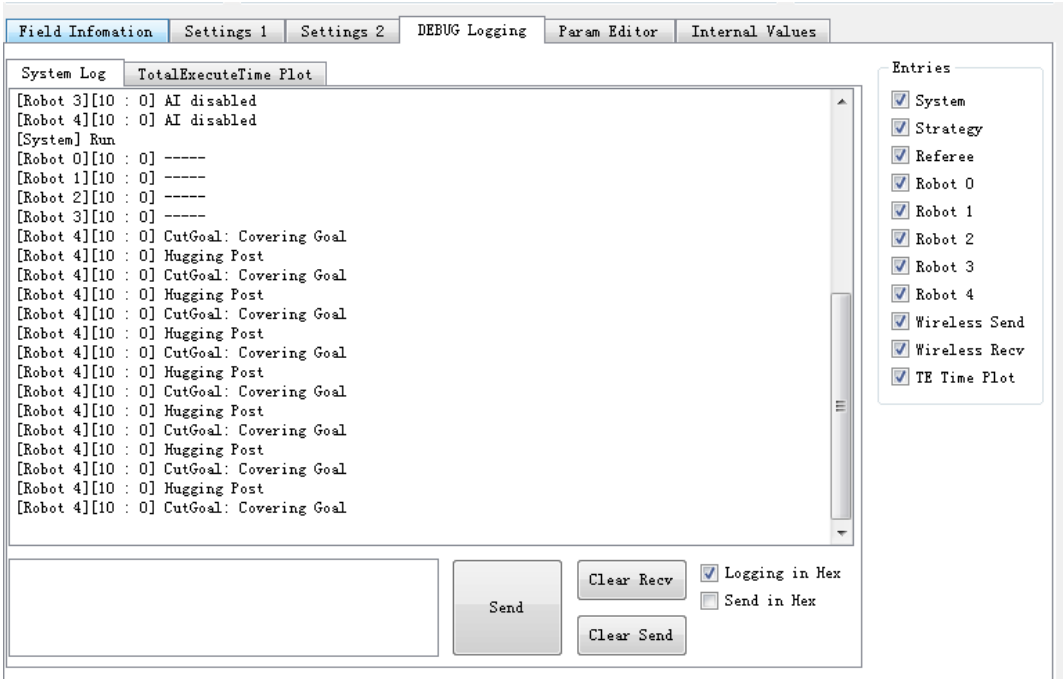

Figure 3-18: Internal Logger

Since the wireless packets are constructed in binary format, therefore, Hex mode is developed to deal with the situation. (Figure 3-20). Comparing to the Normal Mode in Figure 3-19, more bits can be presented in Hex Mode so we will have a better view of the packet (Some bits are invisible in Normal Mode)

Thanks to the QT framework, the conversion from ASCII to Hex is very easy by using toHex() method of a QByteArray type.

The sample codes are as below:

```
if (ui.checkBoxLogWireless->isChecked()) {
if (ui.checkBoxLogWirelessLogHex->isChecked()) {
      ui.pteLogMatch->appendPlainText("[Serial Send] " +
sendBuf.toHex());
      } else {
      ui.pteLogMatch->appendPlainText("[Serial Send] " +
sendBuf);
      }
}
```

The other log messages are generated in the same or similar way by calling the "appendPlainText" method of the text editor.

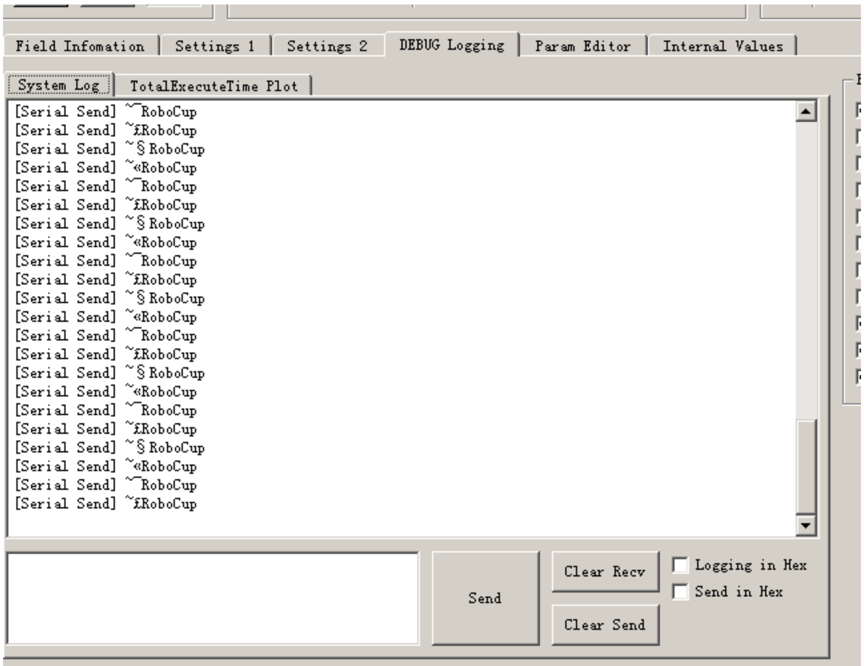

Figure 3-19: Displaying the content of the packet in Normal Mode

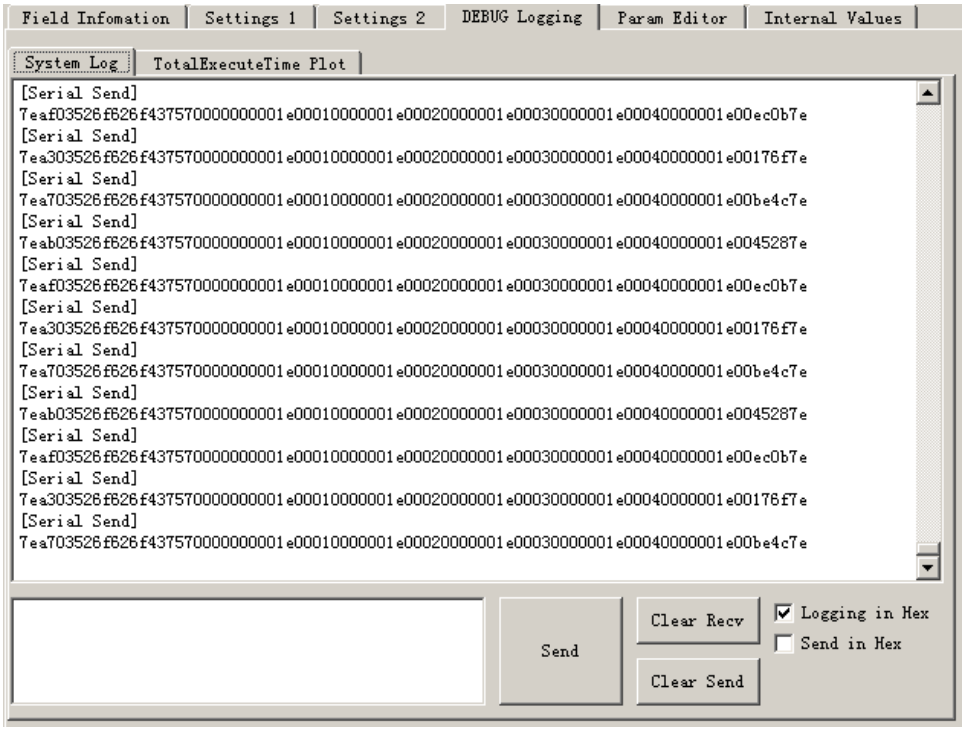

Figure 3-20: Displaying the content of the packet in Hex

Now we will explain the sender part. When the “Send” button is pressed, the slot function will be executed,

```
QByteArray serSend = ui.pteDEBUGSerial-
>toPlainText().toLatin1();

if (!serialComm->isOpen()) {
      serialComm->open(QextSerialPort::ReadWrite);
      if (!serialComm->isOpen())
      {
            QMessageBox::critical(this, "Error", "Unable to
open port!");
            return;
      }
}

if (ui.checkBoxLogWirelessSendHex->isChecked()) {
      serialComm->write(QByteArray::fromHex(serSend));
//…
}
```

First of all, the content in the “Textedit” control will be read and converted by using ToLatin1() method. In this way it can be converted to from hexadecimal later in case we need.

After the step above, the serial port status will be checked. If it is not open an error message will be popped out.

The third step: the application will check if the “Send in Hex” checkbox is checked. If so, it will read the content in hexadecimal (by using “QByteArray::fromHex()” method) before sending them.

At last, the activities will be logged in the log area.

4. STRATEGY IMPLEMENTATION

4.1 Minacity analyzer

Minacity analyze is a very important factor when designing defense algorithms. In the current plan, following condition will be evaluated and a minacity value will be calculated for every enemy's robot:

(1) The distance to our goal line

(2) How easy that it can goal at the current place

Assuming the coordinate of the robot is (x1, y1) and the line segment of our goal line is from (xs, ys) to (xe, ye), the distance (d) can be calculated as below:

(1) Using the distance formula below to calculate the distance from (x1, y1) to (xs, ys) and (xe, ye). Assume the results are "ds" and "de"

$$\mathrm{d} = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

(2) Solve the formula of the line L1 (Ax + By + C = 0), which goes through (xs, ys) and (xe, ye)

$$\begin{cases} A \times xs + B \times ys + C = 0 \\ A \times xe + B \times ye + C = 0 \\ B = 1 \end{cases}$$

(3) Assume L2 is the vertical line of L1 which go though (x1, y1), and their point of intersection is (xp, yp)

The distance d2 between (x1, y1) and (xp, yp) can be calculated using the following distance formula which is used to calculate the distance between a certain point to a line (B = 1):

$$\mathrm{d2} = \frac{|A \times x1 + B \times y1 + C|}{\sqrt{A^2 + B^2}}$$

Thus we can get (xp , yp) by solving the equation set below:

$$\begin{cases} (xp - x1)^2 + (yp - y1)^2 = d2^2 \\ A \times xp + B \times yp + C = 0 \\ B = 1 \end{cases}$$

(4) Assume the distance between (xs, ys) and (xp ,yp) is Ds, and the distance between (xe, ye) and (xp ,yp) is De, finally we can get the result D from the following analyses:

If Ds + De equals to the length of the goal line, then D = d2;

If Ds + De is greater than the length of the goal line and Ds > De, then D = De;

If Ds + De is greater than the length of the goal line and Ds < De, then D = Ds

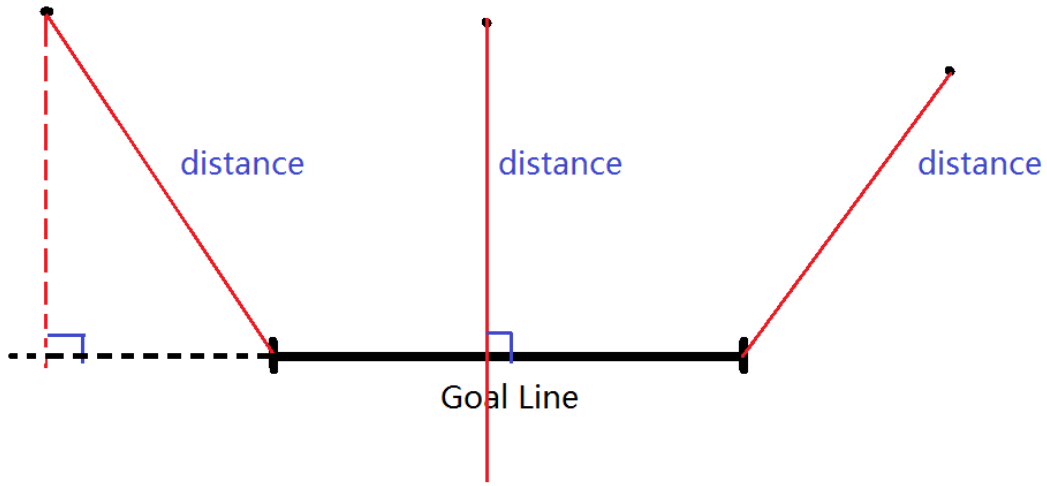

Figure 4-1 Distances between points to goal line

This makes the first part of the result.

The second part is achieved by comparing the possible goal space and takes the maximum as the final value, which is demonstrated in Figure 4-2:

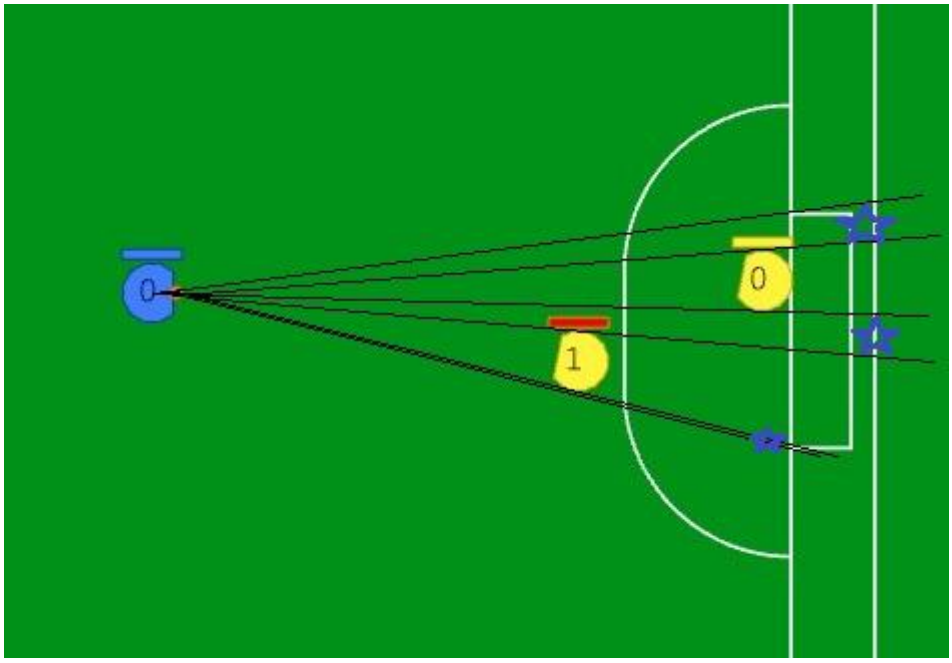

Figure 4-2: Goal analyze

In this Figure, blue robot 0 has three possible spaces to shoot for a goal. However, the middle space is slightly larger than the other two, so we will get the goal line length as a parameter for calculating the minacity value of the blue robot 0.

The pseudo-code of the algorithm is designed as below:

```
class Point //Coordinate
{
      float x;
      float y;
}

class LineSegment //Line class (y = kx + b)
{
      float k; //Slope
      float b;
      Point start; // Start point of the line segment
      Point end; //End point of the line segment

      /*
      Use the start and end point to initialize
      the line segment
      */
      LineSegment(Point s, Point e);

      //Check the distance to
      float distanceToPoint(Point p);
}

Point P_robot; //The location of robot is P.
float r_robot; //The radius of each robot
```

```
float minacity_second_parameter(Point P_robot)
{
int maximum_goal_space = 0;
int goal_space;
foreach (Point P ;on the goal line)
{
goal_space = 0;
LineSegment l = new LineSegment(P, P_robot);

foreach (Point P_otherrobot ;of the other robots on the field)
{

if (l -> distanceToPoint(P_otherrobot) > r_robot) {
      goal_space++;
} else {
      if (goal_space > maximum_goal_space) {
            maximum_goal_space = goal_space;

            /* This macro will record the middle point of this
            space, which will be considered as the best shoot
            direction */
            RECORDSHOOTPOINT;
      }
      break;
}

}
}

return maximum_goal_space;
}
```

In the real situation, the load of the algorithm will depending on the sample rate of the points (on the goal line) used, thus, we take 1cm here so the performance will be acceptable.

Finally, these two results will be combined together. And the minacity value can be calculated.

In the current development stage, the minacity value can only be used to compare which enemy robot might bring bigger trouble to us; in the future we will improve it so that we can design tactics towards certain value intervals. Also, the status of the ball will also be considered.

Here is the effect in the real situation:

Figure 4-3: Minacity analyzer (Acting as team blue)

In Figure 4-3, although yellow No.3 is holding the ball and it is the nearest enemy to our goal, however, because our goal keeper (blue No.0) have blocked part of the goal space, so its minacity have been greatly decreased. In the meantime, we can notice that yellow No.2 is open to goal while no other robots is blocking its shooting space, so it have a higher minacity value than yellow No.3

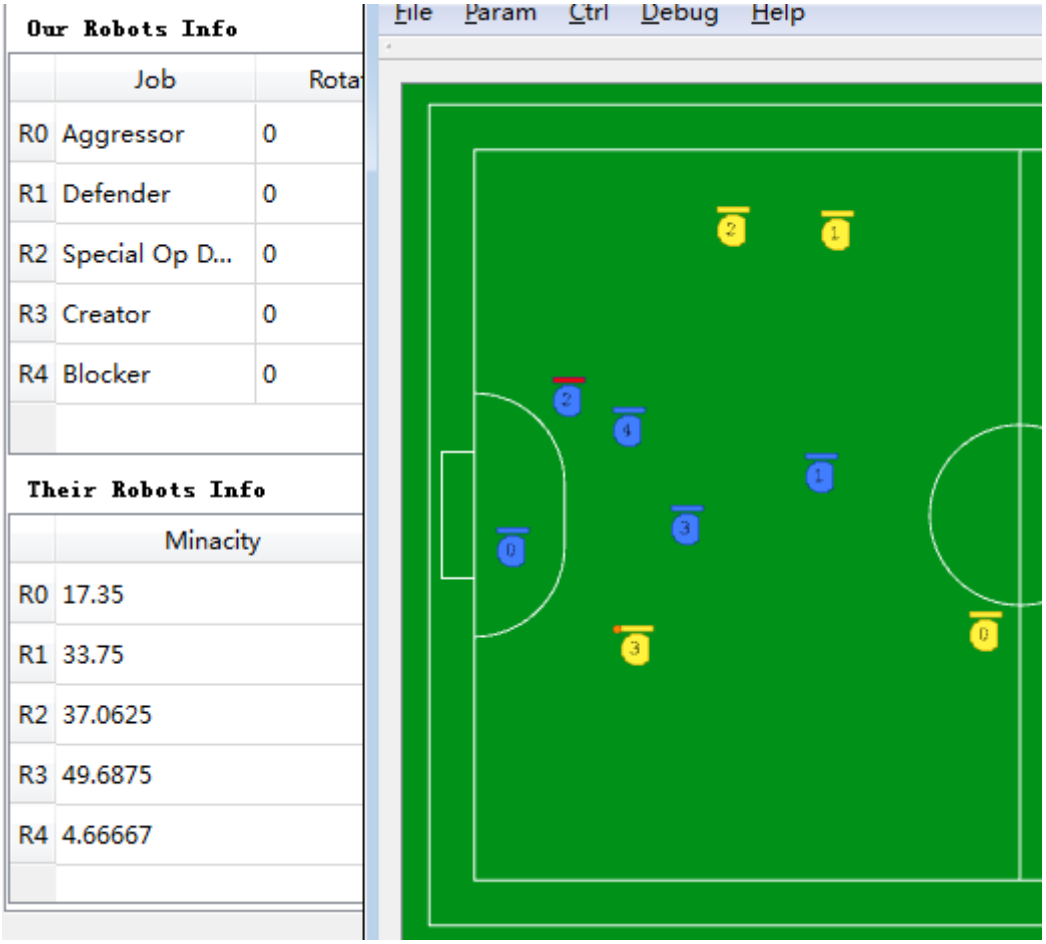

Figure 4-4: Minacity analyzer (Acting as team blue)

In such situation, we use our robot to block the space of yellow No.2, which reduced its minacity value to only 50% of its original value (Figure 4-4).

4.2 Best shooting direction

This paragraph will describe the “RECORDSHOOTPOINT” macro in the code of the selection 4.1

Because all the points on the goal line have the same abscissa, we can get the best shooting direction (actually a point on the goal line) by recording its ordinate.

Thus we can get the target point on the goal line, which is (xs, ys).

Assume that the ball has a coordinate (xb, yb), we can get the equation of a straight line which go through (xs, ys) and (xb, yb).

To make the shoot, the robot has to move to the shooting point. This point can be calculated as below:

Assume the radius of the robot is R and the equation of the line is:

$$y = kx + b$$

We can get the coordinate of the shooting point (xr, yr) by solving the following equation set:

$$\begin{cases} R^2 = (xb - xr)^2 + (yb - yr)^2 \\ yr = k \times xr + b \end{cases}$$

And the rotation angle (t) of the robot can be calculated by this equation:

$$t = \arccos\left(\sqrt{\frac{(xs - xb)^2}{(xs - xb)^2 + (ys - yb)^2}}\right)$$

Figure 4-5 has given the illustration of this process.

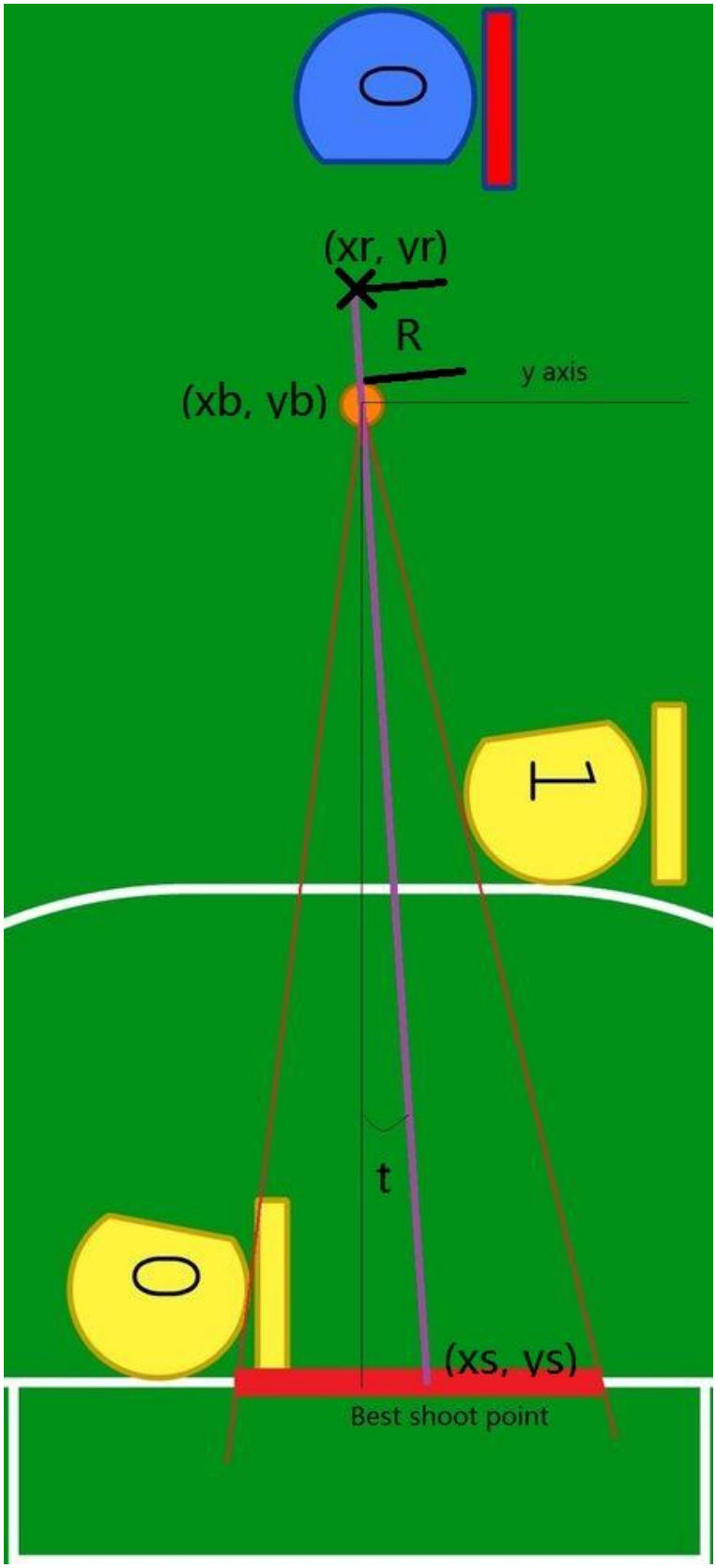

Figure 4-4

4.3 Simple-Kick demo

This demo is based on the “RECORDSHOOTPOINT” macro which is described in paragraph 4.2. A robot will catch the ball then shoot in this demo.

This demo is one of those basics of the direct free kick in our strategy. By well performing the demo, we might have a goal by doing a free kick.

First of all, the robot will get to the prepare point, which is located on the extend line of the best shooting point to the ball. The distance between the prepare point and the ball is a specified value (Bigger than the radius of the robot)

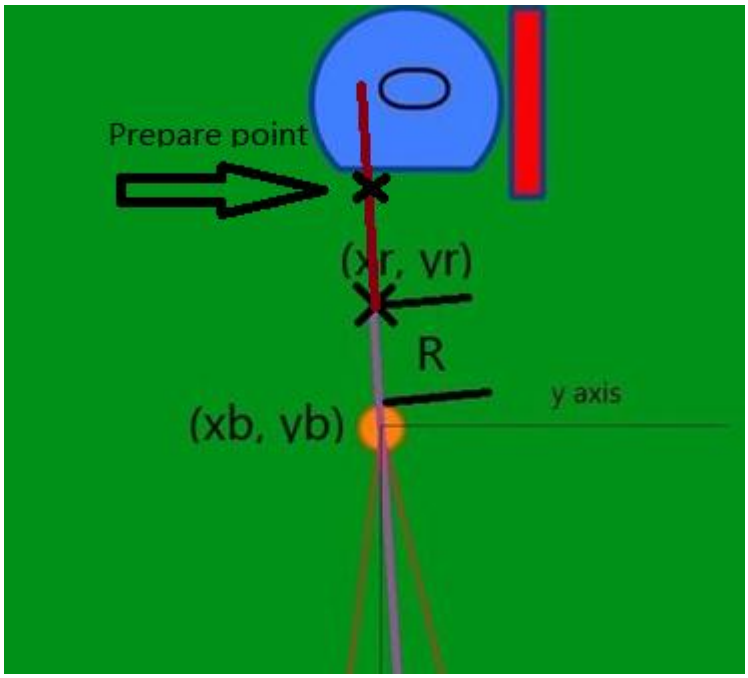

Figure 4-5: A modification of Figure 4-4 indicates the prepare point

After that, the robot will go forward to the ball and make the shoot.

To prevent the case that the ball might go out of field, a square area is pre-defined in this demo. If the ball went out of the area, the robot will go to the center of the field.

The algorithm can be described as below:

(1) Get the coordinate of the prepare point

(2) If the coordinate is out of bound, go back to Step 1

(3) Calculate the rotation that the robot should rotate to kick the ball

(4) Rotate the robot and make it run to the prepare point.

(5) If Step 4 is completed, continue. Otherwise, execute Step 4

(6) Run to the ball and shoot.

5. VISION FILTER

5.1 Background

The coordinates of objects on the field are come from SSL-Vision application, which calculate the incoming data and extract the information from a camera. Unfortunately, there are different kinds of noises exists in the nature that might influence the data collection mechanism of the camera.

On the other hand, when the data arrive at our strategy system, they are already obsolete because of the delay.

Therefore, we need a certain mechanism to filter out the noise, while predict the current coordinate of the target.

5.2 EKF (Extended Kalman Filter) code porting

The first attempt is to port a pre-made algorithm of EKF [], from MATLAB to C++.

Since C++ and QT did not have required matrix function, a matrix class had been made prior to implement the algorithm.

In this class, a QVector container is used to store the elements in the matrix, about the inverse function, since only 2x2 matrix inverse is needed, so we implement it directly as below:

```
float rd = 1.0 / (mat[0] * mat[3] - mat[1] * mat[2]);

HBMatrix m(2,2);
m.setElement(1,1,rd * mat[3]);
m.setElement(2,2,rd * mat[0]);
m.setElement(1,2,rd * mat[2] * -1.0f);
m.setElement(2,1,rd * mat[1] * -1.0f);
return m;
```

The whole definition of the class is as below:

```
class HBMatrix {

private:
	QVector<float> mat;
	int nrow;
	int ncol;

public:
	HBMatrix();
	HBMatrix(float* fArray, int r, int c);
	HBMatrix(int r, int c);

	//Get the number of rows / columns
	inline int getnrow() {return nrow;}
	inline int getncol() {return ncol;}

	float getElement(int r, int c);
	void setElement(int r, int c, float v);
	HBMatrix transpose();
	HBMatrix inverse2x2();
	void reset();
	void reset(int r, int c);

	float getMinor(int r, int c);

	const void operator= (HBMatrix& other);
	friend HBMatrix operator+ (HBMatrix a, HBMatrix b);
	friend HBMatrix operator- (HBMatrix a, HBMatrix b);
	friend HBMatrix operator* (HBMatrix a, HBMatrix b);
};
```

The integration of the function is done as below:

Firstly, a tolerance value is defined as a bound value of the incoming data, for instance, the value of the long side of the whole field. If an object, for example, the ball, is reported outside the field, we can filter it out since it is useless for us to consider a dead ball. If the ball is out of the tolerance value for a long time, we can consider that there is no ball on the field

After that, we can intercept the raw vision data by processing the “ballVision” variable:

```
if (ballVision.getX() < -tolerence || ballVision.getY() < -
tolerence || ballVision.getX() > tolerence ||
ballVision.getY() > tolerence)
// Ball Lost
{
if (ballLostTimer == 0) {
//If the ball is lost for a short time, we assume that it is
static
ballLostTime = HBGlobalTime.elapsed();
ballLostTimer++;

incomingVision.ball[0][i].XPos=acceptableBallLoc.getX();
incomingVision.ball[0][i].YPos=acceptableBallLoc.getY();
return;
}
      else if (ballLostTimer < 300 && ballLostTimer > 0)
{
ballLostTimer = HBGlobalTime.elapsed() - ballLostTime;
incomingVision.ball[0][i].XPos=acceptableBallLoc.getX();
incomingVision.ball[0][i].YPos=acceptableBallLoc.getY();
return;
} else {
incomingVision.ball[0][i].XPos=ballVision.getX();
incomingVision.ball[0][i].YPos=ballVision.getY();
ekfBall.reset();
return;
}

} else {
//If the ball is present, process it inside the filter
acceptableBallLoc.set(ekfBall.process(ballVision));
incomingVision.ball[0][i].XPos=acceptableBallLoc.getX();
incomingVision.ball[0][i].YPos=acceptableBallLoc.getY();

return;
}
```

However, the result is not as we expected. When the ball moves, the data out from the filter changes very slowly, makes it impossible to track the ball in the real time.

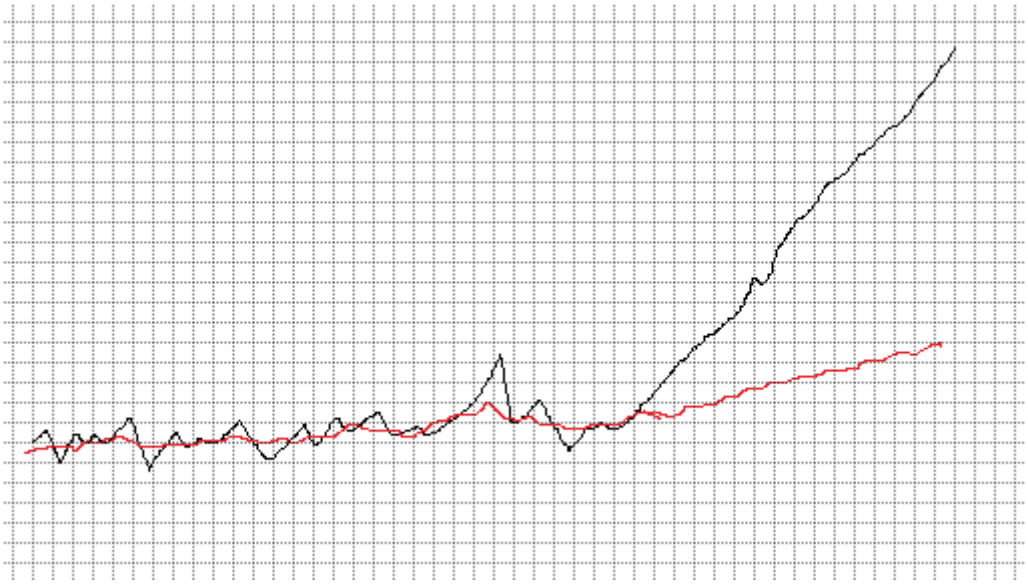

Figure 5-1: The simulate result of ported EKF code

After carefully evaluation, we abandoned this method because:

(1) It does not work
(2) The algorithm have to be rewritten, it will cost time
(3) Even it is working, we need to adjust its parameters, and the optimization process will cost even more time.
(4) We have to implement different filters to the ball and the robots.

5.3 Movement status based filter (MSBF)

Based on the unsuccessful attempt of EKF, a new type of filter is needed to be integrated into our strategy program. Therefore, a movement status filter had been designed to deal with the issues.

Figure 5-2: Overall mechanism of MSBF

Any objects on the field have two states: motionless state and moving state. Thus, the filter is designed to have two different sets of algorithms to deal with these states. Besides, due to the existence of noise and nature resistance, an object might change its state at any time, and its next location can be largely varied, so an intermediate state is designed to deal with such situation, which is named "prediction state".

MSBF is 1-dimensional filter, so it can be applied to any kind of objects in any dimension space if they follow the Newton's laws of motion. For example, assume the filter is f, the coordinate of a robot is (x, y), so we can filter it as (f(x), f(y)); for a ball whose location is (x, y, z), we can filter it as (f(x), f(y), f(z)).

5.3.1 Value buffers and control variables of MSBF

To store the receiving data, some list containers are initialized inside the constructor function of the filter. There are also two pointers, named "Primary List Pointer" (PLP) and "Secondary List Pointer" (SLP), are used to control the data flow. The container at which the PLP points, is defined as the primary list. The other list will be called the secondary list.

Together with the lists, there are also some other variables which control the mechanisms of the filter:

(1) Minimum Moving Speed (MMS): A threshold value of speed defined as the boundary between motionless state and moving state, which means the object

will be regarded as “motionless” if its speed is less than the threshold, even it is still moving.

(2) Delay: The time elapsed when the next piece of data comes. This value can be calculated using the system ticks via “GetTickCount” function.

(3) Speed Limit (SL): This value is calculated using the current speed (v) of the object, the maximum possible acceleration (a) and the Delay (t)
The formula is as below:

$$SL = v + at$$

(4) State Hold Tolerance (SHT): This value indicates how many continuous values, which might cause a change of the state that we should ignore before changing the state. The bigger default value the SHT is, the better resistance the filter will have against noise. However, its response towards real state changes will be slower.

(5) Status Indicator (SI): Stores the current state.

(6) Buffer Size Limit (BSL): Limits the length of the buffers.

(7) Low Speed Moving Threshold (LSMT): Will be described later.

Furthermore, there are two additional lists used to store speed and acceleration.

5.3.2 Base mechanisms and prediction state

The initial state of the filter is motionless mode. If the primary list is empty, any incoming data will be put into the primary list. If the primary list is not empty, an analyze process will begin. For example, we are currently at motionless state. However when we receive the next value and compare it to the last value of the primary list, we find that the speed is above the MMS. In this case, we will put this value to the secondary list and reduce the SHT; otherwise, we will put the value into the primary list and set the SHT to its original value and clear the secondary list.

When the value of the SHT is less than its default value, the filter will enter a phase called “prediction state”. The algorithm in this state is decided by the previous state. However, The SI will not be changed in this state.

When the SHT value is below zero, the state is changed by following steps:

(1) Change the SI;
(2) Clear the primary list and restore SHT to its default value;
(3) Exchange the address stored in the PLP and SLP.

In common situation, the SI will change from either motionless to moving, or from moving to motionless. But there is a special case. In some situation, the speed might exceed the value of SL, which will be considered as noise under normal circumstances. However we will still store them in the secondary list and reduce the SHT. In this case, when the SHT reaches 0, all the procedures above will be executed, but the SI will remain at moving state. After the analyzing process, the filter will calculate the output based on the SI, which will be described in the following chapters.

5.3.3 Moving state

In the moving state, the Newton’s law will be used to predict the output value of the filter:

$$\mathrm{ds} = v_0 \Delta t + \frac{1}{2} a(\Delta t)^2$$

In the formula, “v” is the speed of the object, “a” is the acceleration of the object, “ds” is the correction value which should be applied to the received value, and “△t” here is the delay in seconds. So the output value of the filter should be the last value in the primary list, plus the correction value.

For example, in the moving state the original last value in the primary list is 1, and then a new value 3 comes. Since 3 – 1 = 2 is greater than the MMS and less than

the SL, so the speed (v) will be 2; assume the previous speed value is 1 and the delay is 0.1s, so the acceleration will be 2 – 1 = 1.

Thus, the "ds" can be calculates as

$$\mathrm{ds} = 2 * 0.1 + 1 * (0.1)2/2 = 0.2 + 0.005 = 0.205$$

And the output value is

$$\mathrm{Output} = 3 + \mathrm{ds} = 3 + 0.205 = 3.205$$

5.3.4 Motionless state

In the motionless state, the average value of all the elements in the primary list will be used as an output value.

Assume the length of the primary list is N, "a" is the value of the PLP, "k" is the offset of the PLP, and the output value in the motionless state can be represented as:

$$\mathrm{Output} = \frac{\sum_{k=0}^{N} a_k}{N}$$

5.3.5 Dynamic buffer length adjusting (DBLA)

As mentioned above, if the speed of the object is less than MMS, it will be considered as motionless even it is moving. Figure 5-3 is the simulation in the motionless state. The blue line is the raw value we get from the camera, while the red line is the output value of the filter; the time is used as the horizontal ordinate. From Figure 5-3, we can observe that in the case of low speed movement, the output value is always fall behind. That is because of the old value in the filter, which performed a negative impact on the output. Lessening the MMS seems can be a solution. However this is not possible. Since the existence of the noise, a too small MMS will be easily exceeded by the noise, which makes the rapid status change

between the motionless and the movement states. As a result, this will make the output unstable.

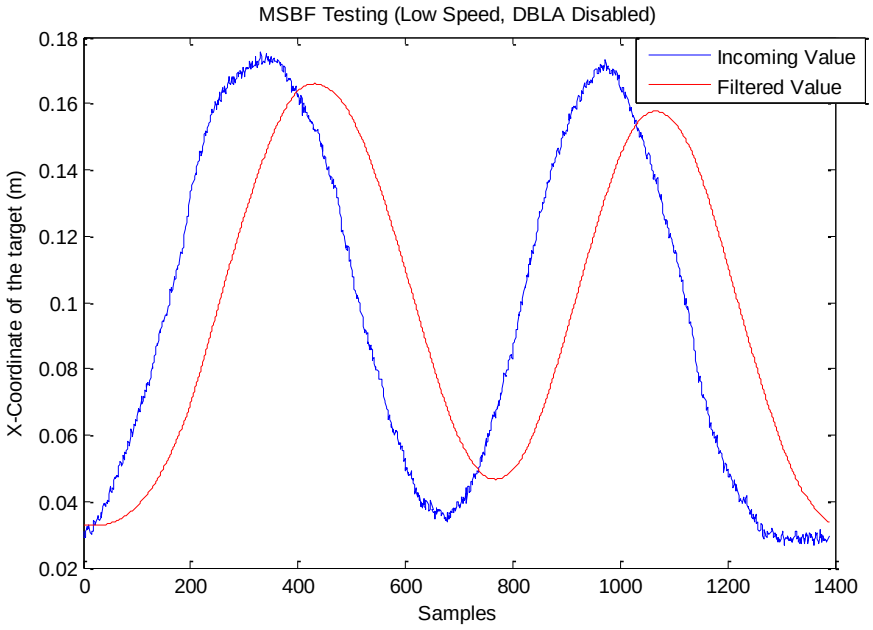

Figure 5-3: Low speed object moving test

So a dynamic algorithm has been developed as a countermeasure of this situation. In this algorithm, LSMT is introduced.

First of all, assume that the ball will move slowly for a long time so that the SI will remain the same. When a new value comes, we will compare the value and the last output value of the filter by doing subtraction. If the difference is above 0, a counter will increase by 1; if the difference is below 0, this counter will decrease by 1. And if the value of the counter is bigger than LSMT, the size of the filter will be divided by 2, with first half of the elements abandoned.

5.3.6 MSBF vs. EKF

Although EKF is a very good algorithm, however it does not suit the project since:

(1) EKF considers every properties of an object into a unique framework. So its portability becomes a big problem when we applying it to different objects.

For instance, each robot have its own rotation value, which should be calculated in the filter, however, the ball's rotation do not need to be considered. On the other hand, the height of the ball should not be processed in the filter but the robots. Therefore, we have to implement different filters to both robots and balls.

(2) The value of the parameters of the EKF, have to be manually adjusted. As for each environment, the best parameters might vary, so the optimizing process will be time-consuming.

(3) The programmer must have a solid mathematics background with a good understanding of the robotics, which is difficult to achieve for beginners.

The design of MSBF solved the issues above since:

(1) It can be applied to any objects by disassembling its location to 2D / 3D space without change.

(2) The filter can adjust its parameters by itself. Since the filter only take the internal timestamps and the location difference into concern, so it is basically immune to the interference caused by the environment.

(3) Effective but easy to implement. The mechanism can be understood by even high school students, which is useful for popularization of such project.

5.4 MSBF Testing

5.4.1 Method

In the test, we apply this filter to the x-axis of a ball in the vision simulator and draw the graph on the user interface. In the 2-dimensional graph, the x-coordinates of the ball are presented in the y-axis of the graph and time is the x-axis. The blue line in the graph illustrates the raw value which we get from the vision server / simulator; the red line will indicates the output value from the filter.

5.4.2 Static object test

First of all, a static ball is used to test the filter. Although a lot of noise has been introduced, the output is steady even the noise is bigger at some point, showing a very good resistance to noise.

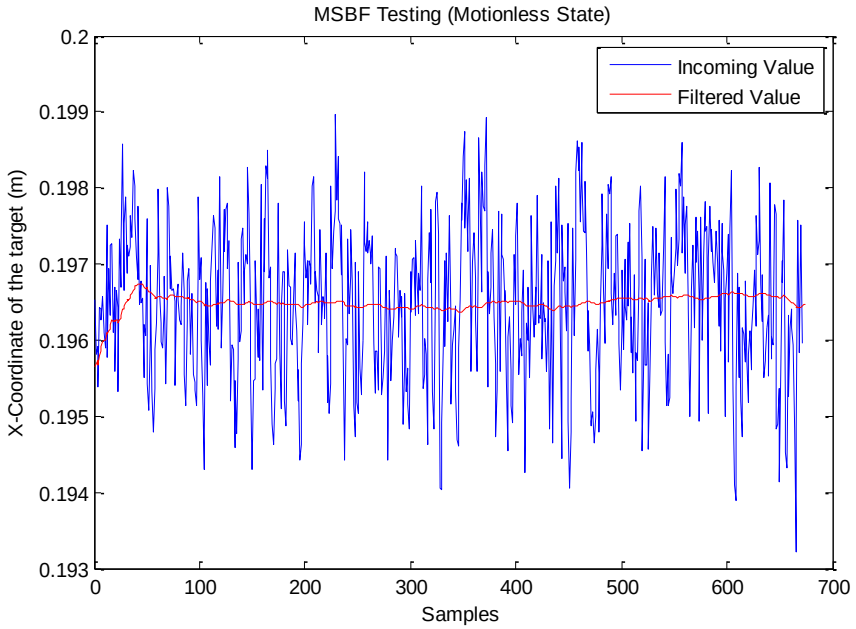

Figure 5-4: Motionless state testing

5.4.3 Low speed moving test

After the implementation of the dynamic buffer, the low speed moving test is done again using the same parameters and a similar moving track. From the result (Figure 5-5), we can observe that the latency time between the output and the input in this situation have been reduced by about 70%.

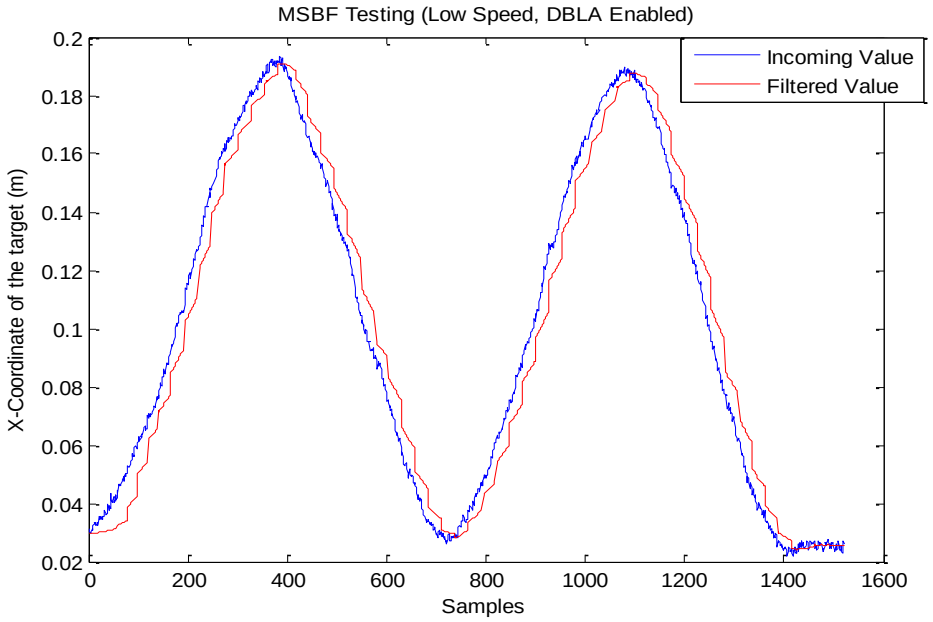

Figure 5-5: Low speed moving test with DBLA implemented

5.4.4 Tests on the moving object

Also, a test on the moving object had been performed. In this test, since we need to compare the previous predicted value and the current input value, thus we deleted the first element in the raw value matrix before plotting in the MATLAB, which aligned the N_{th} value in the predict value matrix, with the $(N+1)_{th}$ value in the raw value matrix. The plotting is shown in Figure 5-6, and we can observe that the filtered value well presented the next incoming value.

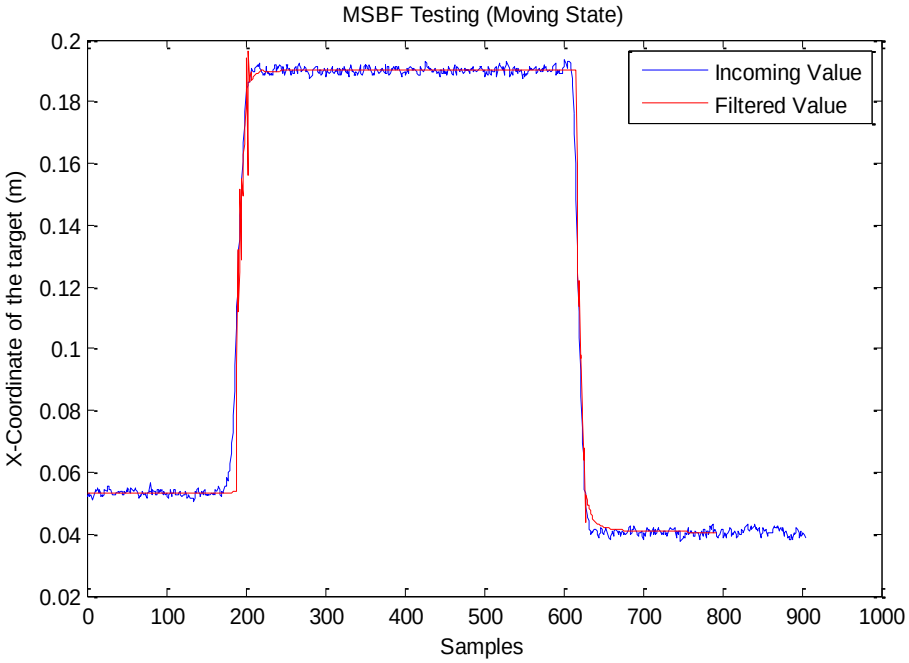

Figure 5-6: Moving object test (with noise)

A more detailed test is performed without noise. The application will log the input, output and the error between the current input value and the previous predicted value. Below are some sample values.

```
In = 1.53351 out = 1.52426 error = -0.000588413
In = 1.51447 out = 1.48462 error = -0.00979524
In = 1.50494 out = 1.50711 error = 0.0203153
In = 1.49542 out = 1.48563 error = -0.011693
In = 1.48589 out = 1.47664 error = 0.000261497
In = 1.47637 out = 1.46677 error = -0.000274773
In = 1.46684 out = 1.45649 error = 7.27231e-005
In = 1.44779 out = 1.42013 error = -0.0086988
In = 1.42874 out = 1.4085 error = 0.00861349
In = 1.41922 out = 1.4205 error = 0.0107177
In = 1.40969 out = 1.40019 error = -0.0108112
In = 1.40017 out = 1.39032 error = -2.8072e-005
In = 1.38112 out = 1.35281 error = -0.00920502
In = 1.37159 out = 1.36975 error = 0.0187788
In = 1.36207 out = 1.35357 error = -0.00768522
In = 1.34302 out = 1.31442 error = -0.0105539
```

In the test, the maximum error is about 4cm and the average error is about 1cm.

6. WIRELESS PROTOCOL FOR DECT MODULE

6.1 Background

As a limitation, the old protocol can only control 4 robot in the same time. Thus, an extra transmitter will be required to control the 5th robot.

To eliminate this defect, a new kind of protocol has been introduced.

Therefore, we need to add support for the new protocol, while maintain the support for the original one we used.

The specification of the module and the protocol are described in << HW 86012 DECT Module >> on http://www.hoeft-wessel.com/index.php?id=238.

6.2 User Interface Control

First of all, a combobox had been added to the wireless options (Figure 6-1), which is used to control the protocol we used.

If we do not need wireless output, or we do not have a serial port for the wireless module, we can choose “Off”.

The “Transparent” is the old protocol we want to support; The “Protocol” is the new type protocol.

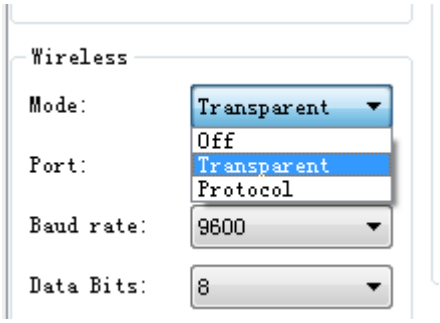

Figure 6-1

6.3 Build the packet

```
qmx.lock();
serialPacket->clear();
if (serMode == 2) {
      lastUsedAddrByte = nextaddr(lastUsedAddrByte);
      serialPacket->append(lastUsedAddrByte);
      serialPacket->append(HDLC_CTRL);
      serialPacket->append(botniaMac);
}

for (int i=0; i<5; i++)
{
      if (serMode == 1)
            buildPacket(i);
      else if (serMode == 2) {
            buildDECTPacket(i);
      }
}
```

Above is the first part of the packet constructing. A lock is used to protect the buffer (serialPacket) in case being modified by different threads at the same time.

Variable “serMode” decides the transmit mode of the module. “2” states that the “Protocol” mode is chosen in the UI.

First of all, the header of the packets is added in the buffer, and then the data segment is added to the buffer almost the same way as the original one. However, according to the specification, the start bit will not taking part in the later frame check sequence (FCS) generating process, so it is not added in this step.

Below is the second part of the code for the packet data generation. At the beginning the FCS is calculated through CRC16 algorithm [10] and appended to the buffer.

```
if (serMode == 2) {
      unsigned int hldc_crc = pppfcs16(serialPacket->data(),
serialPacket->size());
      serialPacket->append((char)(hldc_crc%256));
      serialPacket->append((char)(hldc_crc/256));

      char* i=serialPacket->data();
      int j=0;

      while (j<serialPacket->size()) {
            if(*i == 0x7e)
                  {
                        QByteArray tmp;
                        tmp.clear();
                        tmp.append(0x7d);
                        tmp.append(0x5e);
                        serialPacket->replace(j, tmp);
                        j++;
                  }
                  else if (*i == 0x7d)
                  {
                        QByteArray tmp;
                        tmp.clear();
                        tmp.append(0x7d);
                        tmp.append(0x5d);
                        serialPacket->replace(j, tmp);
                        j++;
                  }
                  i++;
                  j++;
            }
            serialPacket->prepend(HDLC_STARTSTOP);
            serialPacket->append(HDLC_STARTSTOP);
      }
      qmx.unlock();
```

Since “0x7E” and “0x7D” is the reserved characters for the start / stop bit of the packet, so for any “0x7E” appears inside the data part, we should replace it by “0x7D 0x5E”. The same thing is applied for “0x7D”, which should be substituted by “0x7D 0x5D”.

After that, we can finally prepend the start bit then add the stop bit to the buffer and release the lock, manifesting that the data is ready for sending.

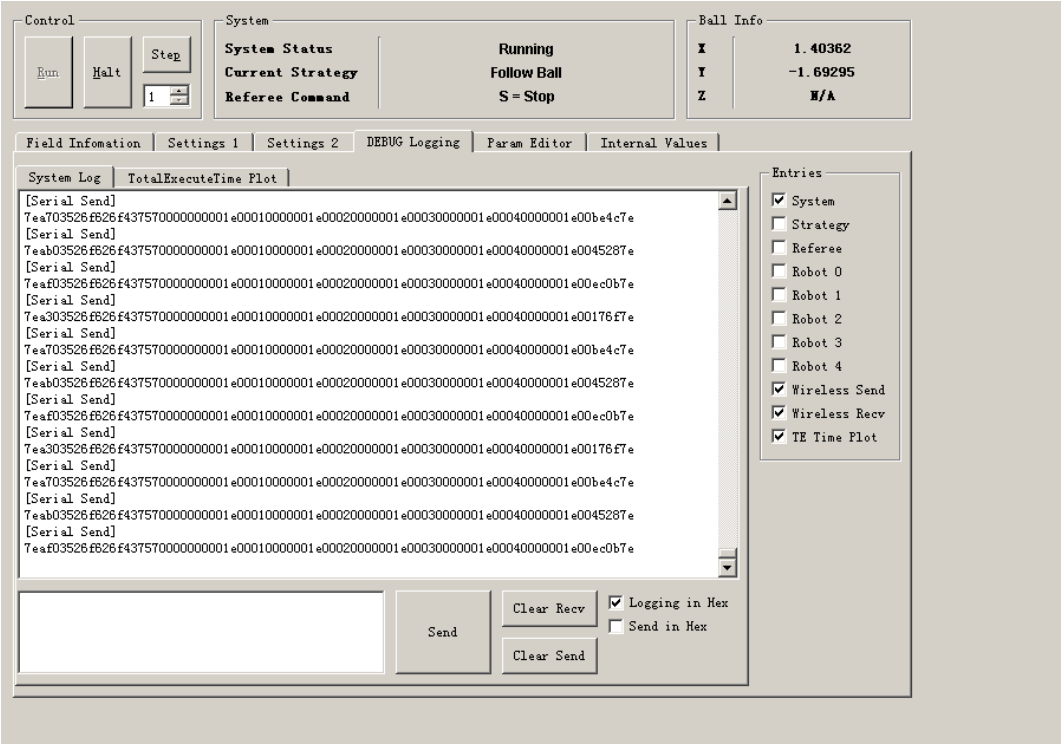

Figure 6-2: Data sending in the Protocol Mode

7. CONCLUSION

This is a big project which contains about 120000 lines of codes and they are written by different people. So there are lots of overlapping parts which are not easy to understand at some time. Therefore I chose the environment upgrading and the UI tasks as the start of the thesis.

In my point of view, the low level structure of the application needs to be refactored. Since the base class is not well designed and some basic function is buggy. For example, the code for judging if some robot had gained the control of the ball uses the distance between them only. And there is no global timer exists in the applications at the beginning, which makes some actions extremely difficult to perform.

However, the done modifications in this thesis can be considered succeed since they increase the predictability and controllability greatly, and the positioning issue is ameliorated by the MSBF. The minacity analyzer can be used to design more advanced dynamic defense strategies and the “Best shooting direction” algorithm can be a good tool in attacking.

An issue which will probably become the primary future work in the current code base is the algorithm of moving state of the MSBF which needs to be made better. A third buffer is planned to deal with the unstable output when great changes happen to acceleration, on the other hand, the algorithm is not yet used in any real competition, thus its full potential have not been reached. We believe that it could perform better if tuned with more experiences.

REFERENCES

Web resources:

[1] Small Size Robot League
(http://small-size.informatik.uni-bremen.de/)
[2] Referee box
(http://small-size.informatik.uni-bremen.de/referee:start)
[3] SSL-Vision
(Carnegie Mellon University, http://code.google.com/p/ssl-vision/)
[4] Protocol Buffers - Google's data interchange format
(Google, http://code.google.com/p/protobuf/)
[5] PUV Robotics
(Vaasa University of Applied Sciences, http://robotics.puv.fi/)
[6] << HW 86012 DECT Module >>
(Höft & Wessel AG, http://www.hoeft-wessel.com/uploads/media/HW86012.pdf)
[7] Qt – cross-platform application and UI framework
(Nokia, http://qt.nokia.com/)
[8] Dependency Walker
(Steve. P. Miller, http://www.dependencywalker.com/)
[9] Extended Kalman Filter
(http://www.cs.unc.edu/~welch/kalman/,
http://www.cs.ubc.ca/~murphyk/Software/Kalman/kalman.html)
[10] CRC 16 algorithm
(Black Duck Software
http://www.koders.com/c/fid6BB47D55FAE4B2158CDF7F923C8CACD953525F04.aspx)