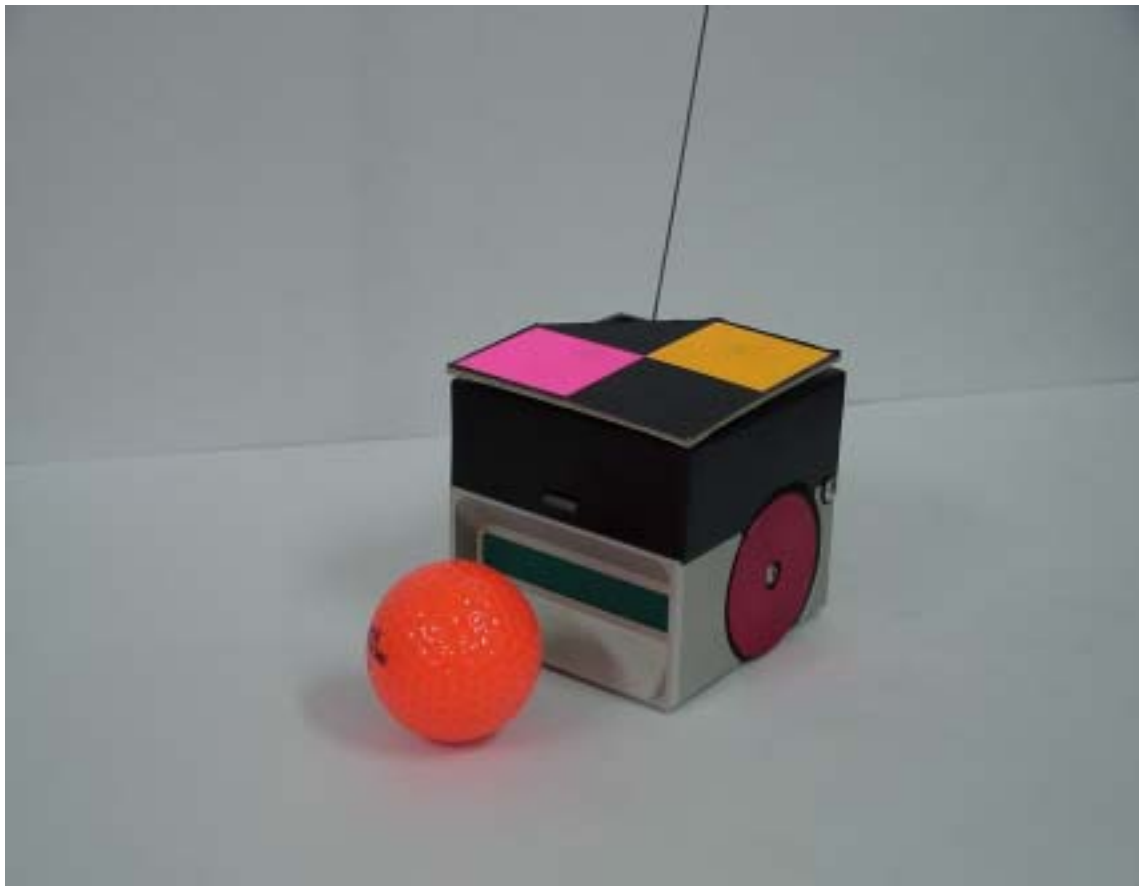


# Robot Soccer

YSR-A 5vs5 System Manual (Meteor 2/4 type)



 **Yujin Robotics Co., Ltd**

## **Yujin Robotics Co., Ltd.**

Soccer Robot Dept. of Kasan-Dong office  
SK Twin-Tech Tower B-204, Kasan-Dong, Keumcheon-Gu,  
Seoul, Korea  
Zip code : 153-802

### Service Calls

82-2-864-3860 (A.M. 8 to P.M. 6)

82-2-864-2789 (FAX)

E-mail : [hskim@yujinrobot.com](mailto:hskim@yujinrobot.com) or [vikilee@yujinrobot.com](mailto:vikilee@yujinrobot.com)

### Sales contact

If you have any question for sales, you can contact our sales manager Mr.  
Hojin Lee. :

[hojin@yujinrobot.com](mailto:hojin@yujinrobot.com)

### Web Page

If you have access to the Internet, you can visit Yujin's web page at the  
following address:

<http://www.edrobot.com>

# Table of Contents

---

<b>1. Robot System.....</b>	<b>7</b>
1.1 Introduction .....	7
1.2 Functions of Components .....	8
1.3 Robot system based on vision.....	9
1.4 Etc.....	11
<b>2. How to Set Camera .....</b>	<b>13</b>
2.1 Camera Setting .....	13
2.2 Setting up the camera.....	15
2.3 Setting Lens.....	18
<b>3. Robot test program .....</b>	<b>22</b>
3.1 Robot test program.....	22
3.2 RF communication module.....	24
<b>4. Installation and Execution of Vision Program .....</b>	<b>27</b>
4.1 Vision Program Installation .....	27
4.2 Execution of Vision Program.....	39
<b>5. Appreciation of Robot Program .....</b>	<b>58</b>
5.1 Vision System.....	60
5.2 Strategy/Tactic system .....	62
<b>6. Robot Control Program.....</b>	<b>72</b>
6.1 Movement of Robot .....	73
6.2 Strategy Part.....	78
6.3 Robot Control Program.....	88
6.4 Strategy .....	95

## List of Figures

---

- Figure 1.1 Organization of soccer robot system
- Figure 1.2 Main parts of robot system
- Figure 1.3 Body of robot
- Figure 1.4 Modeling of A type robot
- Figure 1.5 Matrox Meteor 2/4 card – frame grabber card
- Figure 1.6 Samsung CCD camera SDC-410ND
- 
- Figure 2.1 Positions of Case segregation bolts
- Figure 2.2 Switch to be off
- Figure 2.3 W.B Menu
- Figure 2.4 Menus for Camera (Light Green part: optional menus)
- Figure 2.5 (Front position)
- Figure 2.6 (Side position)
- Figure 2.7 Guide of Luminosity
- Figure 2.8 Guide for Zoom
- Figure 2.9 Guide for Focus
- Figure 2.10 In case of high Luminosity (Bright)
- Figure 2.11 Screen data after click SetColor in case of high Luminosity
- Figure 2.12 In case of low Luminosity (Dark)-
- Figure 2.13 Screen data after click SetColor in case of low Luminosity (Dark)
- Figure 2.14 In case of proper setting
- Figure 2.15 Screen data after click SetColor in case of proper luminosity (Best)
- 
- Figure 3.1 Setting of Robot ID
- Figure 3.2 Dip Switch
- Figure 3.3 Screen shot of the robot test program
- Figure 3.4 Commercial communication module
- Figure 3.5 Channel Setting for Transmitter & Robots
- 
- Figure 4.1 Matrox Meteor 2/4 card
- Figure 4.2 Placing Matrox Meteor 2/4 card in a PCI slot
- Figure 4.3 Connecting Camera Cable
- Figure 4.4 Visual C++
- Figure 4.5 Option view

Figure 4.6 Vision program  
Figure 4.7 Zoomed Screen  
Figure 4.8 SetColor  
Figure 4.9 Ball color range of the real image  
Figure 4.10 Select the robot for setting the robot team color  
Figure 4.11 Set the robot team color  
Figure 4.12 Set the robot ID color  
Figure 4.13 Set the boundary of the playground  
Figure 4.14 Set the robot size  
Figure 4.15 Set pixel size  
Figure 4.16 Save the configuration file

Figure 5.1 Flow chart of Robot Soccer System  
Figure 5.2 Feedback System  
Figure 5.3 Functions Processing in Vision  
Figure 5.4 Copy “YujinSRHost” in the CD and select all files  
Figure 5.5 Select Registration information by right button of mouse  
Figure 5.6 Delete Read Only property in Registration information  
Figure 5.7 Game Class  
Figure 5.8 Member Function  
Figure 5.9 Making HomeRoboMovetoCenter() function in Void type  
Figure 5.10 Observer part of Game menu

Figure 6.1 Duty rate  
Figure 6.2 H-bridge Circuit Map for each direction  
Figure 6.3 Block for Internal Controller for a Robot  
Figure 6.4. Block for a Total Controller  
Figure 6.5 Coordinate System of A Robot  
Figure 6.6 Relation for Prior and Angular Speed  
Figure 6.7 Control of a Robot’s Angle  
Figure 6.8 Control of Position

---

Table 3.1    Setting's for Robot ID's

Table 3.2    Specification of Pins

Table 3.3    Description of module

Formula 6.1

# 1. Robot System

## 1.1 Introduction

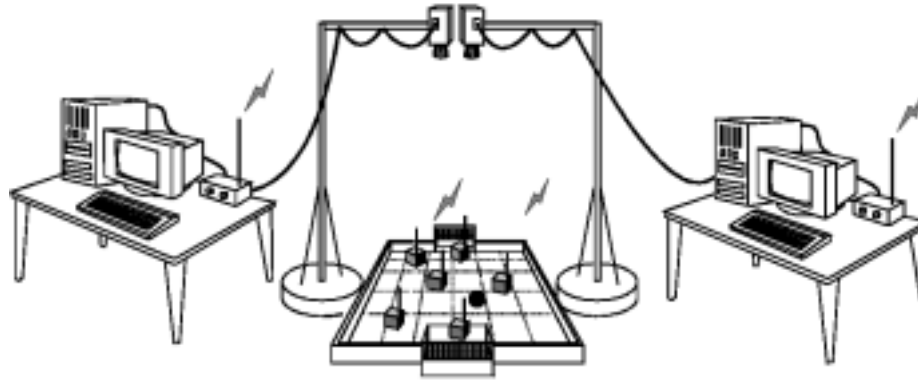


Figure 1.1 organization of soccer robot system

As you can see Figure 1.1, this drawing describes the organization of soccer robot system. On this picture, the intelligence is included in the host computer, the robots, or both of them. According to where to be embodied for the intelligence of the system, there can be divided into Vision-based soccer robot system and Robot-based soccer robot system.

Vision-based soccer robot system can be divided into Remote-brainless soccer robot system and Brain-on-board soccer robot system. Most embodied robot systems are included in one of those 3 ways or mixed of them. As Figure 1.1, our soccer robot system is a remote-brainless soccer robot system, so the host computer has the intelligence and sends commands to the robots through the transmitter. If the robots receive the commands from the host computer, they only control the movement and operation.

For Vision-based soccer robot system, a robot system is a part of robot soccer system. Figure 1.2 shows a diagram of robot system. Robot system consists of four parts; micro-controller, motor driver, communication, and power unit.

The complete robot is illustrated in Figure 1.2. The robot has a circuit board, battery, motor, and body (including wheels). The circuit board can be separated into the lower circuit board that has power devices (voltage regulator, motor driver, etc.) and RF module and the upper circuit board that has the micro-controller (ATMEGA128)

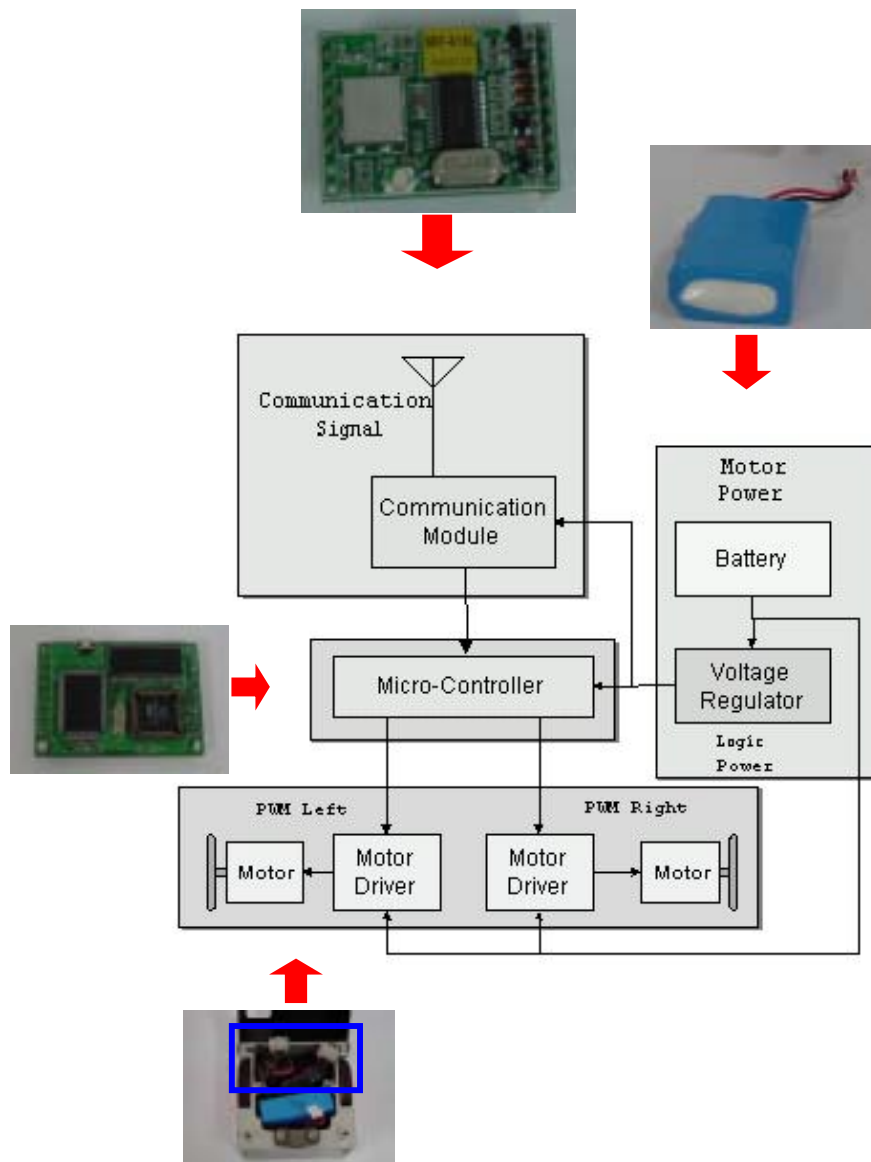


Figure 1.2 Main parts of robot system

## 1.2 Functions of Components

1. Communication part: receives communication commands from the host computer
2. Power part: makes power to operate motor and micro-controller.
3. Micro-Controller part: translates the communication commands from the host computer and makes control commands of motor.
4. Motor part: impresses electricity to motor as much as wanted by commands of motor.



The micro-controller was made used of AVR ATMEGA 128 so that the users can modify enclosed program or make out a new robot program for themselves.

The voltage of charge battery is 8.4V and it is used as the power of motor. 8.4 V comes down to 5V through voltage regulator (7805) and it is used as the power of logic circuit. The gear ratio of motor is 8:1 and an inscribed gear is used. Encoder is included in the motor and its resolution is 512 pulses / per one revolution. The encoder makes the images clear as 4 times and receive 2048 pulse.

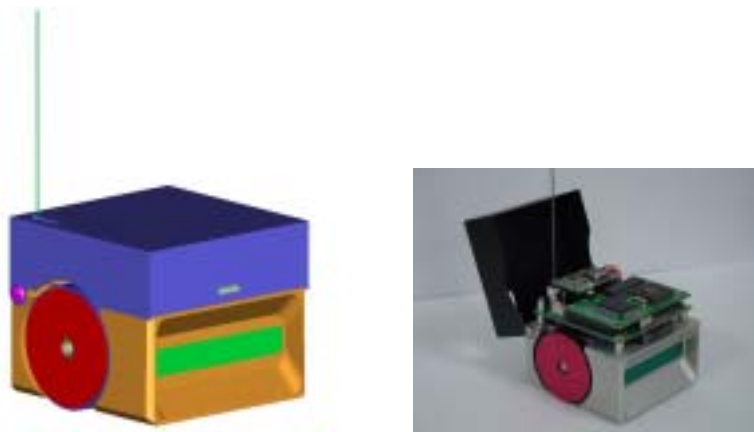


Figure 1.3 frame of YSR-A robot

### 1.3 Robot system based on vision

Robot soccer can be divided into two groups; robot soccer system based on vision and robot soccer system based on robot.

Robot soccer system based on vision can be grouped into remote-brainless soccer robot system and brain-on-board soccer robot system. The most of implemented robot systems use one of among the three methods or a hybrid method.

The trend of soccer robot system is vision based remote-brainless system. This system has no intelligence in the robot and it just follows instructions from the host computer.

In general, most robot has a controller to control position and other things. However, vision based remote-brainless system has the controller in the host computer. For accurate control, fast sampling time is required. The time of image processing, control algorithm, and communication has to be fast. Therefore, high performance image processing system and host computer are required.

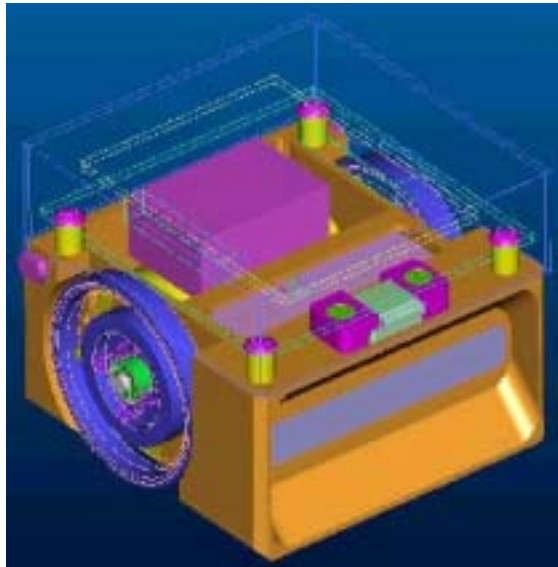


Figure 1.4 Modeling of YSR-A

## 1.4 Etc.

In an YSR-A robot, the robots, RF (Radio Frequency) module, and the video card are provided. Beside these hardware, several other things are needed for the robot soccer system. First, you need to have a frame grabber card, we recommend you to use METEOR - II from Matrox. Also you need to have a VGA card as AGP type with Ram of more than 16 mega to run our vision program.

The following frame grabber card can be used.



Figure 1.5 Matrox Meteor 2 card- frame grabber card

For the camera, a CCD camera with zoom lens should be used for the playground to fit into the view area of screen. SAMSUNG Digital Color CCD Camera may be used as the CCD camera. The model number is SDC-410ND



Figure 1.6 Samsung Camera CCD-410ND

The camera stand is 2 m or higher in height, and to display the whole area of the playground on the screen at this height, a lens with a zoom function has to be used. The zoom lens's focus length must be between 3.5mm and 8.0mm and it must have a manually controllable iris. By controlling the iris, the brightness of the image on the screen can be set to a proper level of brightness even though the brightness of the surrounding environment around the game field is changed.

The official dimension of the playground is on the FIRA web site ([www.fira.net](http://www.fira.net)). If the playground is not available, you may build your own playground just for the practice based on the information given on the web page.

In addition, a battery charger and a power supply for the transmitter are required. Input voltage for the transmitter should be more than or equal to 9 V, and a current of 300mA of DC power supply is needed. A power supply will be supplied with the YSR-A package. YSR-A soccer robot system provides the battery charger., Li-Ion batter charger. The model is AIDI-510 and its input voltage is AC220V 60Hz, and output voltage is 8.4V, 800mA.

A piano string can be used as the antenna. The length is 16.5cm approximately.

## 2. How to Set Camera

- To let the color of attached color patch see clearly in vision program, please see the following way to set the camera and lens above the ground as below figure.

As you can see the below suggested procedure, these are descriptions of camera setting for smoothly progressed robot soccer games.

The sequences are same as following,

**Setting of Camera function -> Settle Camera -> Setting lens of camera in best fitted**

\*\* To keep in mind, you have to set up camera & lens certainly at most with the image on the screen as same as you can see in your eyes. So please make sure this point, and set it up.

### 2.1 Camera Setting

#### 1) AGC S/W off

- AGC (Auto Gain Control), a function to add some regular values automatically according to brightness of input images
- Why AGC to be off?

In robot soccer, vision is a system to be calculated just as it recognizes the image. If in use of AGC, the color of the patch will be changed on the screen and will be some error in vision process.

- How to set AGC to be off?

#### (1) Separate Camera case



Figure 2.1 Positions of Case segregation bolts

(2) Put the switch inside to be off



ON -----> OFF

Figure 2.2 Switch to be off

(3) Attach the case again

- **Notice: please pay attention not to be damaged of parts inside during fixing.**

## 2) W.B menu

- What is W.B (White Balance)?  
W.B handles R, G, B values according to brightness of the surroundings when the image is input on the camera.
- The reason to set by manual?  
=> Vision system of soccer robot is the one recognizes colors. The systems that recognizes colors are also important to let the image see clearly, but there can be happened that vision system doesn't operate if the colors are changed because of the luminosity. If the system is set as Auto-operation, changes of colors according to luminosity around the surroundings. To avoid this case, you have to fix the system for itself to get same color even if the changes of the surroundings' luminosity by manual.
- How to set by manual  
During setting the camera, there can be a case that you will see the screen as blue totally. The reason why the screen appears as blue, W.B menu is not well fixed. At this time, put 'R' button as below figure.

\* SEE VOD \*

Screen captured: blue -> normal status (refer "[Setup\(WB\).exe](#)")



Figure 2.3 W.B Menu

(1) Setting/check optional menu of camera

⇒ For the setting of functions for camera should be formed as below figure. If you finish as below figure, it means that you set everything for all the functions.



Figure 2.4 Menus for Camera (Light Green part: optional menus)

## 2.2 Setting up the camera

1) Unit of Stand:

⇒ The sequence to bond each stands is same as below figures.

- 1) Stand peak stick combination
- 2) Reversed 'L' shape stick combination with (1) peak stick
- 3) 3-legs stand combination with (2)
- 4) Connecting BNC cable to power of the camera
- 5) Sticking Camera on top of the stand
- 6) Executing program

## 2) Setting position of camera

- Fix the camera to be set up above from the correct center of ground.



Figure 2.5 (Front position)



Figure 2.6 (Side position)

- Set the guide for luminosity of lens by taking turn the guide for the screen to be brighter.

\* SEE VOD \*

Dark Screen / Bright Screen (refer "[Setup\(color\).exe](#)")



Figure 2.7 Guide of Luminosity

- Set the guide for zooming by taking turn for the screen to be seen full with ground.



\* SEE VOD \*

Unfit Zooming / Best fit Zooming (refer "[Setup\(Zoom\).exe](#)")



Figure 2.8 Guide for Zoom

- Set the guide for focus by taking turn for the screen to be seen clearly.

\* SEE VOD \*

Improper focusing / Proper focusing (refer '[Setup\(Focus\).exe](#)')



Figure 2.9 Guide for Focus

**\*\* Notice for setting position of camera**

In case the screen is seen to be inclined in one side, or shaped as echelon formation.

➔ Camera should be set up at the right angle.

## 2.3 Setting Lens

**Before you fix the lens of camera, you have to put the enclosed fluorescent color patch on the ground. So it will be much easier for you to set the proper status watching the color.**

- 1) Put the provided fluorescent color patch on the ground.
  - 2) Choose 'Color Tuning' menu among menus of program. Click 'Default' so that all values are to be 127.
  - 3) Handle the guide for brightness of lens.
- ➔ Please see the marked numeral data using 'Set Color' button.
- ➔ (The numbers don't need to be exactly same, but below numeral data will be seen if the given condition of the place is similar to the sample. Please check for your reference.)

**\* Condition of place for numeral measurement**

**\* Degrees of lighting equality: place in room light with no change of brightness**

**\* Luminosity: 1200 Lux**

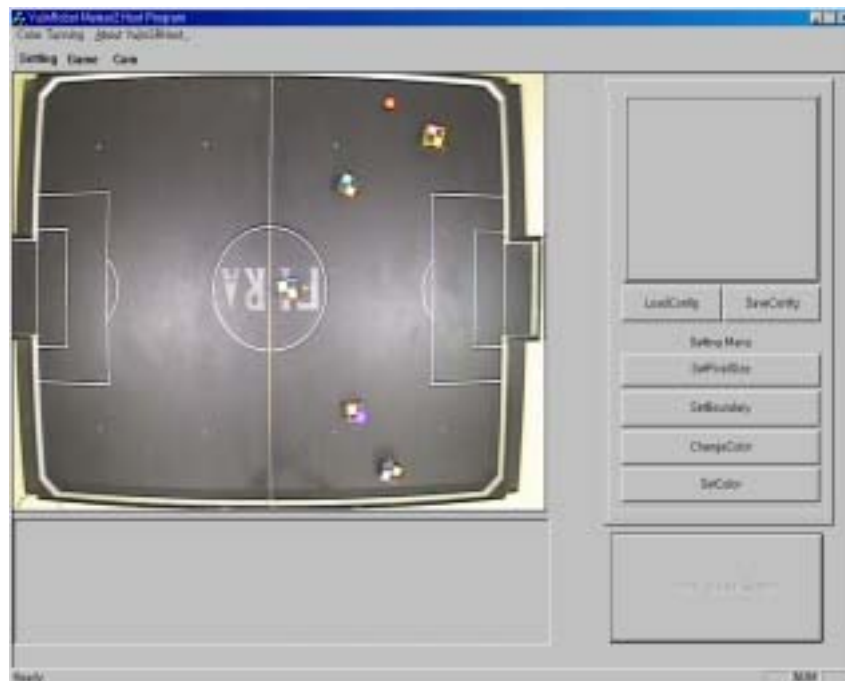


Figure 2.10 In case of high Luminosity (Bright)

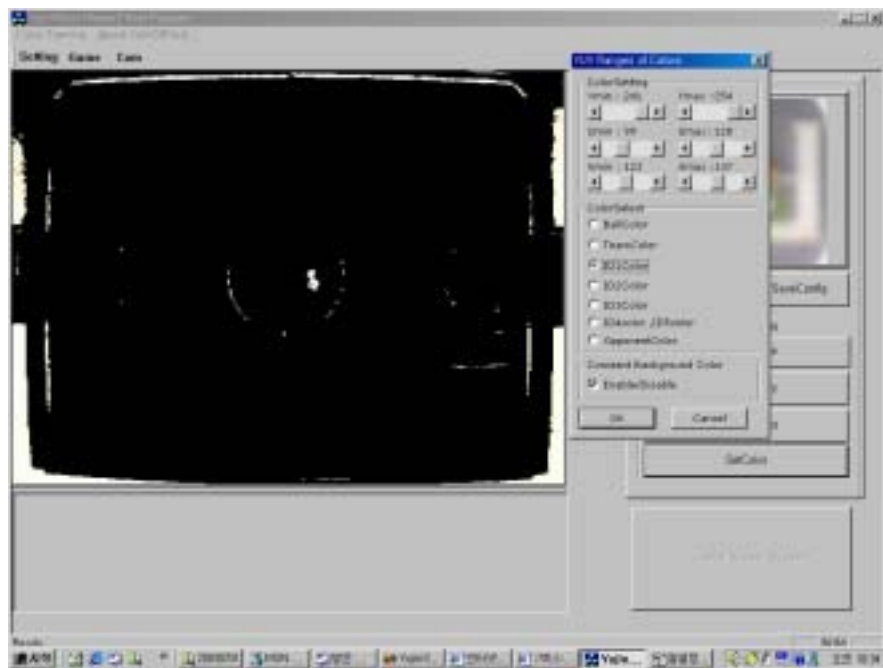


Figure 2.11 Screen data after click SetColor in case of high Luminosity

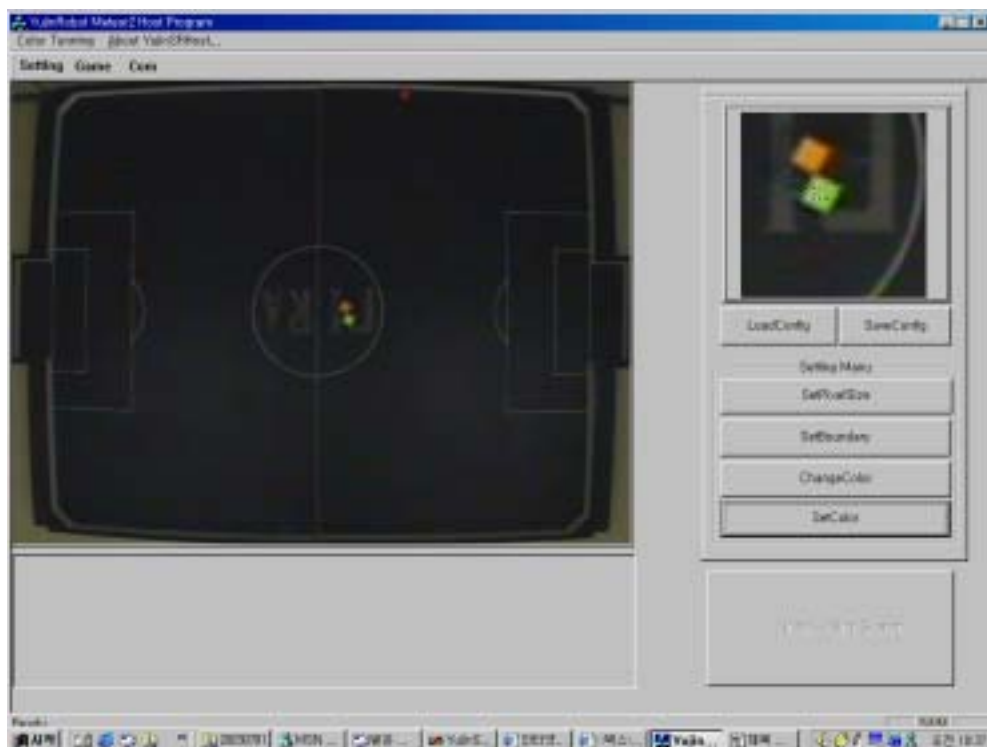


Figure 2.12 In case of low Luminosity (Dark)

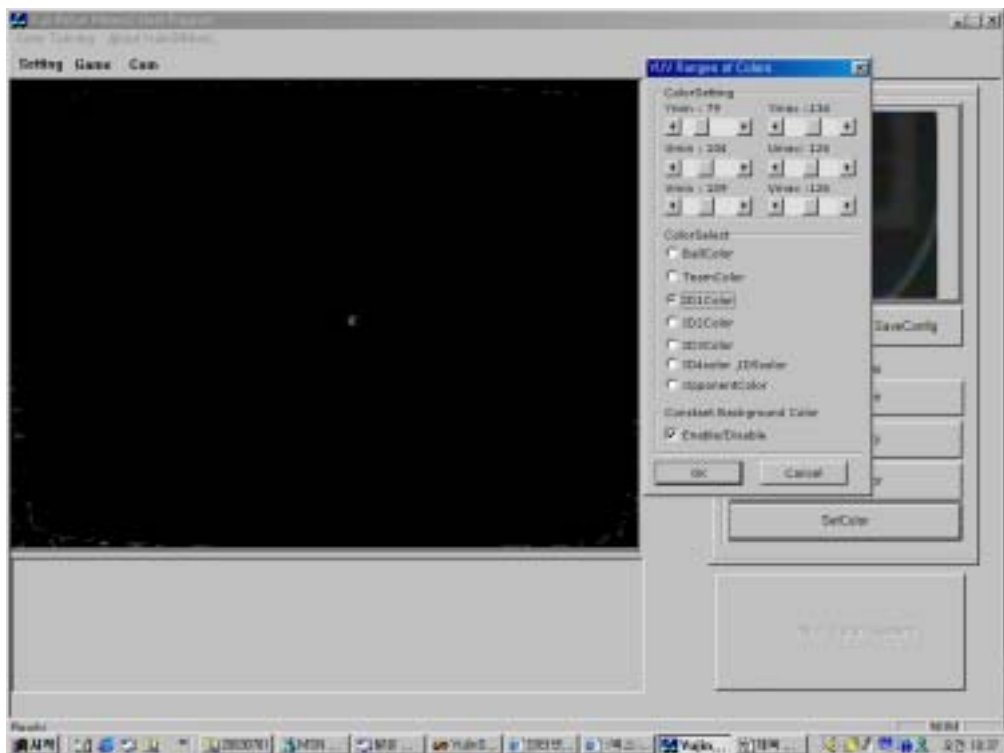


Figure 2.13 Screen data after click SetColor in case of low Luminosity (Dark)

- In case of proper setting -

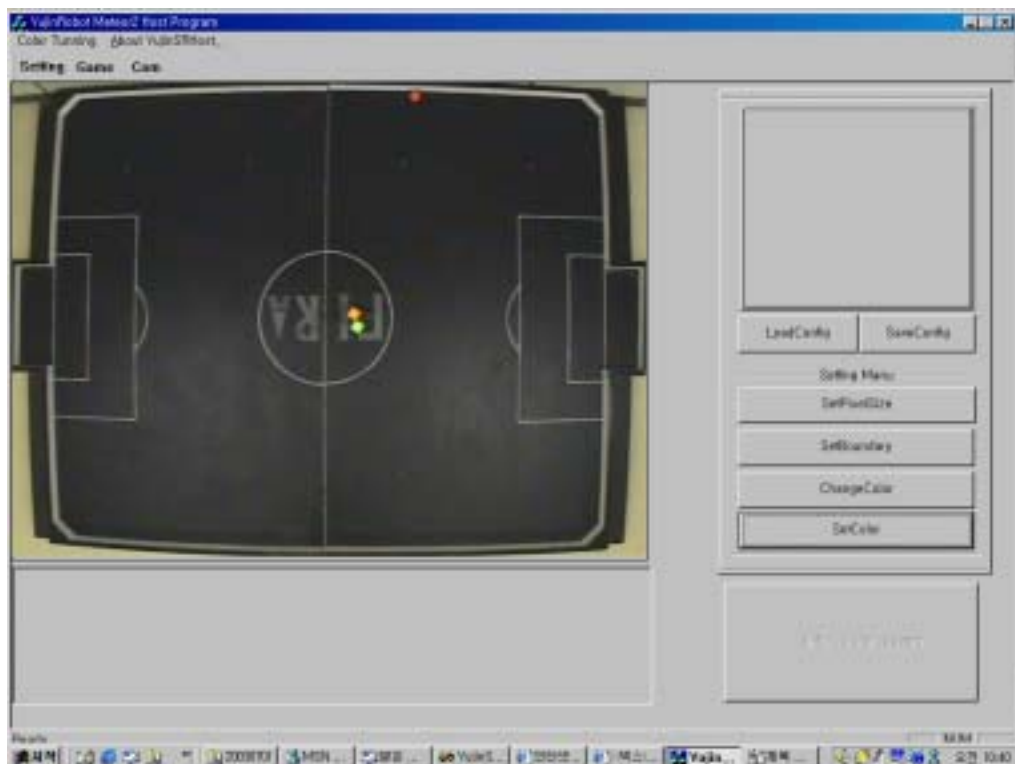


Figure 2.14 In case of proper setting

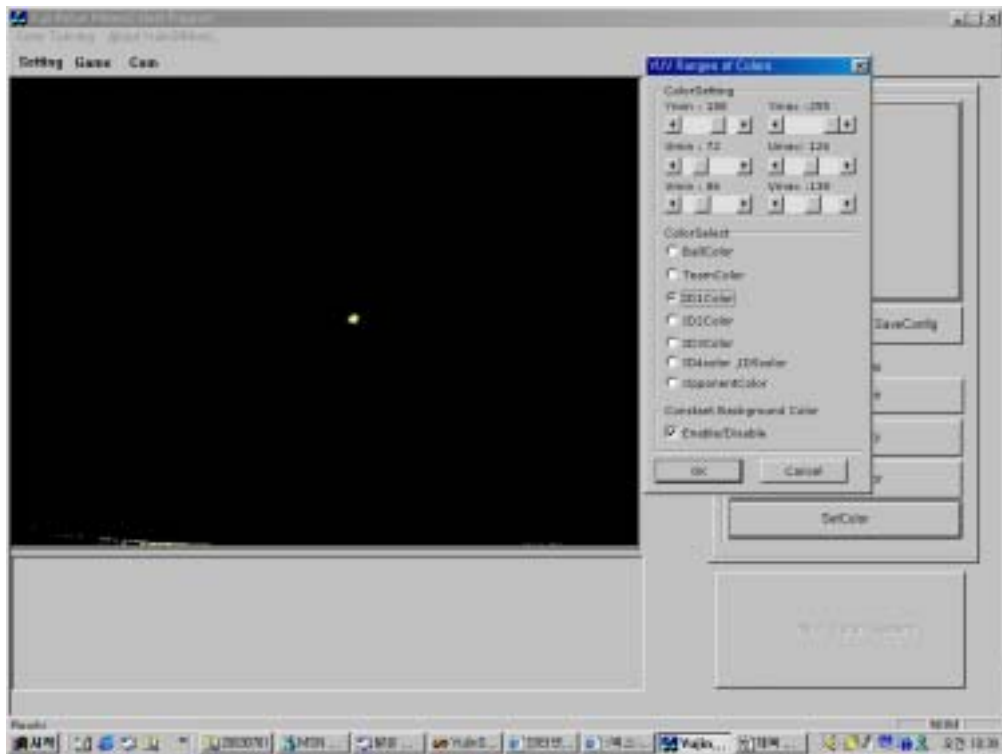


Figure 2.15 Screen data after click SetColor in case of proper luminosity (Best)

\*\* The conditions can be highly different only adjusting luminosity guide of lens.

## 3. Robot test program

### 3.1 Robot test program

To test whether the robots work properly, you need to set the robot's ID correctly. The settings for the robot's ID are as follows. Figure 3.1 is an actual example of setting up for ID while the upper part of the figure is the front of the robot.



Figure 3.1 Setting of Robot ID

There are three dip switches on the bottom right side of Figure 3.1. In the dip switches, #1 to #3 are used to select ID of the robot. As three robots shall play a soccer game simultaneously, the status of OFF for dip switch #1 means robot #1, and the status of OFF for dip switches #2 and OFF for #3 means robot #3 respectively. Be careful not to be OFF more than two switches concurrently.

	DIP SW #1	DIP SW #2	DIP SW #3
Robot #1	OFF	ON	ON
Robot #2	ON	OFF	ON
Robot #3	ON	ON	OFF

Table 3.1 Setting's for Robot ID's

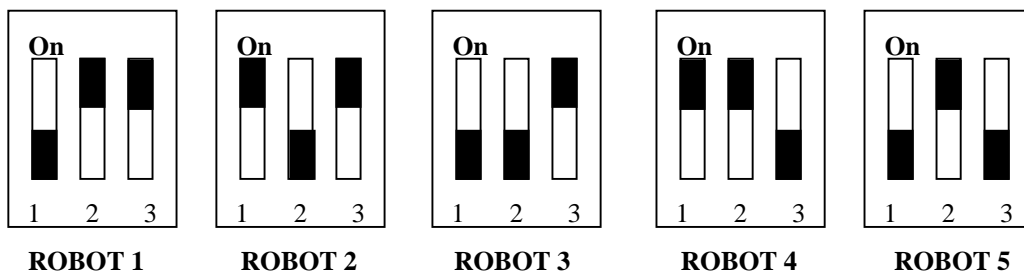


Figure 3.2 Dip Switch

After setting up the robot' IDs, you may test the robots with using RF communication (wireless communication) on the vision program next to this chapter.

If one motor does not rotate, and the other does operate well, you have to check if there's any problem between motors.

First, put the power off of robot, and try to rotate the wheel does not move by hand. If the motor does not rotate, please look into its gear part if it was well assembled and try this again. After doing above, if it does not work, separate the motor cable and put the power directly to the motor and check if it works or not.

Then make the speed 127 and if the wheels does not rotate as fast as it can after clicking 'Send Data' button or "Start" button, the battery connected to robot is not a good condition. Therefore please replace it as a newly fully charged battery.

Even though you have already checked as above, if the robot does not move, let's check into domestic team, each robot's ID JUMPER setting and part of communication.

Inside the transmitter, there is a micro-controller named as 89C51.

When sending data from host computer to transmitter, it is usually deleted of 0xAA following behind PWM data. If the host computer already received all the data of one team, the transmitter sends 0xAA (\* 0xAA should be sent during 3 ms before and after transmission to open a communication track.) together behind PWM data of each robot during transmitting through RF module.

When you test the robot with serial cable using RF module and if the robot still does not work, you have to check if the transmitter is power on with LED light, and its input voltage is more than 9V (\* Model 7805 should have a voltage more than 9V.)

Also you can see the LED light of transmitter is flickering so fast when the host computer sends data to COM Port. It means that there is no problem for communication between host computer and transmitter.

## 3.2 RF communication module

To play robot soccer, the computer should send out instructions to each robot using wireless communication. There are two methods to accomplish wireless communication; RF (Radio Frequency) module and IR (Infrared Ray) module.

IR module is sensitive to the external light. If a competitor uses IR module, it results in increasing communication load. IR module is used to educational purpose.

RF module uses radio wave and it is suitable for long distance communication. There is no communication barrier between the host computer and the robot. Also it supports multi channels. However there is mutual interference between CPU, sensor, motor drive, and RF high frequency communication. It makes EMI noise. Therefore, you should have measures. Also, the circuit of RF module is complex and know-how of manufacture is needed. So, it is difficult to make RF module. Consider frequency according to the law and avoid frequencies of the broadcasting, mobile phone, and pager.

Most teams of robot soccer use the commercial RF module. There is RF module that is developed to digital communication and it guarantees high speed of communication and reliability. But it is expensive and doesn't support various frequencies. The widely used RF module is RF-418 and RF-433. It is able to send out data up to 40Kbps. The following transmitter information is specification of RF-418 and RF-433.

- Duplex Transmission data at up to 40 kbit/s
- 10 channels practicable
- Speed of data transmission: 19200 BPS
- Reliable 30 meter in-building range and 120 meter in open range
- -107dBm receive sensitivity
- Single 4.5 V to 5.5 V supply < 15mA (tx or rx)

The actual RF module is illustrated in Figure 3.4 and 3.5.





Figure 3.4 Commercial Communication Module

As shown in the above figure 3.4, you can do bi-directional communication, such as same manner above, after setting each robot's ID, you have to test if the robots work properly. As a way of testing the robots, it is serial RF communication (wireless communication) used. To use multi-channel communication module, you have to set the radio-frequency and the channel as below.

The dip switches from no. 4 to no. 8 are used for setting of channel for RF-module. If the switch no. 8 is down to the mark 'ON', it is SRF-418 module. And if it's opposite, it's SRF-433 module.

	SW 4	SW 5	SW 6	SW 7	SW 8
CH 1	OFF	ON	ON	ON	ON
CH 2	ON	OFF	ON	ON	ON
CH 3	OFF	OFF	ON	ON	ON
CH 4	ON	ON	OFF	ON	ON
CH 5	OFF	ON	OFF	ON	ON
CH 6	ON	OFF	OFF	ON	ON
CH 7	OFF	OFF	OFF	ON	ON
CH 8	ON	ON	ON	OFF	ON
CH 9	OFF	ON	ON	OFF	ON
CH 10	ON	OFF	ON	OFF	ON

Table 3.2 Settings of Channel through Radio Frequency

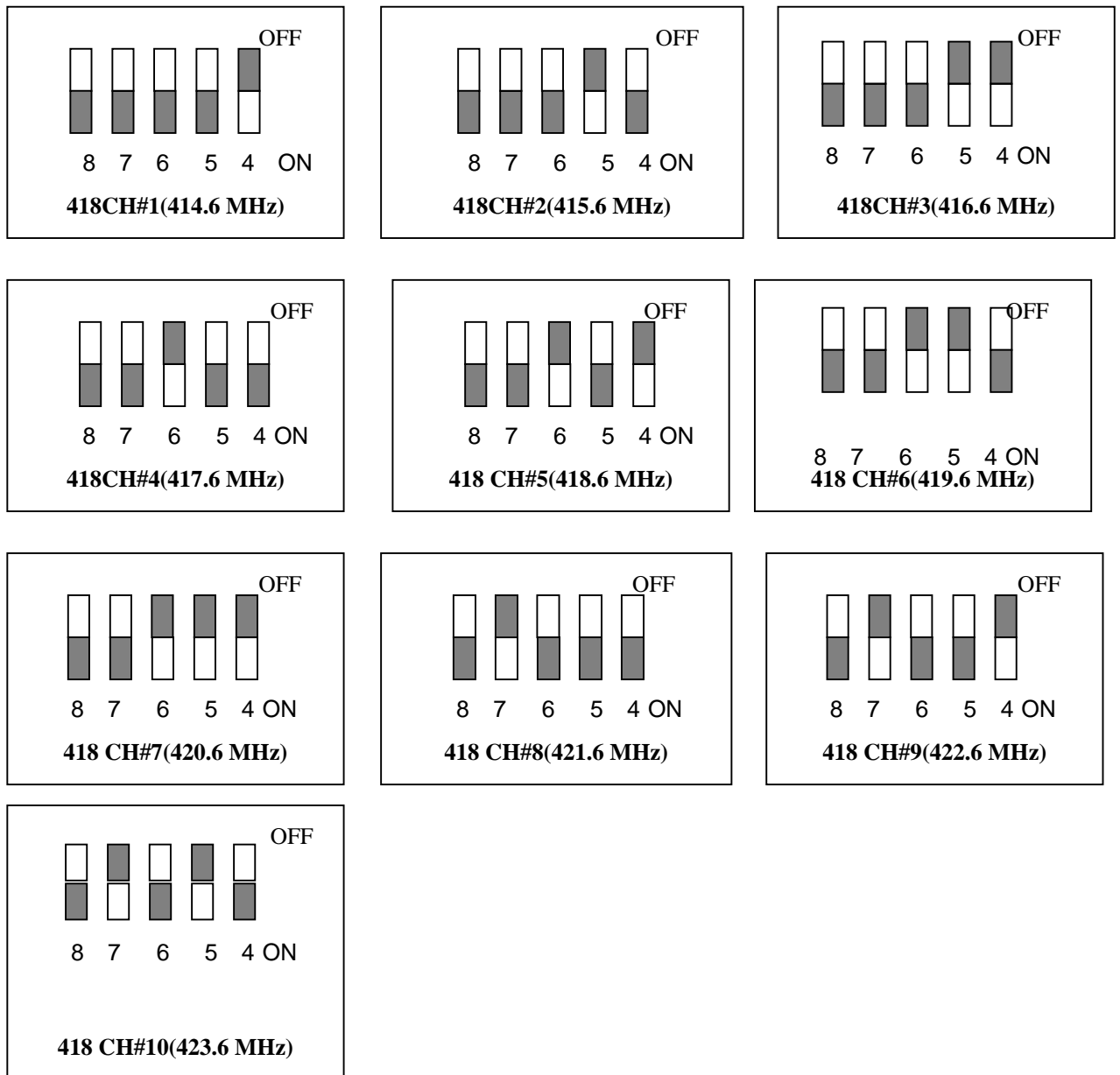


Figure 3.5 Channel Setting for Transmitter & Robots

## 4. Installation and Execution of Vision Program

### 4.1 Vision Program Installation

#### - MATROX METEOR 2/4 Installation Guide

Before you start installation, you have to make sure if your computer environment are set by Win 2K/32 bit color/1024x768 (It is changeable because we're updating so you can also check with our homepage <http://www.edrobot.com/> )



Figure 4.1 Matrox Meteor 2/4 card

1. Install Matrox Meteor 2/4 Board to the empty PCI slot.

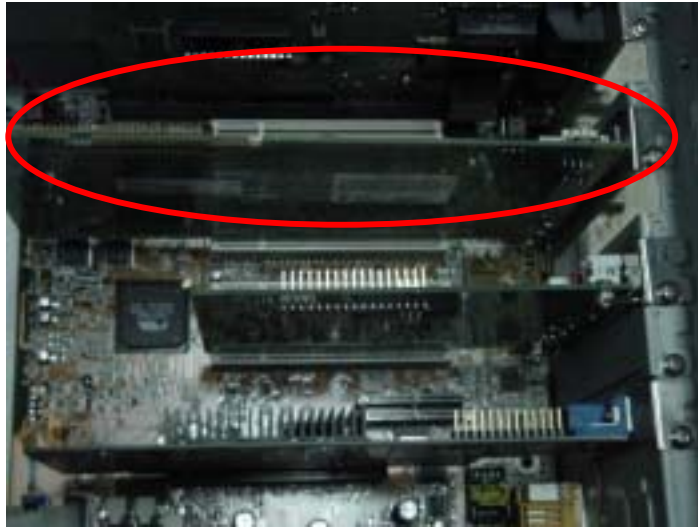


Figure 4.2 Placing Matrox Meteor 2/4 card in a PIC slot

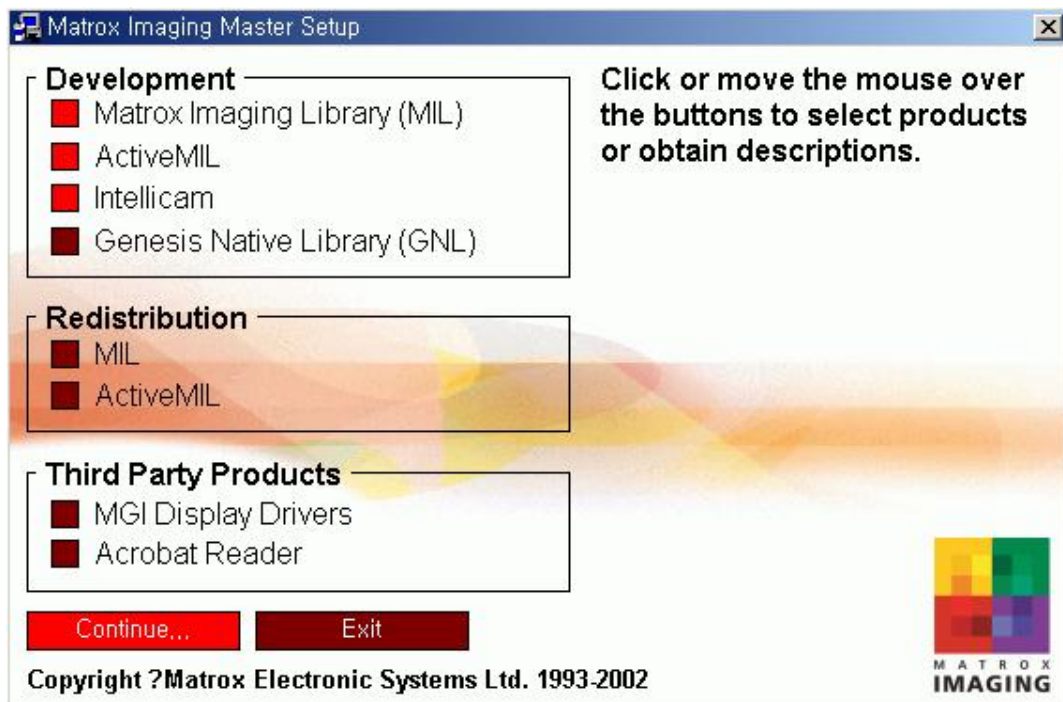
2. Connect the camera cable to its connecting terminal at the back of Matrox Meteor 2/4 card.



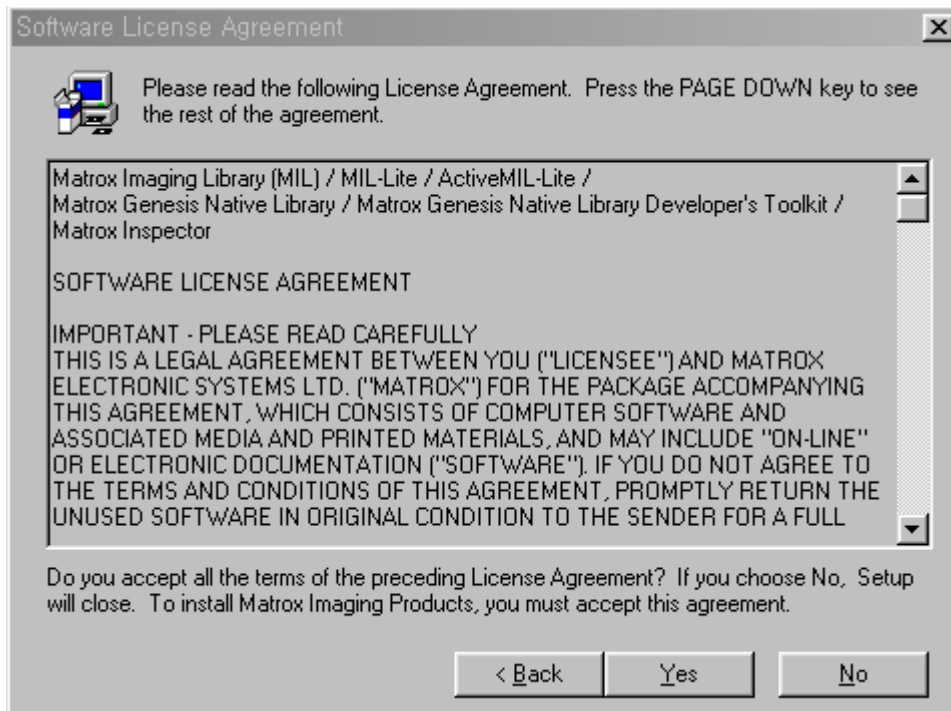
Figure 4.3 Connecting Camera Cable

3. Insert Installation CD to CD ROM DRIVER.
- Notice : before you insert CD, you will find the notice that new device was found when you turn on the computer after installation of Matrox Meteor 2/4 card. In that case, click cancel button and then insert CD to install.

4. By autorun, you can see “Matrox Imaging Master Setup View”(MIL-LITE 7.1 Setup view) in a few seconds.
  - By autorun, and you can see Setup view. If your CDROM does not support Autorun, double click **Setup.exe** in the root directory of the Install CD.



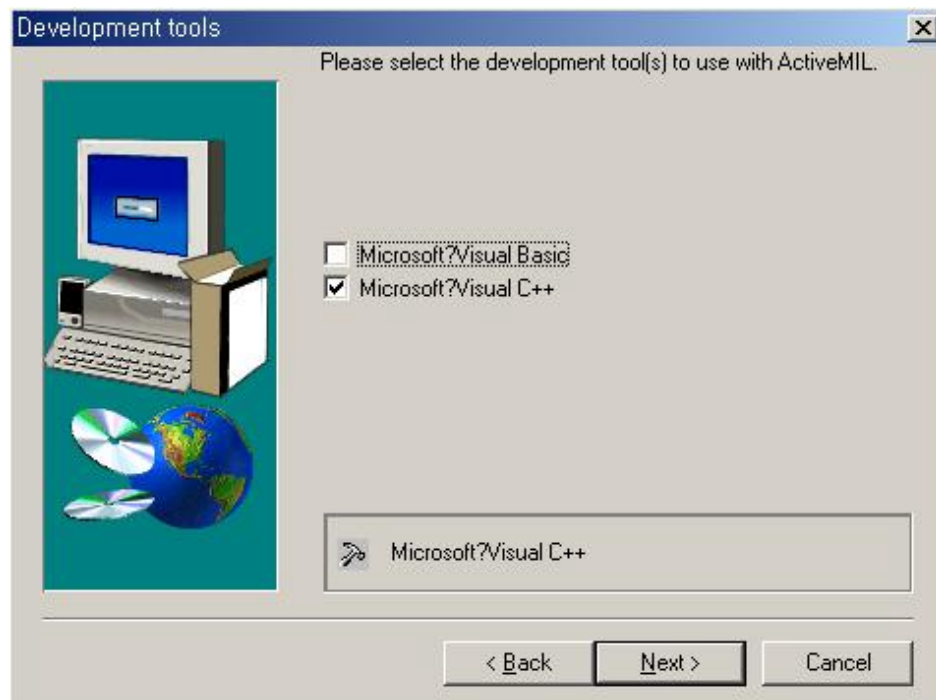
5. Check MIL, ActiveMil, and Intellicam in Development menu, and then click Continue... button. .
6. Click Yes button in the software License Agreement View.



7. Chose the destination folder to be installed, and click Next Button.  
default directory is “c:\Program files\Matrox Imaging”



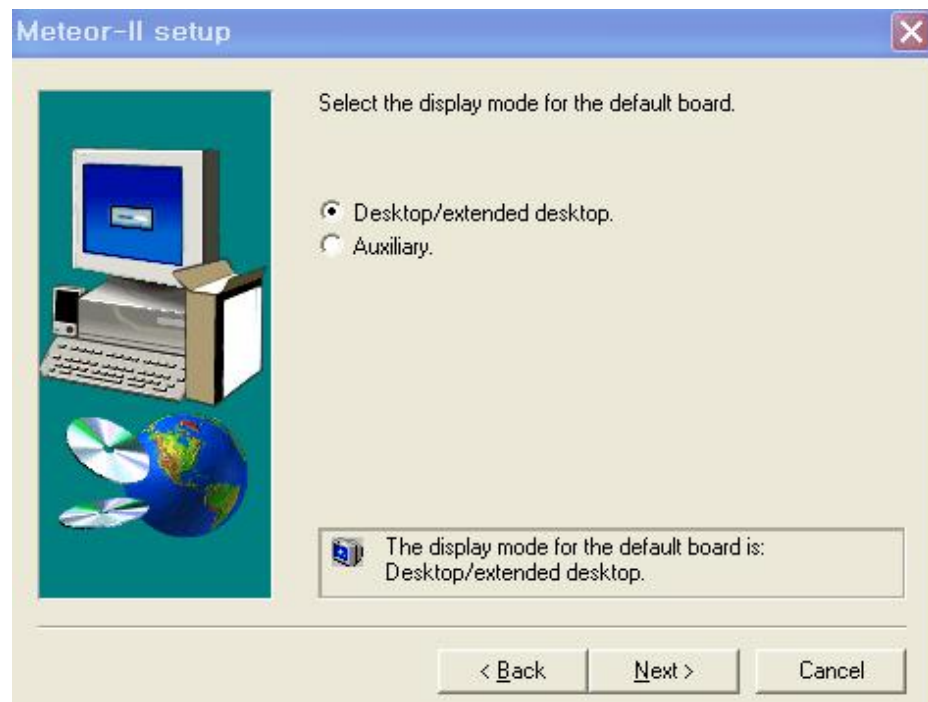
8. Chose your compiler.  
Visual C++ users need to check Microsoft Visual C++, and click Next> button.



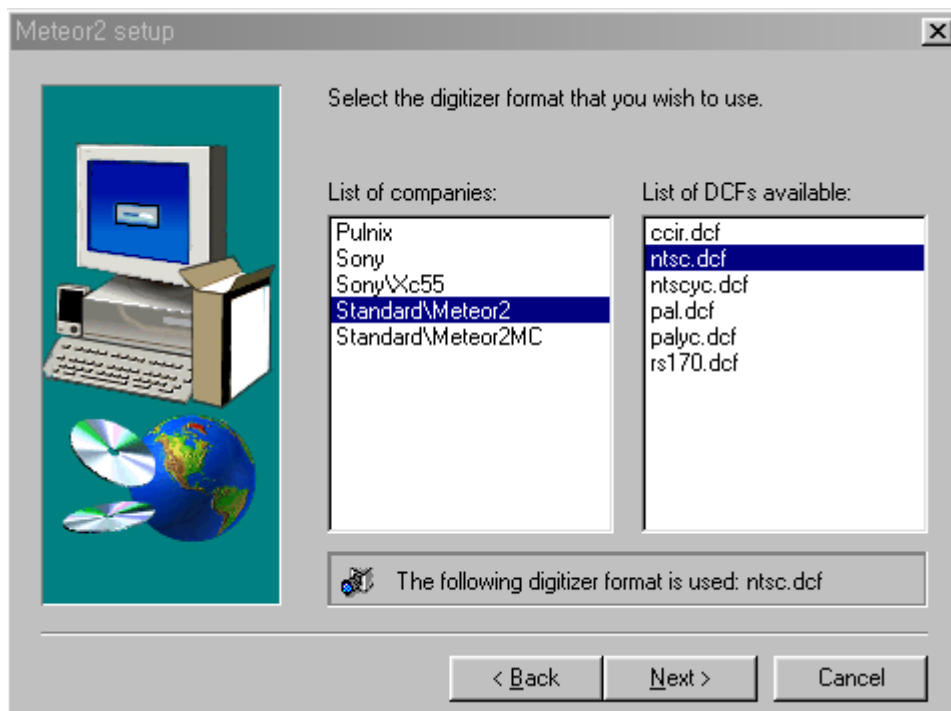
9. Then, select the board driver Meteor-2/Standard & VGA card as below figure and click **next>** button.



9. You will find next figure that is asking the system used for Frame Grabber. Select Desktop/extended desktop and click **next>** button.



10. Chose your default DCF and click Next> button. As shown in below Figure.,  
the provided grabber is Standard\meteor2 and DCF is **ntsc.**

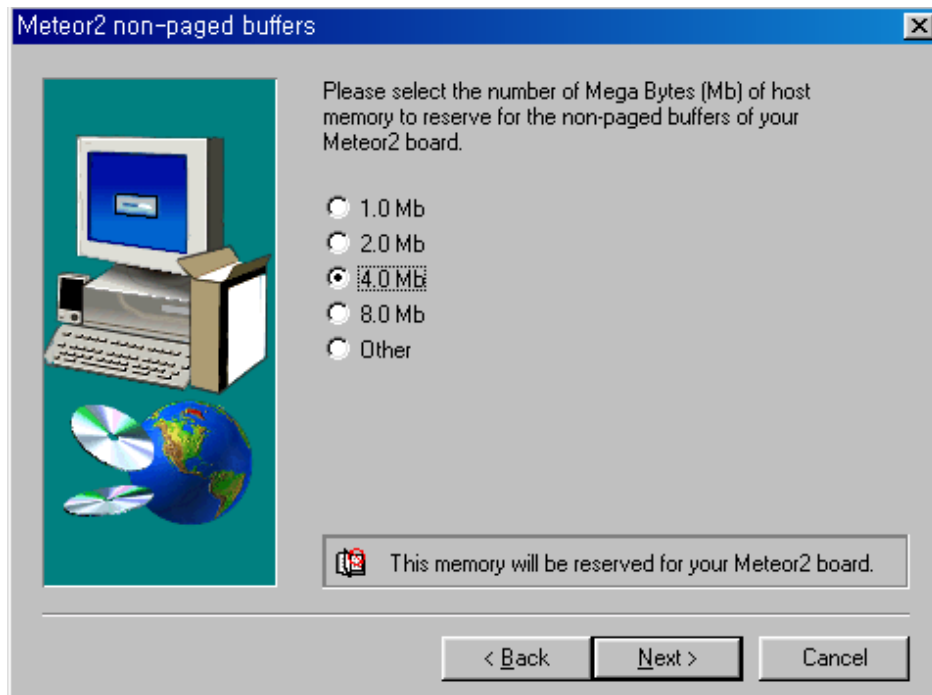




11. If you have more recent driver than ver. 7.1, you can check yes. Or if you do not have any new driver or this is your first installation, check No.



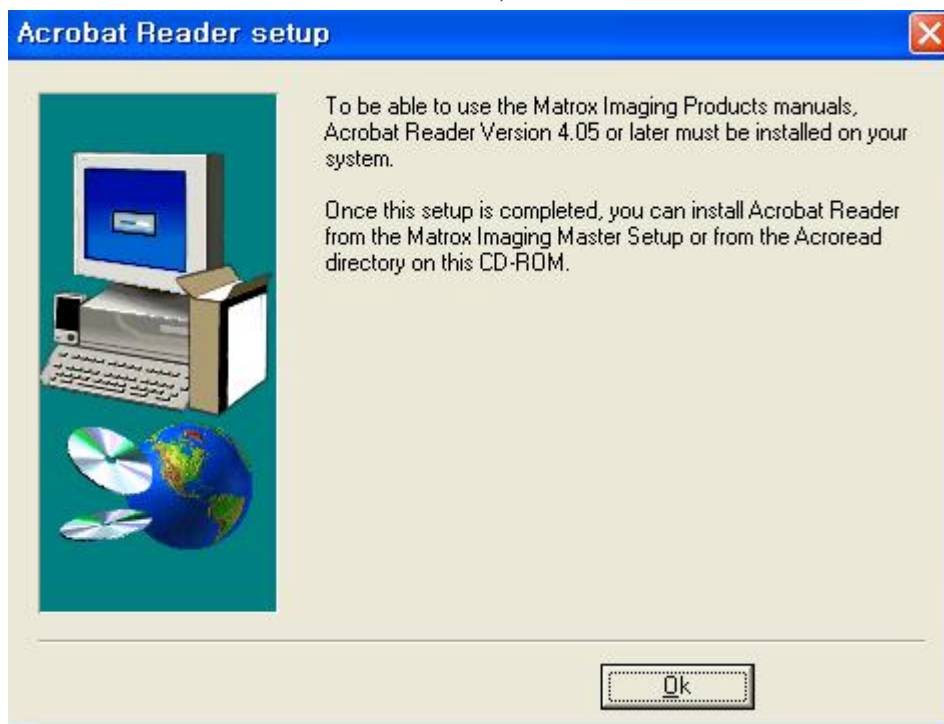
12. Check Host memory's space that will be reserved for your Frame Grabber. Check 4.0Mb and click Next> button. Because this memory is reserved only for Frame Grabber(Meteor 2), the other applications can not use this memory. (You will find these reserved memories can be check and the system memory will be decreased in the booting menu.)



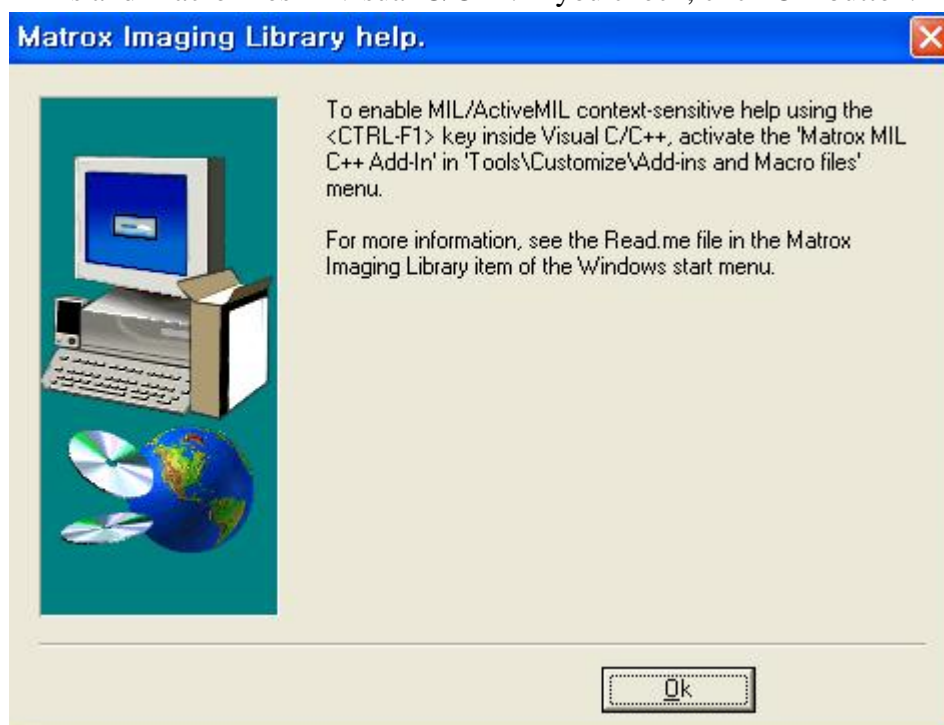
13. Then, the setup program shows all settings. You can change any settings or start installation (click Ok).



14. After all the installation, a dialog box will be appeared to check Acrobat Reader Ver. 4.05 for Matrox Imaging Products Manual. If you already have Acrobat Reader more than ver. 4.05, click OK button



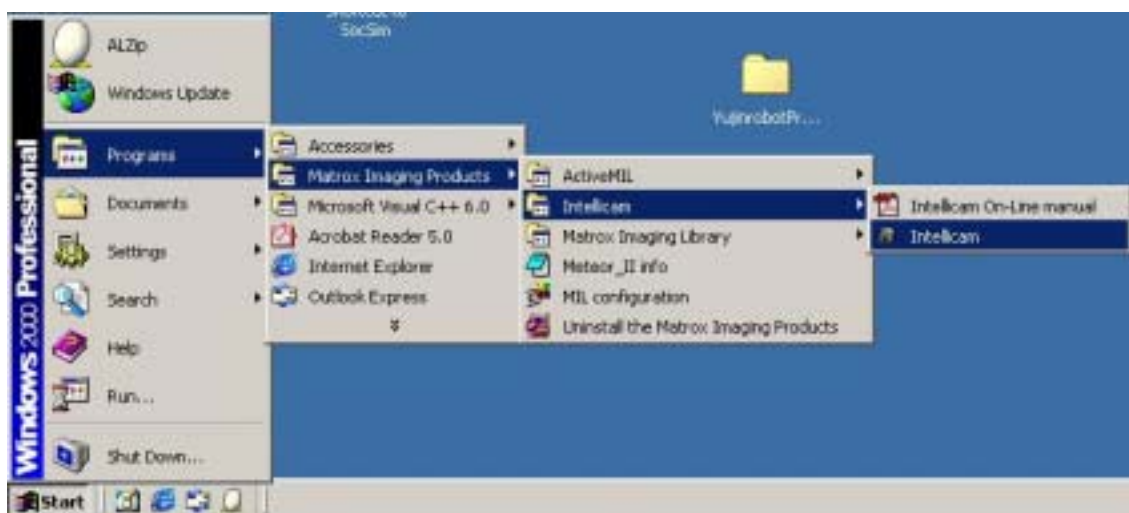
15. Then, below figure will be screened if you need any help for installation of MIL/ActiveMIL, click , <CTRL/F1> key, and look for Matrox MIL C++ Add ins and Macro files in Visual C/C++. If you check, click OK button.



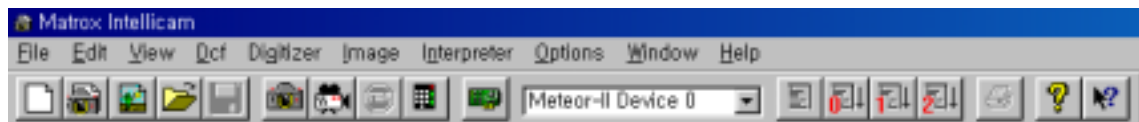
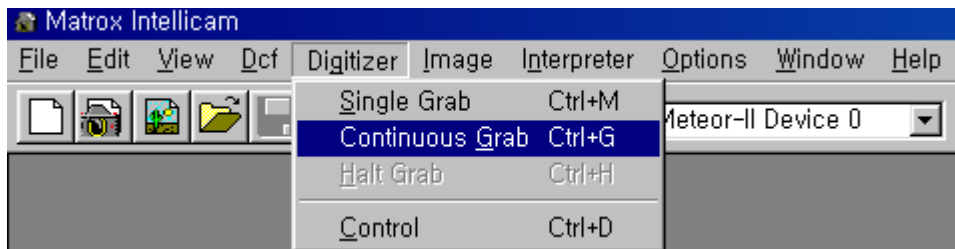
16. After all above procedure, the dialog if you want to reboot the computer and click Yes button, and Finish.



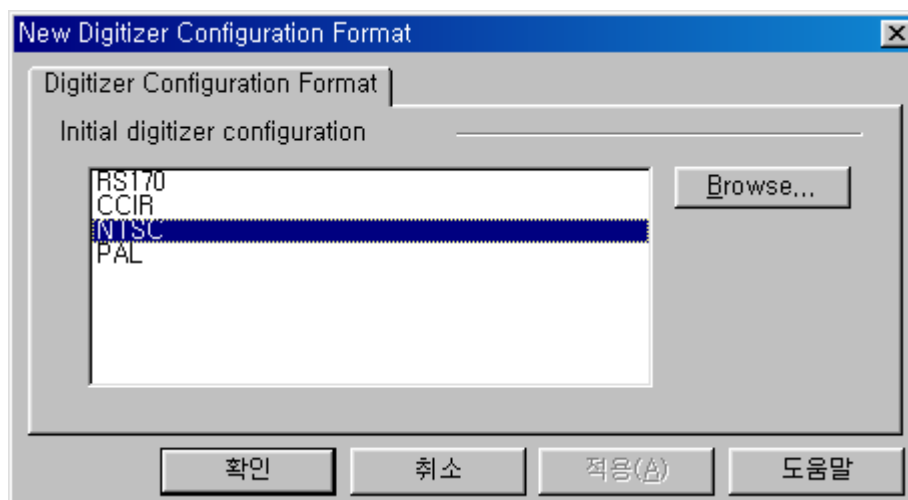
17. After rebooting the computer, you can check if the installation is completely finished. Through executing of the program named Intellicam in the Matrox imaging program in the program files of your computer.



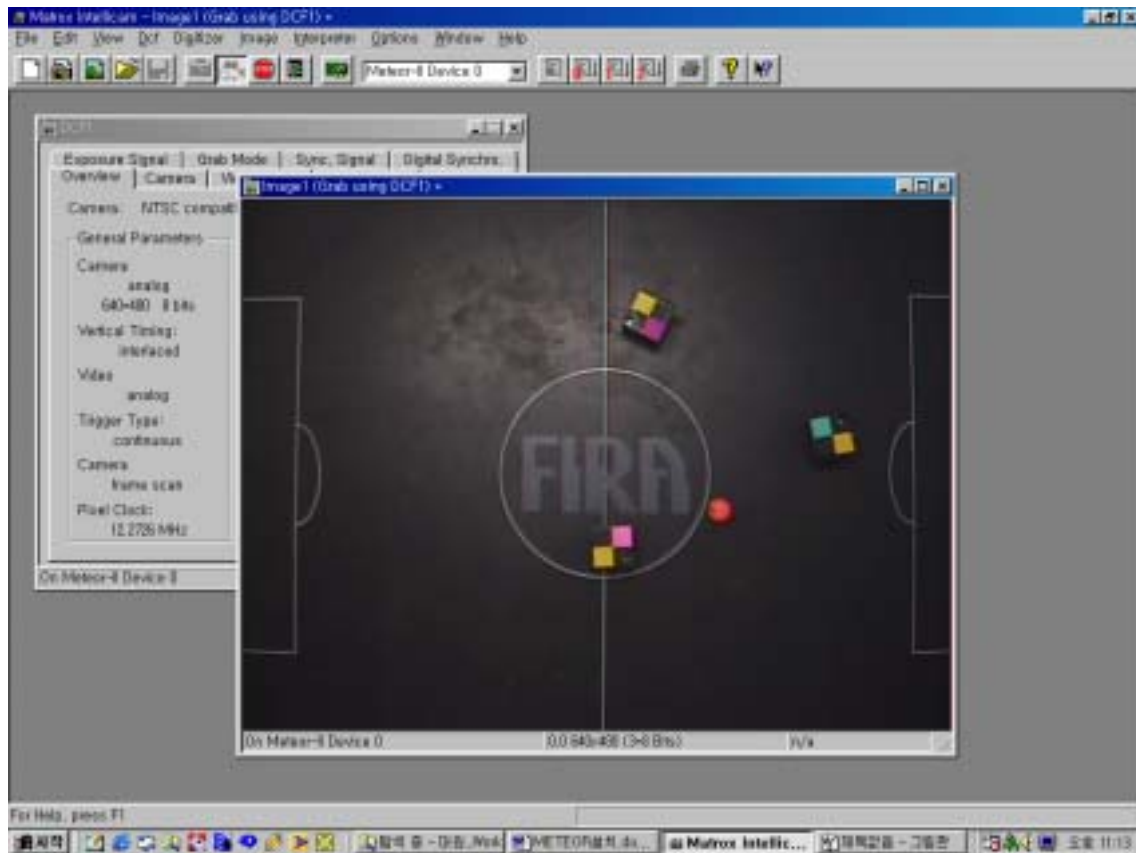
18. In the Intellicam, click Continuous Grab in the Digitizer menu, or click .



19. Click NTSC in the DCF menu.



If you can see next view receive from the camera above the field, the Frame Grabber Installation is well finished.



Install VGA Card and Frame Grabber driver. There is VGA Card Driver in the directory “:\mgadvr” of the provided CD.

The Frame Grabber Driver will be installed automatically by running MIL Lite 7.1.

## 4.2 Execution of Vision Program

There is **:\Win2k** file in the provided CD. Copy this directory to your hard disk. Select all files in the copied directory of your hard disk and see file attribute. If 'read only' attribute is checked, you need to release this attribute (make it unchecked). You can run Visual C++ by double-clicking "YujinSRHost.dsw" in the copied directory (Figure 4.4). This file is the project file of Visual C++.

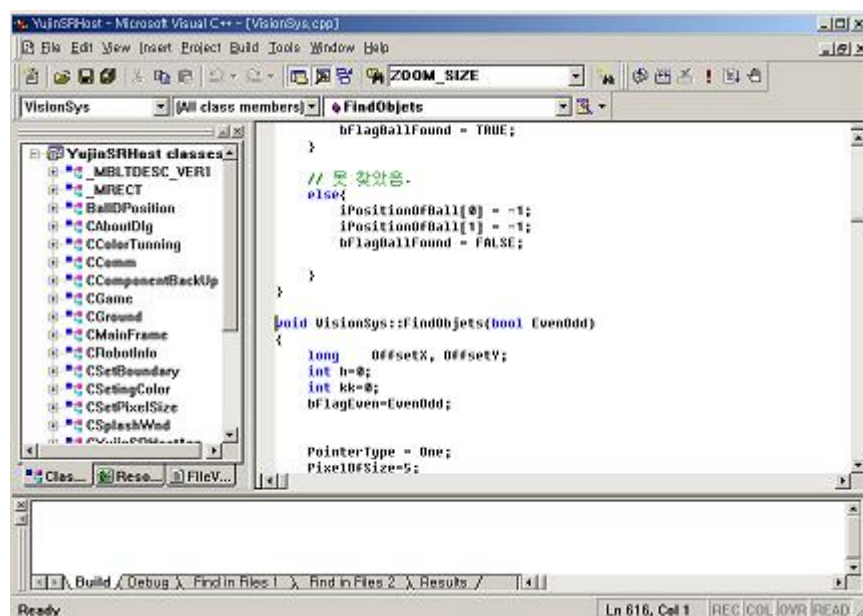


Figure 4.4 Visual C++

On the left side of Figure 4.4, many files are listed. From now on, you can learn the functions of various buttons on the vision program. Note that you should set your windows background color as **32bit color, 1024 × 768**.

And then, rebuild all and execute the program (or click )

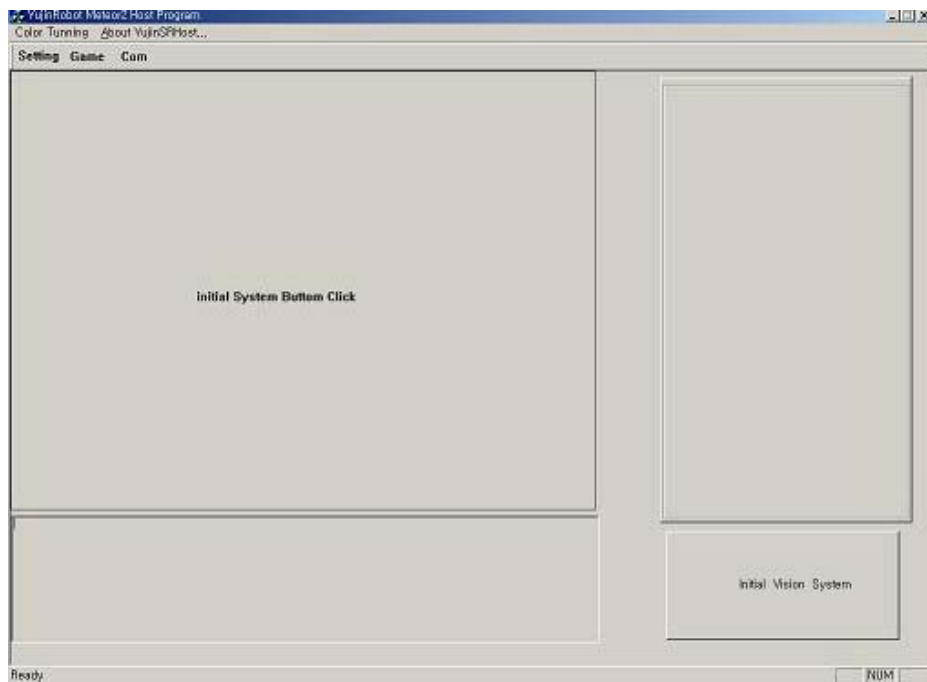



Figure 4.5 Vision Program

Figure 4.5 is the program window that pops up 3 or 4 seconds later after you click  on the Visual C++ (Figure 4.4). Then click Initial Vision System button on the bottom right and wait for a few seconds. Then you will see the screen of the ground received by camera.

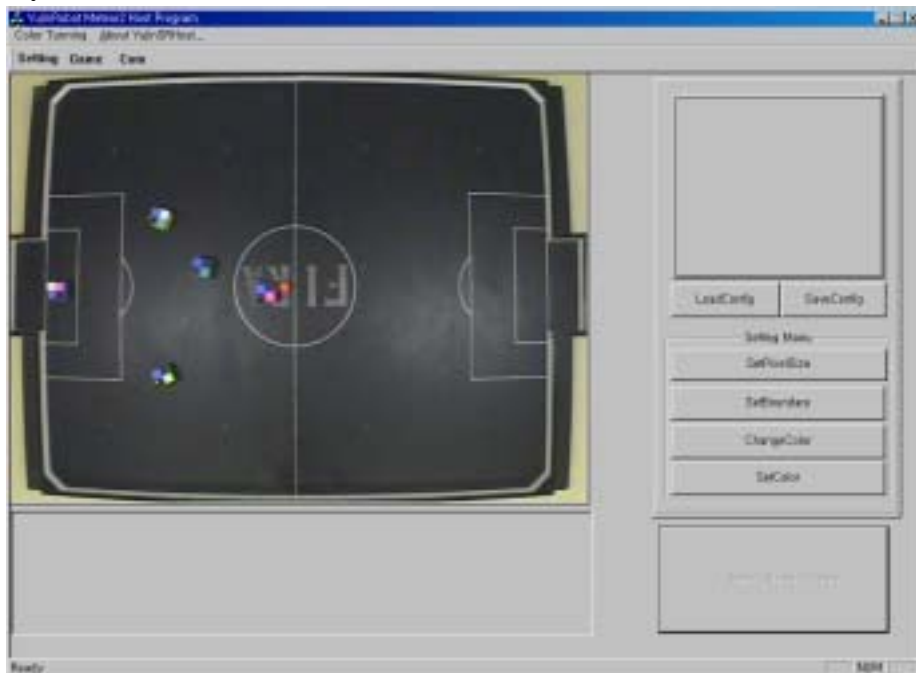


Figure 4.6 Computer Screen after Vision Program Execution



As figure 4.6, if you can see the screen of the ground, you have to set the colors for the robots and team color, and the ball's color for the computer to trace every position in the real match.

Before you set the colors, you can control the brightness of the field through Color Tuning menu on the upper window. Click Color Tuning menu on the vision window and control the bar of each section after you see a dialog box.

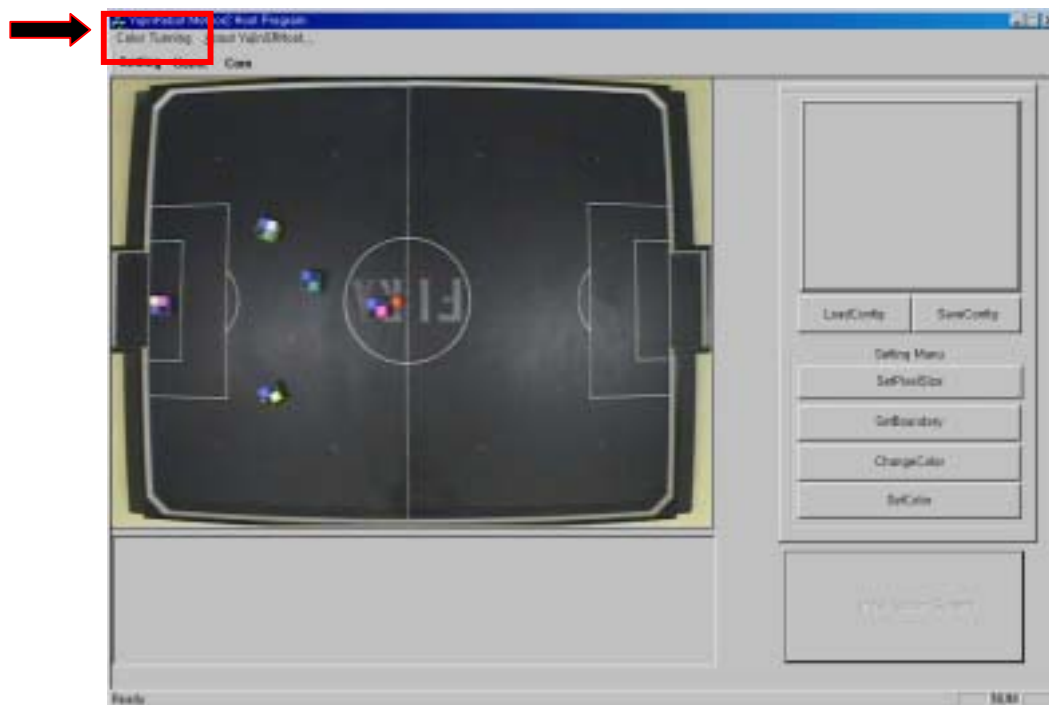


Figure 4.7 Menu for Color Tuning

If you think the screen of the ground is rather light or dark, you can control the red box menu as figure 4.7 with Color Tuning menu. As Figure 4.8, you will see a new dialog box after you click the Color Tuning menu.

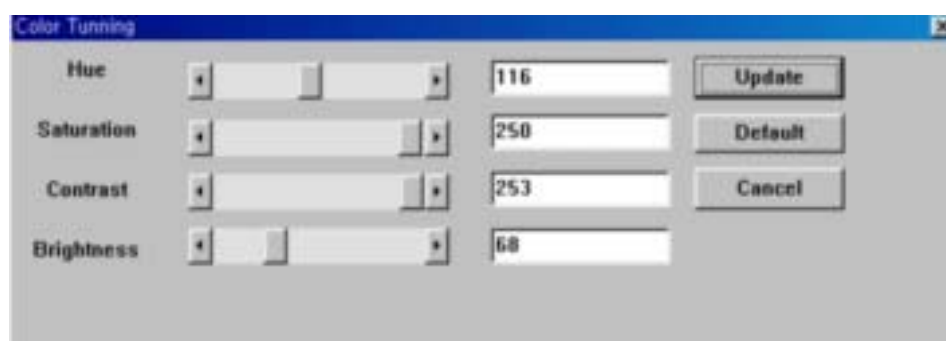


Figure 4.8 Dialog box for setting color parameter

Controlling the sliding bar appropriately, you will find the screen of the ground to be changed by your control. For example, if the light around the ground is rather brighter than the setting in the beginning, control the brightness bar a little lower so that you can solve the problem with light. For most case, you can only change the brightness values as upper or lower ones so that the color patches of the robots in the ground can be seen same as your can see through your eyes. The Default button is to change to a fixed parameter as you controlled and Update button is to keep all you have set for now, and the Cancel button is to disregard all contents you did and return to the previous working condition.

Now you need to set the ball color, robots' team color and ID color. First, to select the ball color, click "SetColor" button. And then, click the ball displayed on the screen. About three times zoomed screen will be appeared beside the actual playground screen as shown in Figure 4.9.

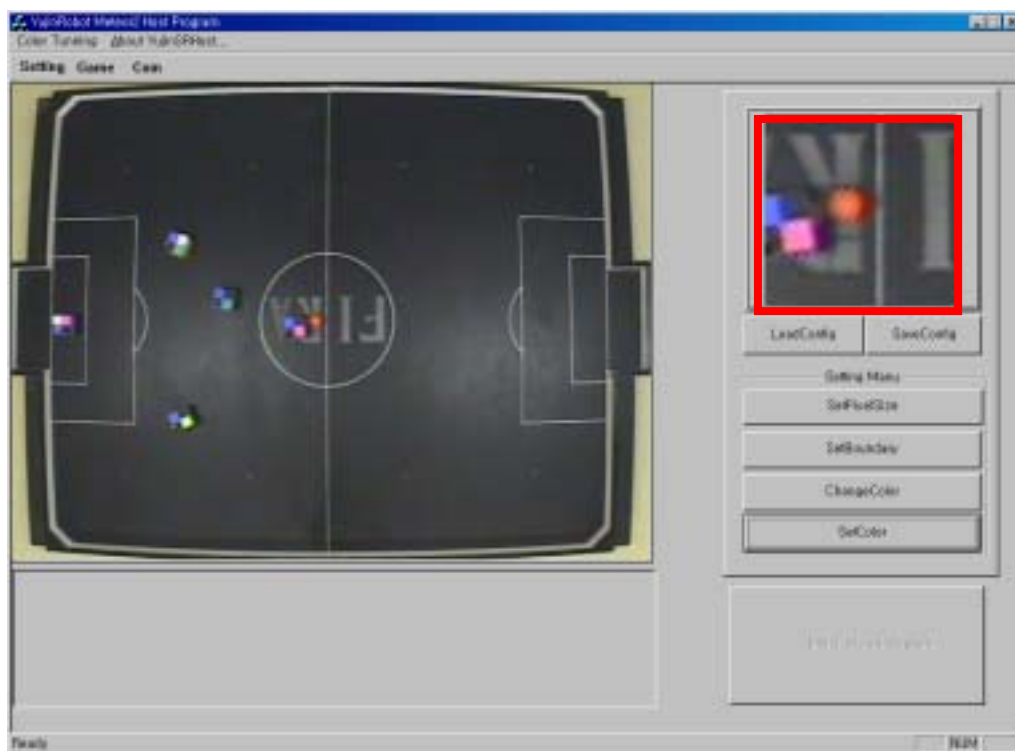


Figure 4.9 3 times zoomed ball after putting 'SetColor' button

In the zoomed screen, drag the mouse to draw a small square that fits into the ball. You can assign the starting point of the square by clicking the mouse in the enlarged screen, drag the mouse to draw the square, and end drawing the square by double clicking the mouse. Then, in the actual playground screen, the area colored with the

same color as the one selected by the mouse in the zoom screen will be colored green and rest of them will be blacked out as illustrated in Figure 4.10.

As shown in Figure 4.10, select 'Ball Color' in the 'ColorSelect' panel and adjust the YUV values to set the color of the ball.

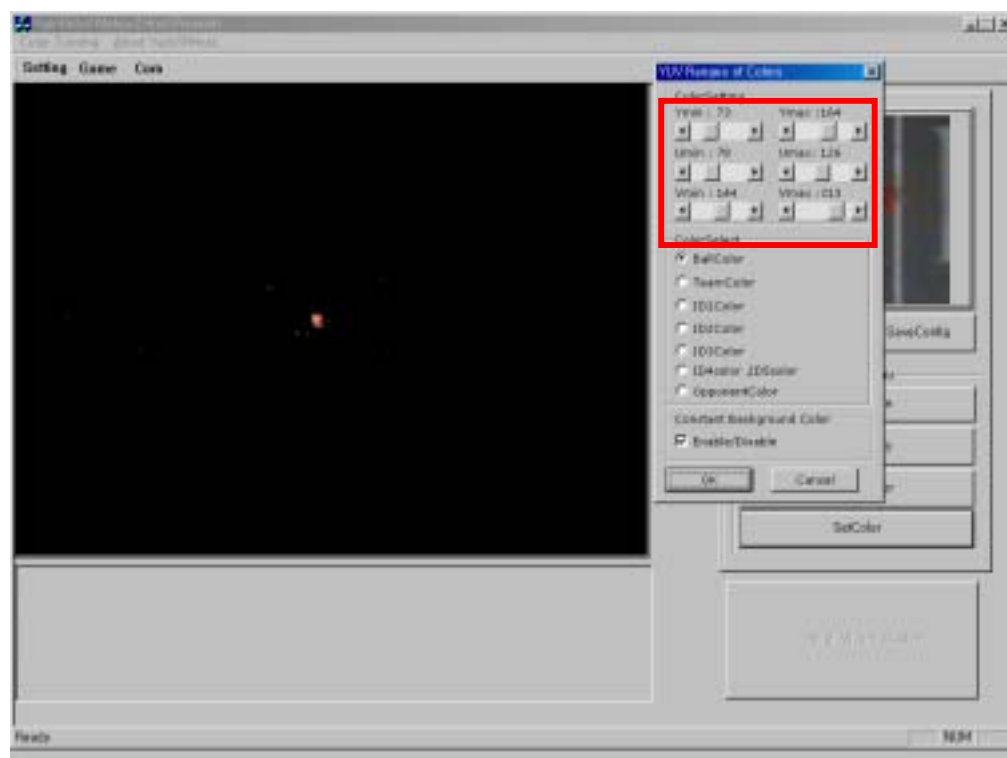


Figure 4.10 Setting YUV values of the ball color

Set the correct color for the ball by changing the minimum and maximum values of RGB. If you mark the check box named 'Constant Background Color' at the end of the 'YUV Ranges of Colors' dialog box, only the selected color will appear in green and

In the figure 4.10, Ymin, Ymax, Umin, Umax, Vmin, Vmax values are the values of the minimum and maximum values of each. To describe the colors, we usually use RGB ranges of color. However it is rather difficult to set the colors in the dark places or bright places. The colors set in the dark places are different in the bright ones.

For example, Red color will be brighter in the light place, but will be dark red in the dark place. In this case, we can say this is just red, but the computer cannot calculate as human eyes can see.

Therefore we have to set each value of the colors. The most important thing for setting colors, you have to set the colors if it is dark, middle, or lightest one for the

computer to calculate the color exactly.

So we use YUV values of setting colors. First, Ymin and Ymax mean values of brightness. Ymin means it is set if the color is more than some values, and Ymax means it is set if the color is less than some values.

You have to set the Ymin as smaller one and Ymax as bigger one because the light of the ground is not equal. Umin, Umax, Vmin, Vmax values are for the ranges of colors. These values should be not so different between min values and max values. So you can set the colors that you want. In this way, you will find the setting color of the ball in the figure 4.10.

So controlling the bar of min values little by little, and max values bigger by bigger, you have to make sure if the orange color for the ball is clear on the black background. So if the ball color has some holes inside, the computer cannot search the position of the ball in the real match.

After setting up the ball color, now set the team color for the robots. The procedure is the same as you did to set the ball color. Click 'Set Color' button first. Select the robot by clicking the mouse in the actual playground screen as shown in Figure 4.11.

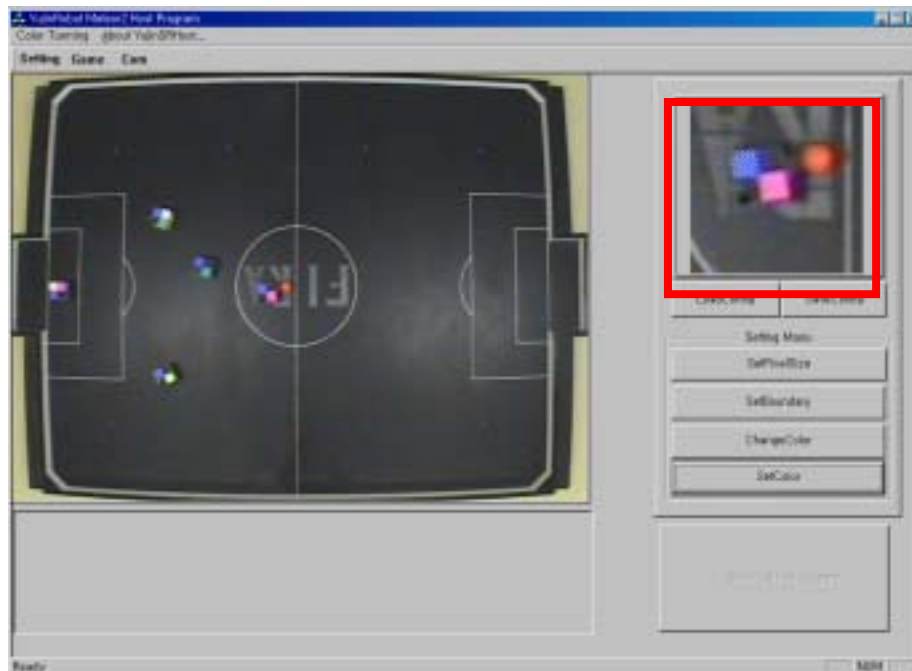


Figure 4.11 Select a robot for setting Team color

Like Figure 4.11, a three times zoomed screen will be appeared, and this is illustrated in Figure 4.12. In the zoomed screen of Figure 4.11, drag the mouse to draw

a square that fits into the area that is colored with the team color (blue). After the selection, a dialog box named 'YUV ranges of Colors' pops up. Now, in the panel, 'Select Color', select 'Team Color' and then adjust the YUV values to set the team color. This process is the same procedure as the one you went through to select the ball color. Decrease the value of Ymin little by little first and then increase the value of Ymax little by little. If the green colored area does not change even though the value of Y is changed, try to change the value of U. After that, change the value of V to obtain the color you want. Since RGB color ranges exist for each particular color, the smallest minimum value or the biggest maximum value does not guarantee to have the best color setting. Therefore, you need to find the best combination of the minimum and maximum values. Figure 4.12 illustrates the team color based on the color range which was set previously. Once you have a satisfying result, click 'OK' button.

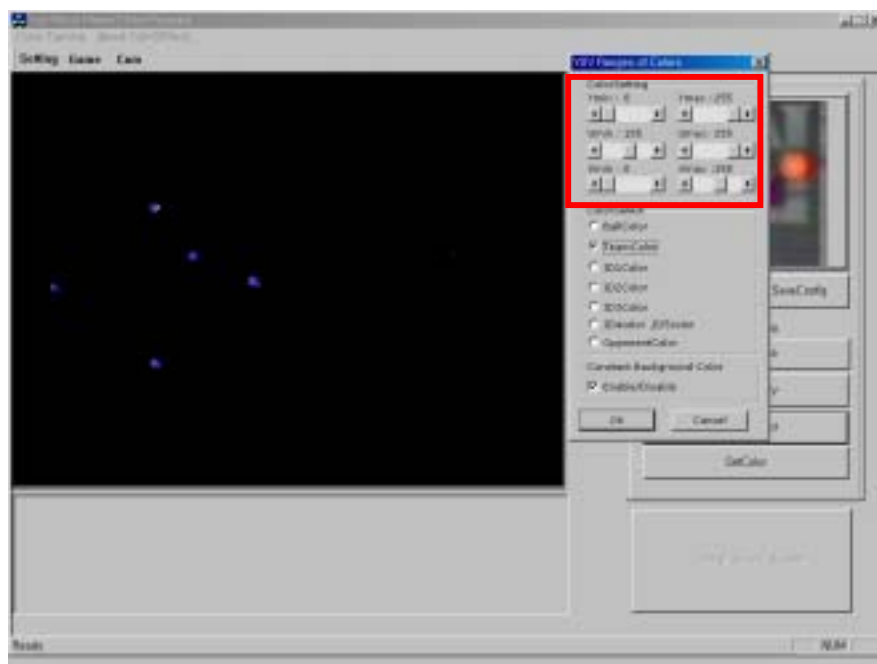
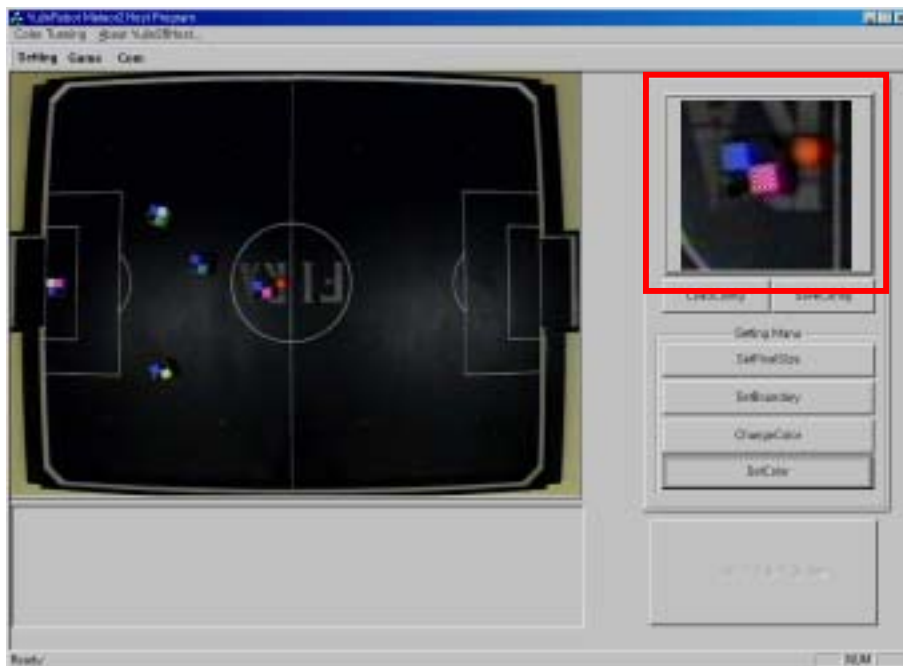


Figure 4.12 Setting team color

After setting up the team color, now you need to set up the ID color of each robot. In the 'Select Color' panel, select 'ID1 color'. Again, this setting process has to follow the same routine that you went through previously to set up the ball color and the team color. Figure 4.13 is the screen shot of the window to set up a robot's ID. Once you obtain a satisfying result, click 'OK' button to continue.



The way of setting robot ID2 Color, ID3 Color in the same above way. After setting ID colors of all robots of your team, you can set the opponent team color (if your team color is blue, the opponent team color will be yellow.). Then setting opponent's team color, select Opponent Color in the SelectColor panel.

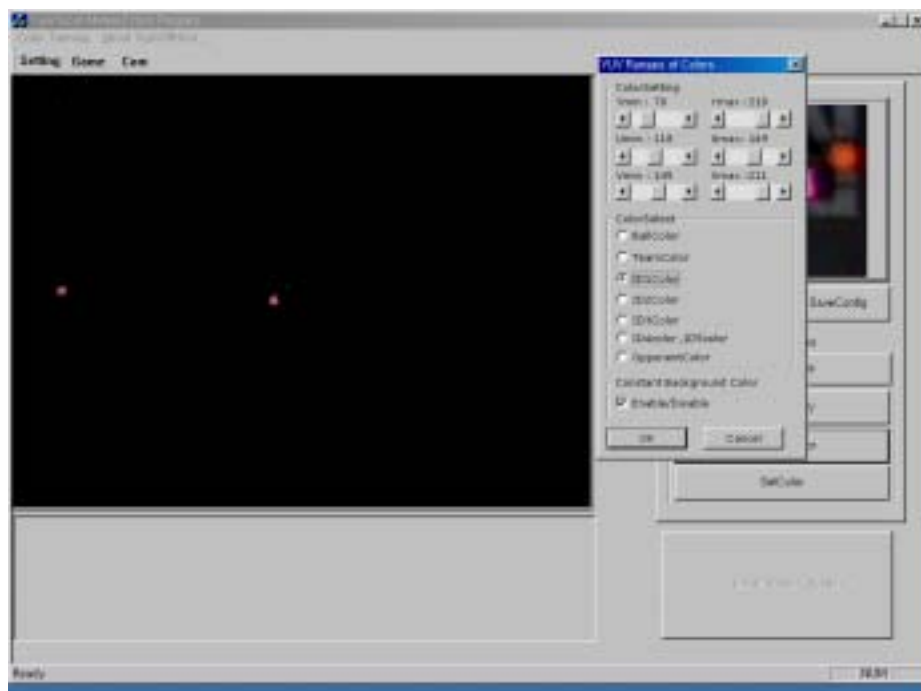


Figure 4.14 Screen for setting YUV values to settle each robot's ID color

In this way, set the other robots such as ROBOT ID2, ROBOT ID3. However you must use same color patches for ROBOT ID4, AND ROBOT ID5 as ROBOT ID1, ROBOT ID2.

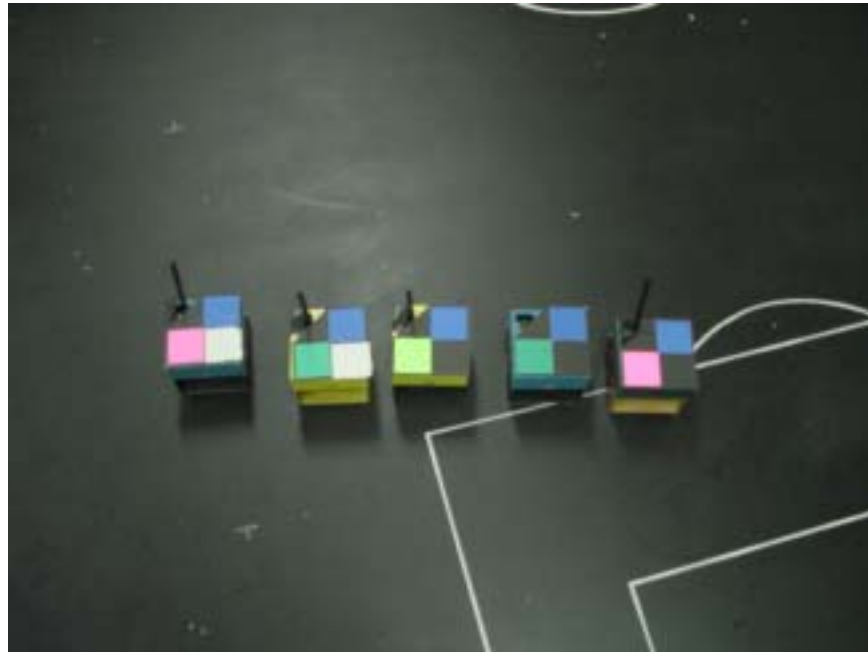
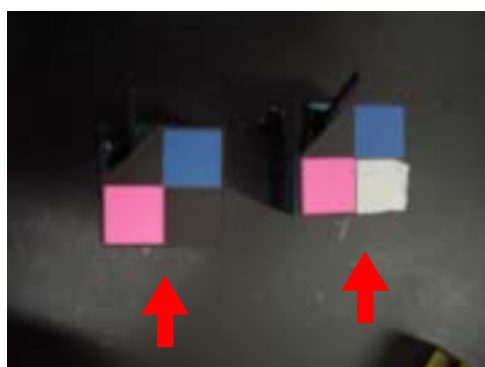


Figure 4.15 Home Team Color

For ROBOT ID4 AND ID5, you should add one more color paper for these 2 robots so that they can be different from ROBOT ID1, and ID2 with same ID colors as figure 4.15.



**Robot1      Robot4**

Figure 4.16 Robot 1, Robot 4 color



**Robot5      Robot2**

Figure 4.17 Robot2, Robot5 color

As figure 4.16 & 4.17, ROBOT ID4 AND ID5 are distinguished from ROBOT ID1, AND ID2 by using white square color paper.

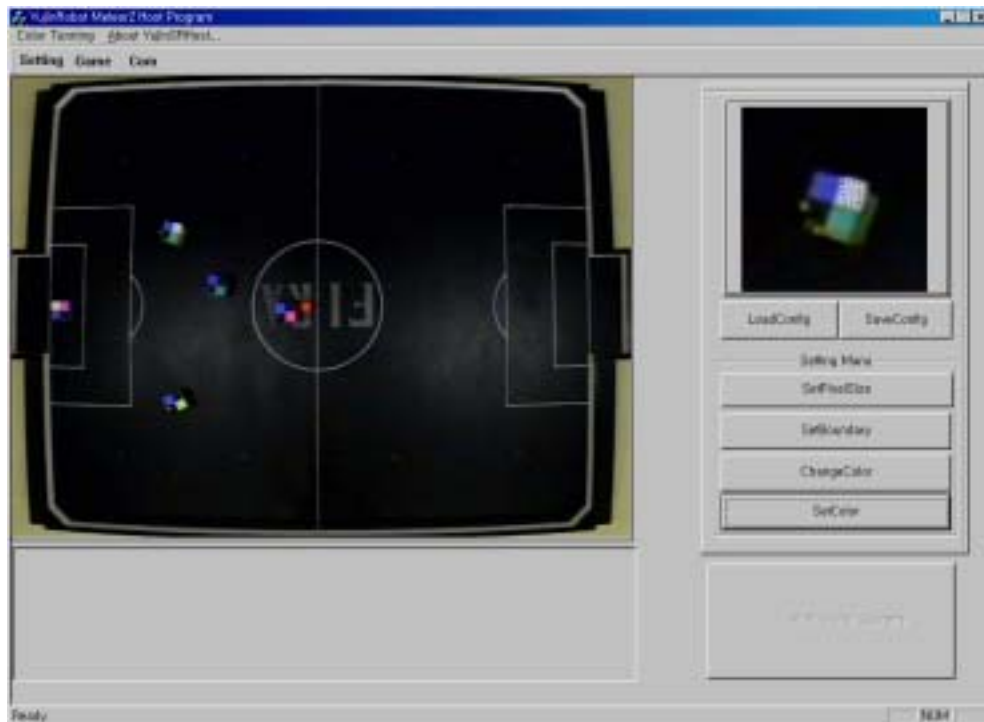


Figure 4.18 Putting ‘SetColor’ and clicking robot ID1 to distinguish from ID4 and ID5

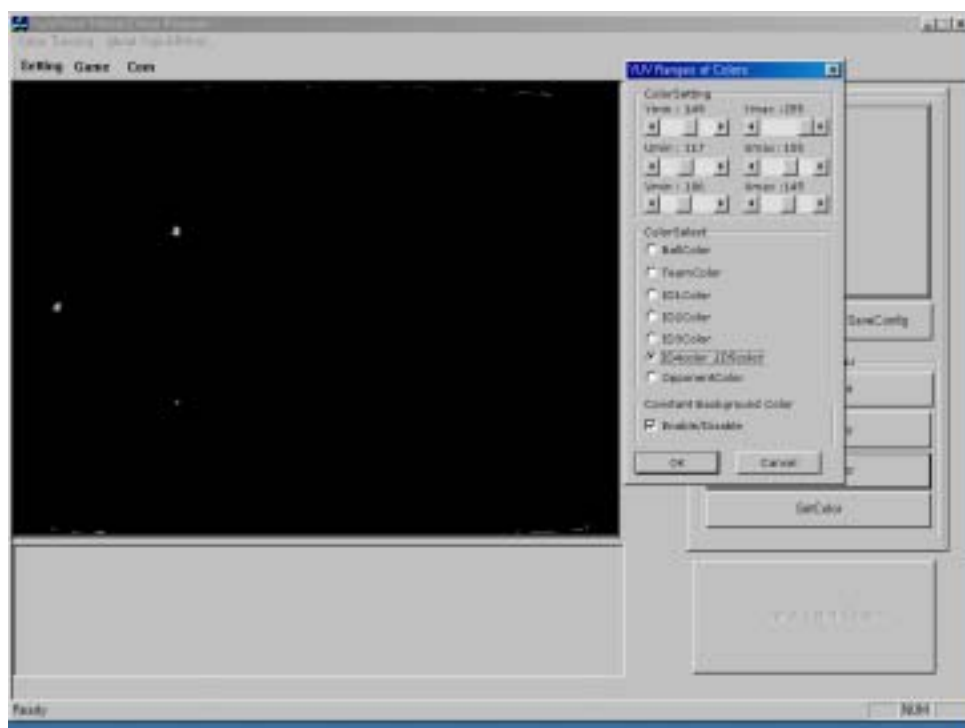


Figure 4.19 Setting YUV values after proceeding Figure 4.18



If it requires for you to find positions of opponent robots in the strategy program, you have to set the opponent's team color. How to set the team color of opponent team is same as setting domestic team color. In 'ColorSelect' menu, you can choose as 'Opponent Color'.

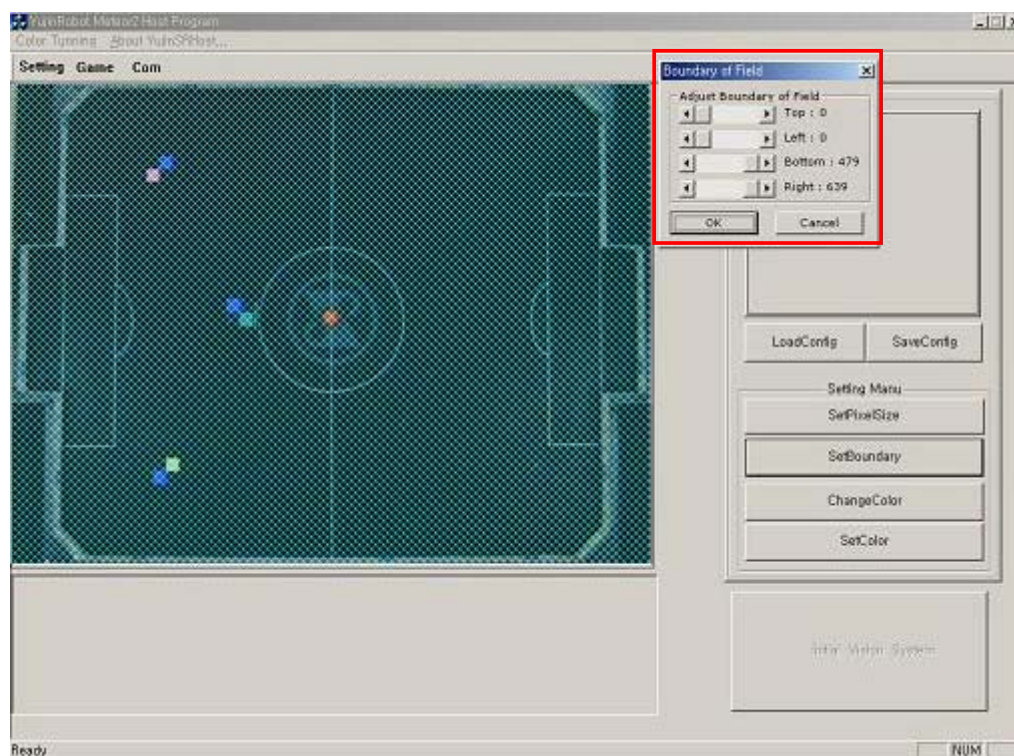


Figure 4.20 Setting the boundary of the playground

After setting up the ball color, team color and robot ID colors, you need to set up the boundary of the playground. Click 'Set Boundary' button to open up the 'Boundary of Field' dialog box. In this dialog box, you can set the boundary of the playground. To set the boundary, fix the color patches on the robots and keep the robots at the edges of the playground. In Figure 4.20, the robots are located at the upper edge of the playground. Even if the robots are actually located within the area of playground, in the playground screen captured by the camera, robots might seem to be located outside the playground. Therefore, you need to set the boundary of the playground that is a bit larger than the actual size.

The green rectangle changes its size as the range of 'Top', 'Left', 'Bottom' or 'Right' are varied. You can set the bottom boundary of playground in a similar way. You should be careful while setting the right and left boundaries of the playground.

You should not include the goal post in the right and left boundaries. Thus, in the program, the range of x is from 0cm to 150cm because the length of the playground is 150cm without the goal posts, and the range of y is from 0cm to 130cm. Also, the right, left, upper and bottom sides of the actual playground can be different from those of the screen shot captured by the camera, depending on the positioning of the camera.

After setting up the boundary, set the pixel size by clicking “Set Pixel Size” button in the right side menu.

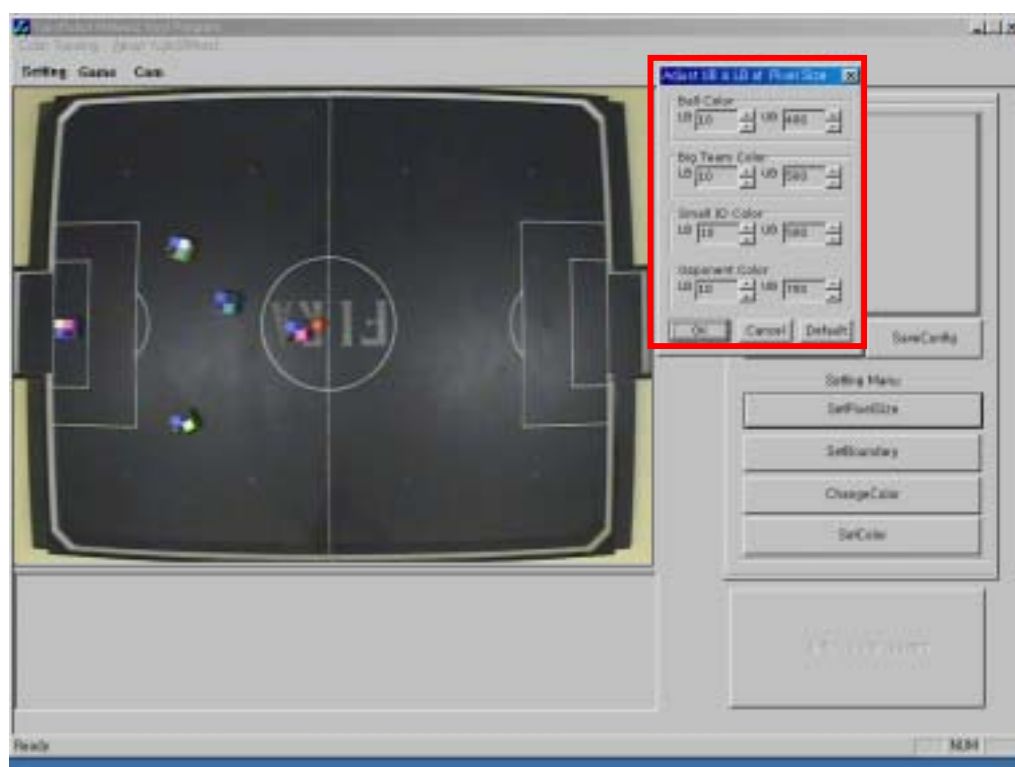


Figure 4.21 Set the pixel size

If you select a robot in the playground screen, the robot will appear in the three times zoomed screen. In the zoomed screen, draw a square that surrounds the robot, and then a dialog box that informs you the size of the robot will be opened. When the whole playground fits perfectly into the screen, the size of the robot will be about  $14 \times 14$  pixels. That is, a robot with a dimension of  $7.5\text{cm} \times 7.5\text{cm}$  will be appeared in the program screen as a square of  $14 \times 14$  pixels. Using this information, you can calculate the maximum and minimum size of the ball color, team color and ID color patches. Thus, 130cm of actual length will be equal to 240 pixels and 170cm (including the goal posts) to 313 pixels. So, the size of the ball would be around 26 pixels. Also, the size of team color patch will be around 77 pixels and the size of the

ID color patch will be approximately 13 pixels. These values may differ depending on the size of the color patch and the zoomed range camera.

Figure 4.21 illustrates the case when the 'Set Pixel Size' button is clicked. A dialog box named 'Adjust UB & LB of Pixel' opens up, and from this dialog box, adjust the maximum and minimum pixel sizes of the ball color, team color and ID color.

**Once the sizes of the robot and pixel are set, you do not have to set them again.**

After these steps, save the configuration by clicking the 'Save Configuration' button on the right side below 3 times zoomed screen as shown in Figure 4.22.

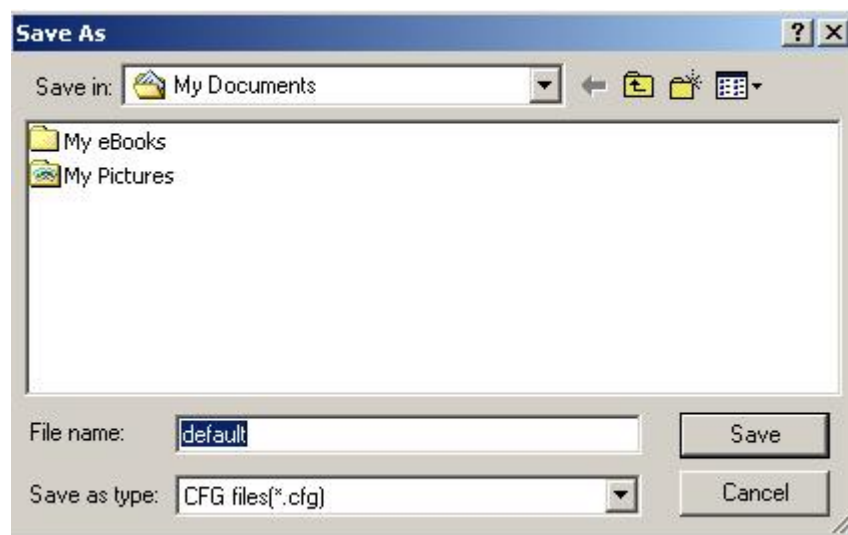


Figure 4.22 Save the configuration file

Saving the config file is very important to keep this environment after you finished, and execute it again later. So please make sure that you have to save as a config file after you set colors for above.

If you already finish all settings and savings, you will find three menus on the upper left side of the vision program. (On the figure 4.17, there're three menus for Setting, Game, and Com.). Click **Game** button, and the screen will be changed as figure 4.23.

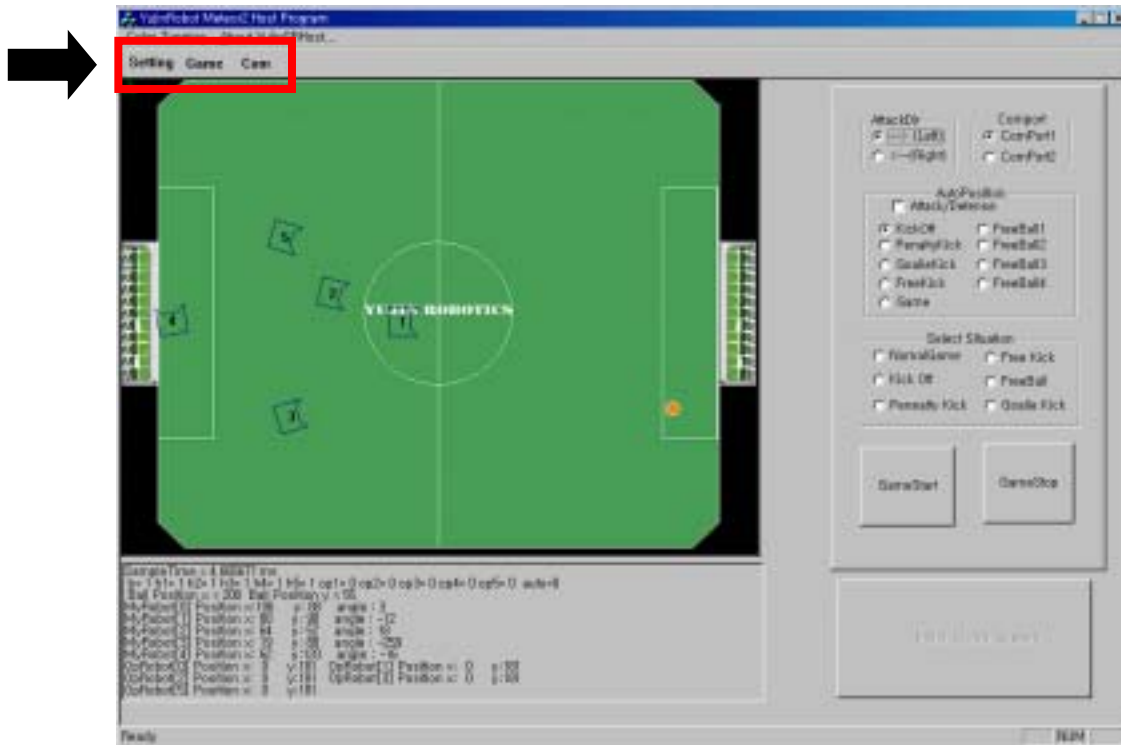


Figure 4.23 Menus for Setting, Game, & Com

Through these 3 menus, you can set and play soccer robot games generally. First menu (Setting) is for setting colors of robots ID's, Team color, Opponent color, and the ball for the computer to trace their position. Second menu (Game) is to select buttons necessary in the match. Third menu (Com) is to test the robots if they are working properly.

First, let me explain the menu about Game. Game menu is to let the robots to move to a selected position or to work a set-play. Or originally, you can set the Game start and stop.

First of all, you need to select a serial port for the RF/IR communication as figure 4.24. If you are using COM1 for communication, mark a check named 'COM1' or 'COM2' if you are using COM2. As the computers with Pentium II or better CPU use a PS/2 keyboard and mouse, use 'COM1' port for communication in such models. For computers with the Pentium processor, use COM2 because most of the computers use the COM1 port for the mouse.

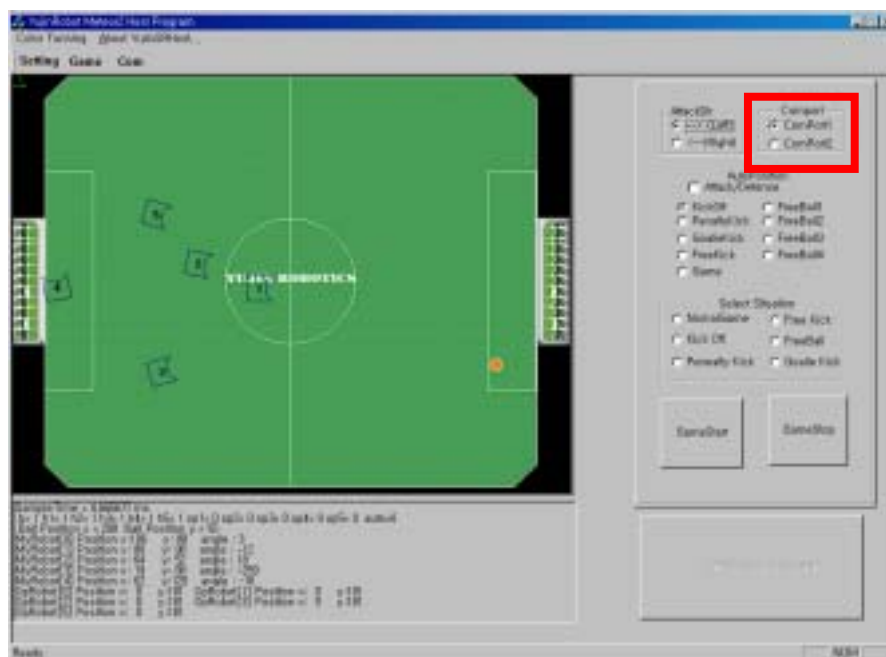


Figure 4.24 Comport setting

As Figure 4.24, after setting the serial comport, you have to choose the direction of attack for domestic team. The setting is same as Figure 4.25.

On the screen, the left side of the screen corresponds to the left goal post and the right side of the screen is the right goal post. Since the home field will be switched between the first and second half of a game, you need to specify which side refers to your home field as figure 4.25.

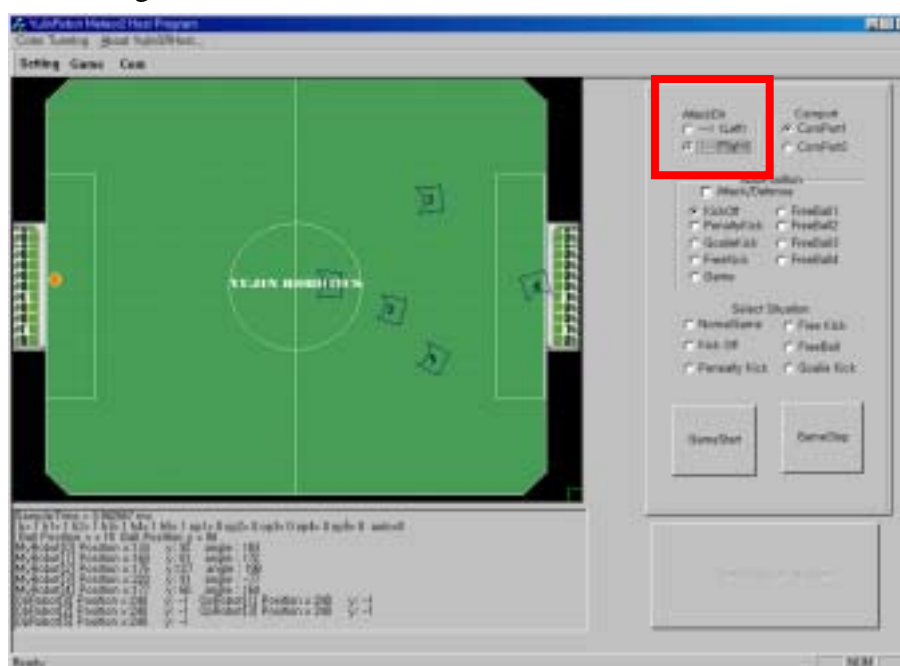


Figure 4.25 Position of attack from left to right

Figure 4.26 a red box means a direction for the team to attack from right to left. So left side of the goal area on the screen will be the left goal side in the real match and right side of the goal area on the screen will be the right goal area. The first half and the latter half will be changed to the opposite side. So you have to change it after a half.

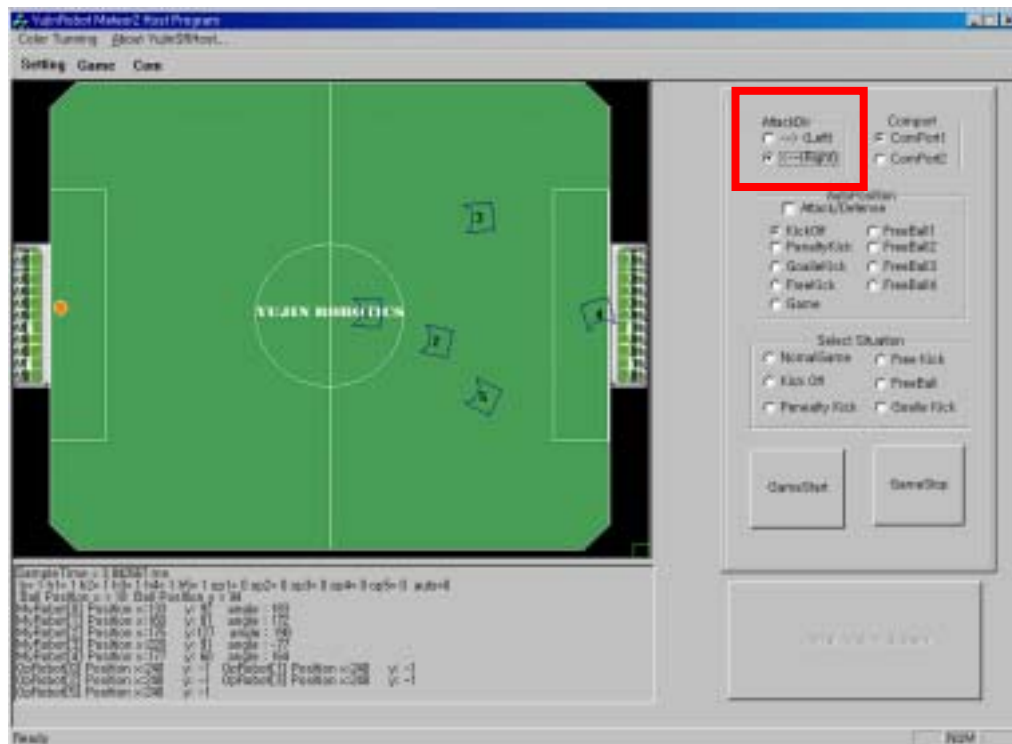


Figure 4.26 Position of attack from right to left

As you can see on the figure 4.27, there are many executing buttons for the game. There're functions to set if the team is for attack or defense, Auto Position, and Select Situation. The menus are same as figure 4.27.



Figure 4. 27 Executing buttons for Game menu



As you can see on the figure 4.27, Auto position plays a role for the robots to move and wait to the positions which were set in advance in case of penalties such as Free Ball, Goalie Kick, Penalty Kick, etc... To use Auto Position menu, first you have to set the color settings for each robot's ID color, team color, opponent color, and the ball's. After that, save the color setting on the Setting menu. Then turn to Game menu, and select one in the Auto Position as you want to move the robots in case of penalties.

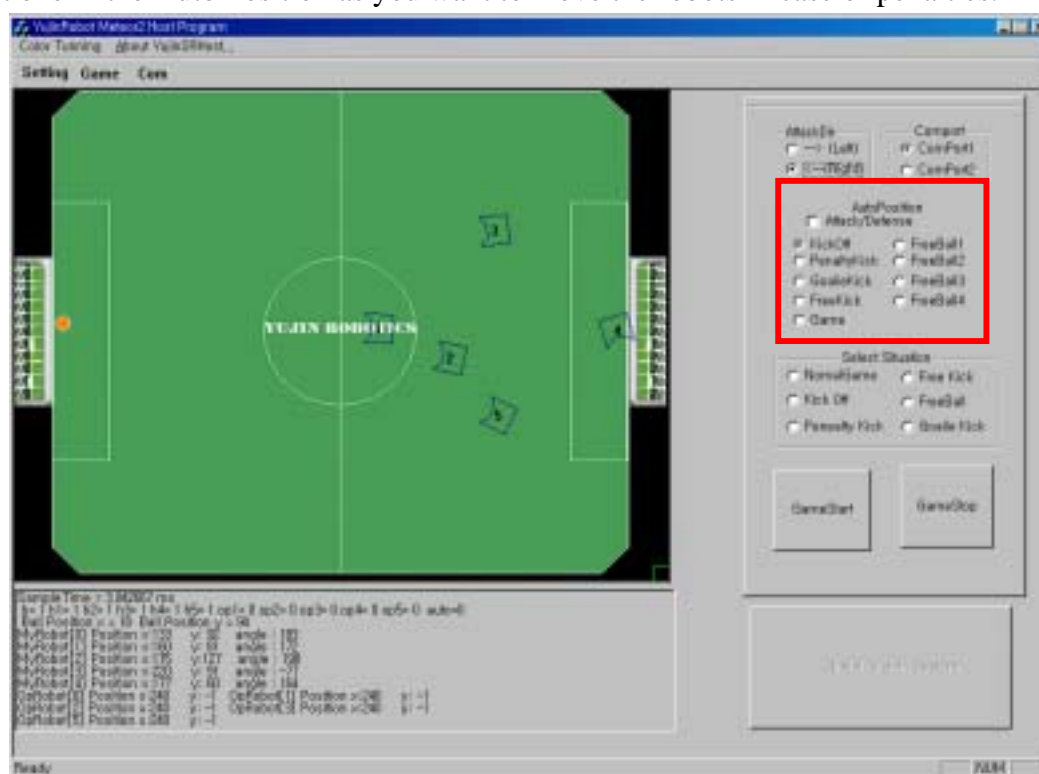


Figure 4.28 Auto Position menu

If you click Attack/Defense box on the Auto Position menu, your team will be a defense team and the opponent will be an attack team. In the opposite, if you do not click, your team will be an attack team.



Figure 4.29 Case of defense & moving robots to KickOff position

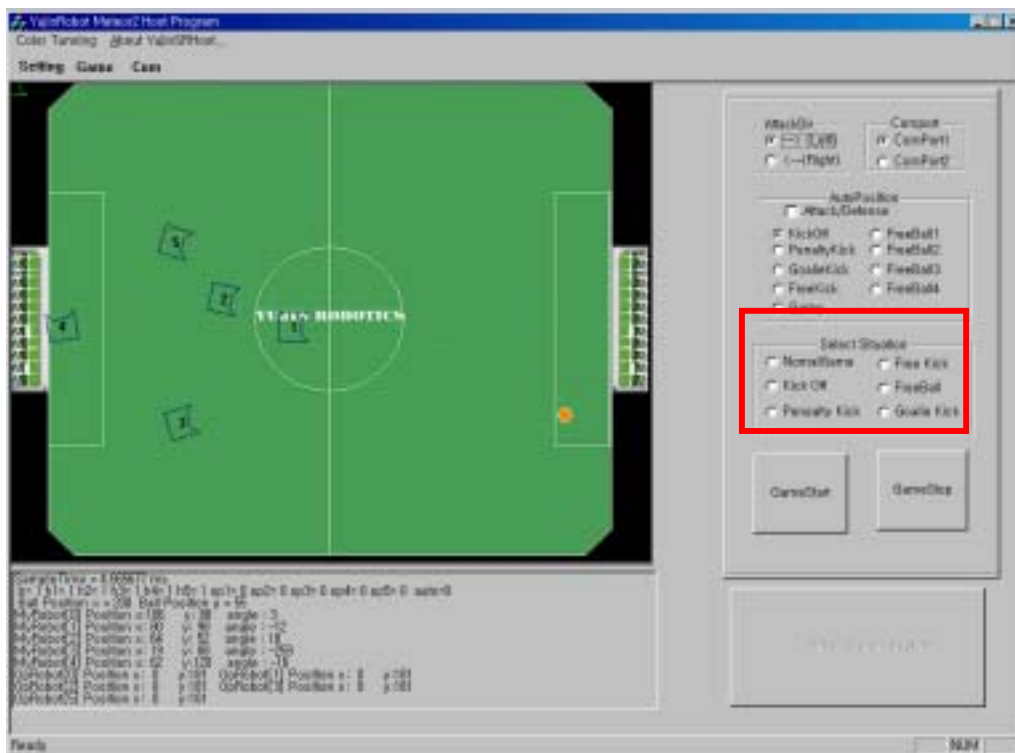


Figure 4.30 Select Situation Menu

Figure 4.30 describes the Select Situation menu of the vision system window in the middle of the right menus. This menu is a function to make the robots to move to the specific situations, not to move basically made by basic strategy. In case of Goalie Kick, Robot no. 1 grasp the ball and tries to shoot fast to the opponent's goal line. In this case, the robot should not shoot the ball in any position. Instead it should in the better position. Therefore this menu plays a role for the robot to shoot as a higher velocity than movement function values usually.

To use this function, save the color settings you made out. After that, click the position among Select Situation functions and put the GameStart button.

In the 'Select Situation' panel, there are various actions that the robot can take in many situations such as Kick Off, Penalty Kick, Free Kick, Free Ball, Goalie Kick and Defense. Defense is selected by default. Select an appropriate situation from this panel that matches with the various happenings during the game. For instance, if you get a penalty kick chance, select 'Penalty Kick' so that your robot can perform the penalty kick towards the opponent's goal. On the other hand, if your opponent gets a penalty kick, select 'Defense' so that your robots can assume the defense mode. The same method can be applied to the other commands as well.



If you have been successful to follow every single step up to now, setting up for the vision program is completed.

You need to pay the very careful attention on setting up the boundary of the playground and the colors of the robots, because a poor setting can make the vision program to lose the tracking of the motion of the ball and robots.

Finally, if you did settings for Robot's ID, and channels with transmitter through dip switches inside the robots, now you have to test the robots if they are working properly. Figure 4.31 describes how to test the robots easily even in the real match in vision program. To test the robots, click Com menu on the upper side of the window, and you will see the real screen of the ground and move the direction buttons on the keyboard by controlling their velocity.

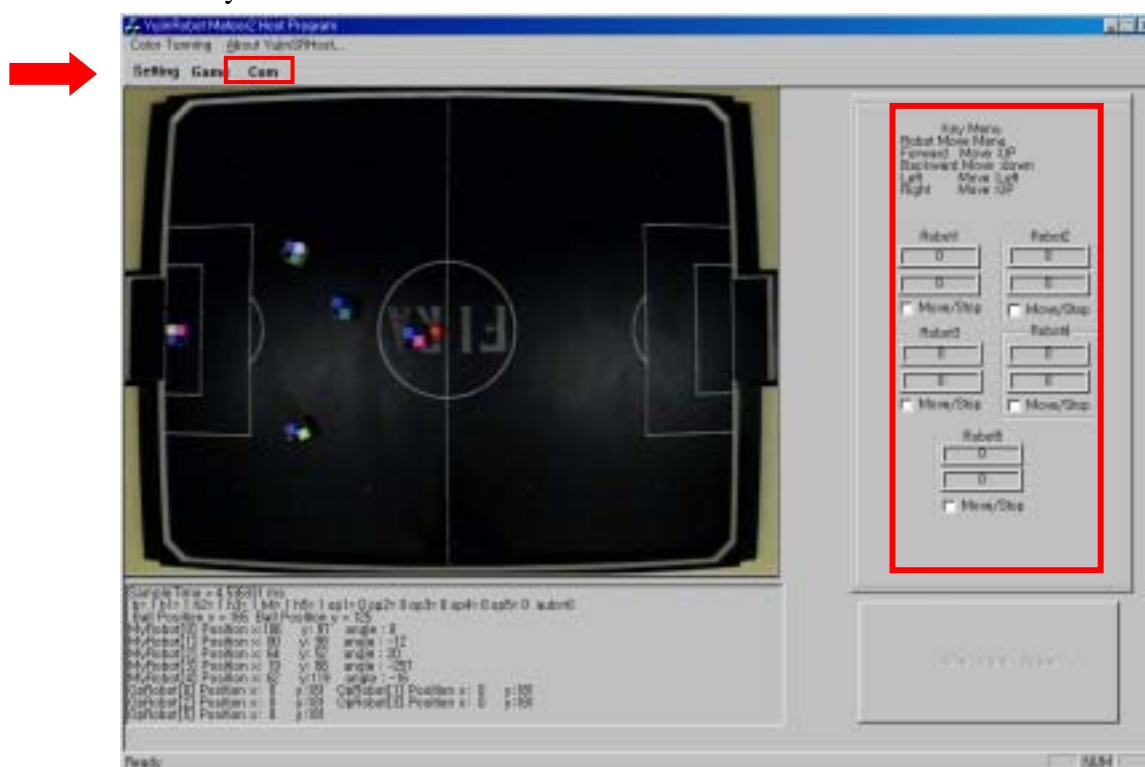


Figure 4.31 Keys for Robot Test on Com menu

Figure 4.31 shows an example to test the robot if they work completely. On the condition of the robots POWER ON, you see the description how to use the scroll bars ,  $\uparrow$  key means to move the robot to the upper side,  $\downarrow$  key means to move the robot to the below,  $\rightarrow$  key means to move the robot to move the right side, and  $\leftarrow$  key means to move it to the left.

In this case, you have to make sure that there should be V check on Move/Stop button to make the robots move. If you don't, the robots will not work properly.

## 5. Appreciation of Robot Program

First of all, let's look at figure 5.1 to understand the system of Robot program. This figure shows a general flow to find the position of each robot through vision system and let the robots move to the desired position after searching the position of robots.

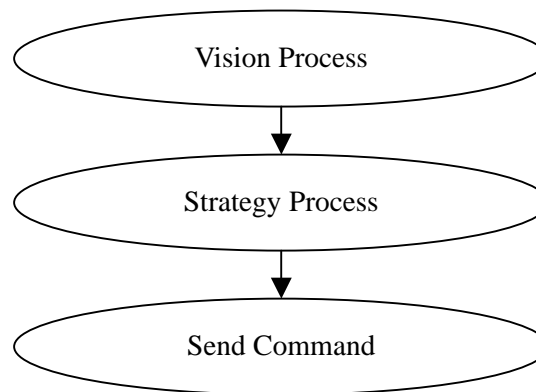


Figure 5.1 Flow chart of Robot Soccer System

As figure 5.1, it is called as Feed Back system that finally moved robots are received a command again after strategy calculation from the host computer through vision program inside.

Through Feed Back system, you can make a best system if the exact controller is well made up with fast answer of sensor, and correct output of controller. So it depends on these items.

The Soccer Robot is to match which team makes a best system of soccer robot game. Figure 5.2 shows a block diagram between soccer robot system and feed back.

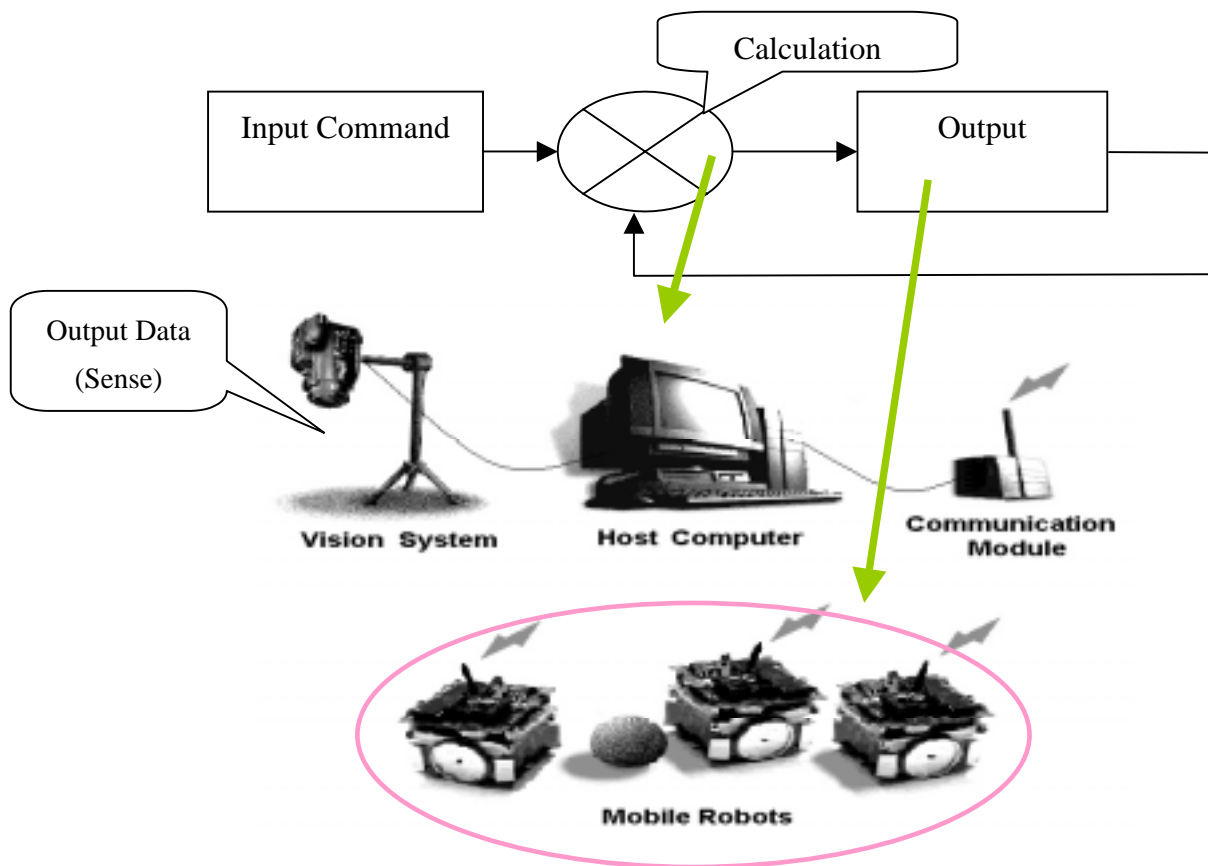


Figure 5.2 Feedback System

## 5.1 Vision System

Our Soccer Robot system uses NTSC way. NTSC is set up as renewal of 30 times per a second (Image Update). It means a screen renewal once per 30ms(milli-second) (Image Update). The system our company using processes vision system per 16 ms calculating a unit of field supporting in Meteor 2/4 card.

Vision system means to find the position of robots through the color patch using the color information. Vision system is materialized as VisionSys Class, searching some approximate location of a robot on the total ground and in the local area, it searches again its exact position. Figure 5.3 describes functions of vision process.

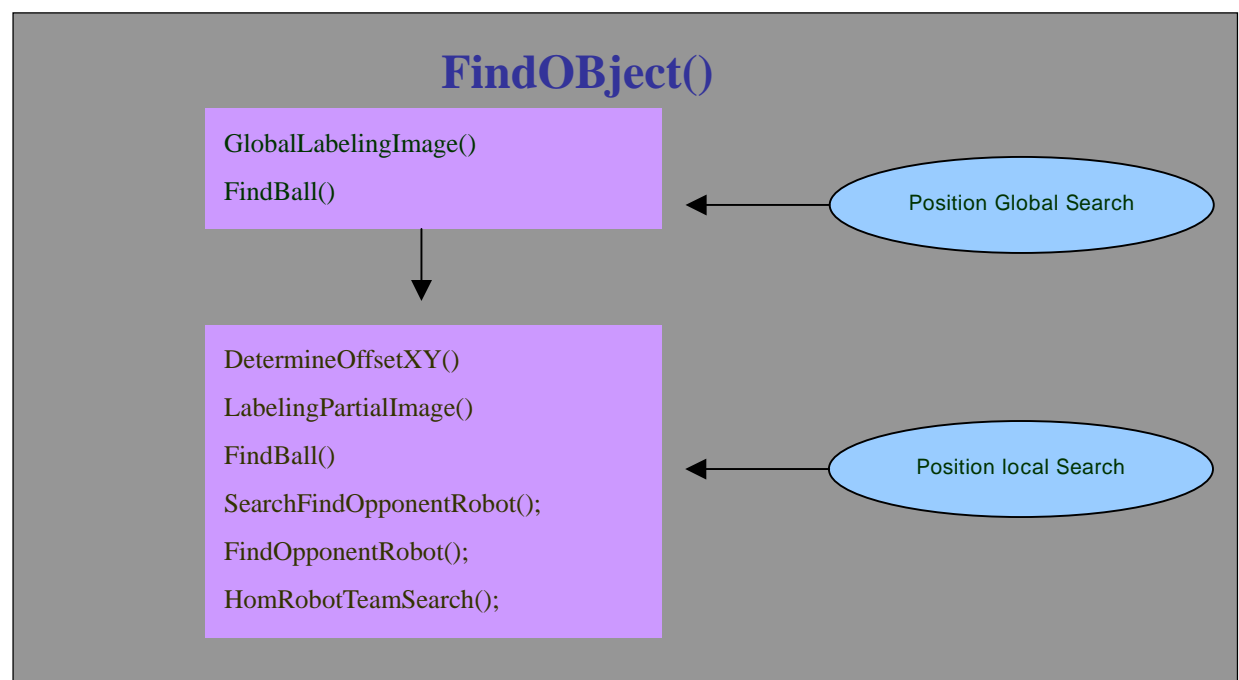


Figure 5.3 Functions Processing in Vision

It is important to find each position exactly, but also necessary to find as fast as it does. Our system is set as process within time to send a command to robot after vision processing, and strategy calculation in 16 ms. But, you have to make sure that the vision should be calculated in 16 ms about the strategy.

After vision processing, the position of the robots and the ball can be received from a function named `Get...Position()` in Vision sys Class. The position of the ball can be used of a function named `GetBallPosition()`, our team robots of `GetHomeRobotPosition()`, and opponent team of `GetOpponentRobotPosition()`.

```

for(int i=0;i<7;i++)
    findFlag[i]=0;

m_pVisionSys.FindObjets(EvenOrOdd);           //vision process
m_pVisionSys.GetBallPosition(PositionOfBall); //position of ball
m_pVisionSys.GetHomeRobotPosition(RobotPosition); //position of our team robots
m_pVisionSys.GetOpponentRobotPosition(oppRobotPosition); //position of opponent
m_pVisionSys.GetFindFlag(findFlag);           //vision processing or not
ConvertVisionPoint(m_CGame.MyRobot,m_CGame.OppRobot,&m_CGame.Ball);

```

The position received is a logical coordinates (axis values for pixels on the screen) , so to receive the information of positions in the real ground, you can call the function named ConvertVisionPoint() in View class.

In the actual strategy functions, position information used is saved on the next variable in Game Class.

At this time, the point of sign X, Y can be described as 170 \* 130.

```

Ball           // whether find the point of the ball's position and the ball or not
MyRobot        // whether find the position of robots and angular point or not
OppRobot // whether find the position point of the robots or not

```

## 5.2 Strategy/Tactic system

To move the robots actually to any position, you have to make a strategy in which robots can move to the desired position using some points found in the vision program. This is called as Strategy function, this plays a role for the robots to shoot or pass the ball and evade the opponent robots in the game.

Our program is materialized as Game Class in the program called “YujinSRHost”

To embody this program, copy this program from CD provided, and select all files as Figure 5.4.

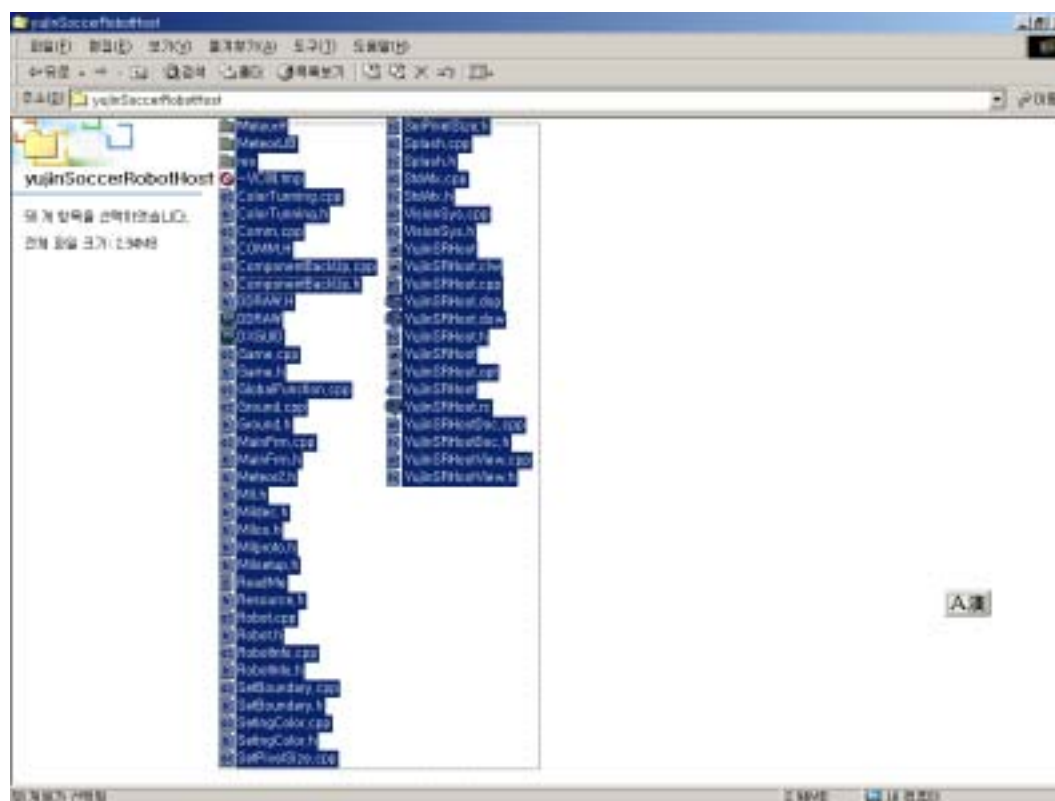


Figure 5.4 Copy “YujinSRHost” in the CD and select all files

As figure 5.5, click the right button of your mouse, and select information and delete only for read property. (Figure 5.6).

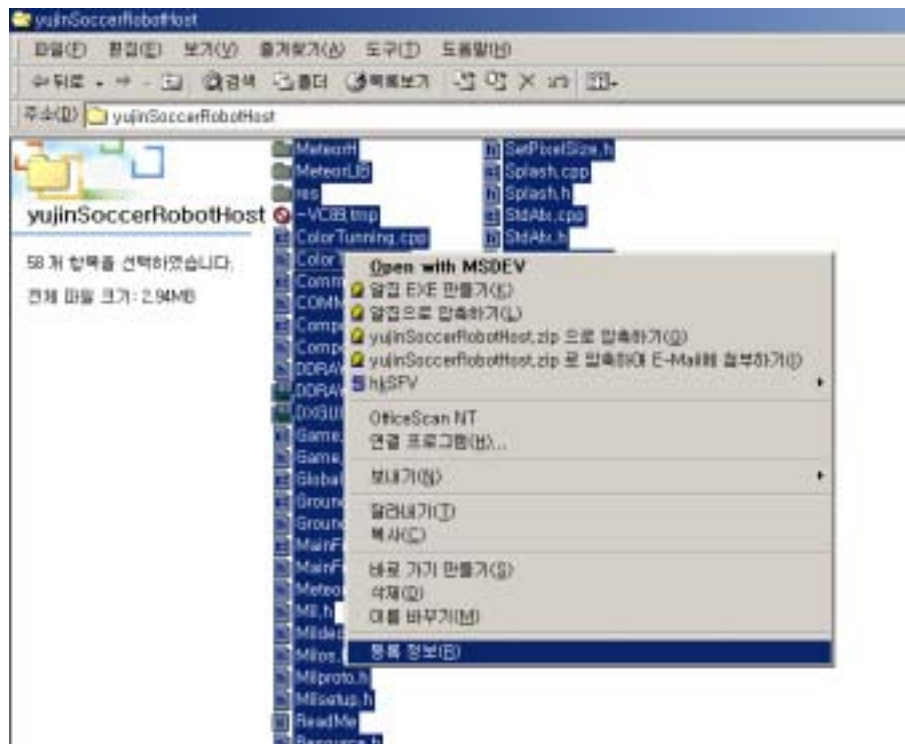


Figure 5.5 Select Registration information by right button of mouse

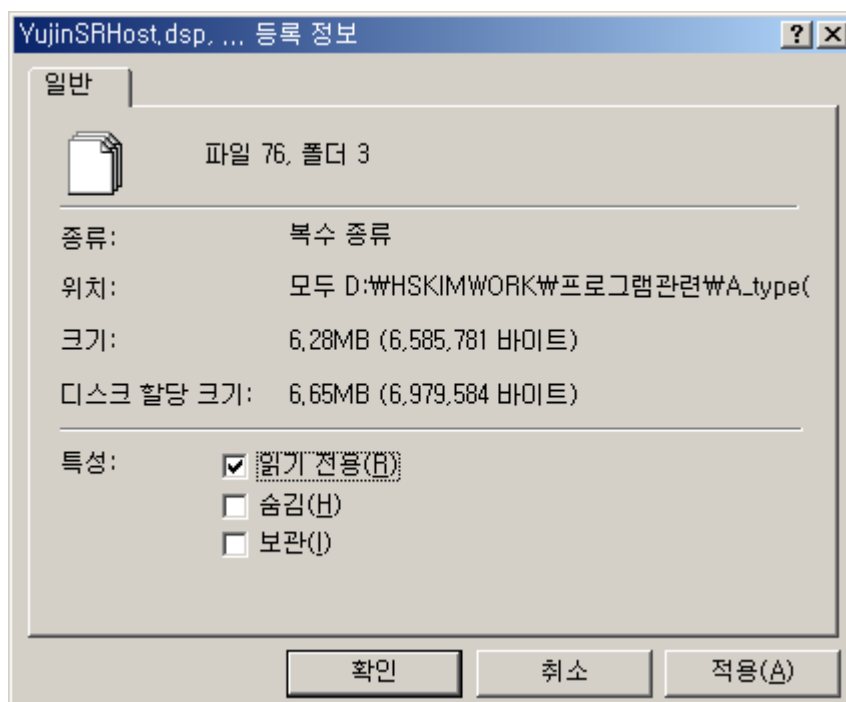


Figure 5.6 Delete Read Only property in Registration information

Double-click a file named “**YujinRobotSRHost.dsw**” and you will see Visual C++ is executed.

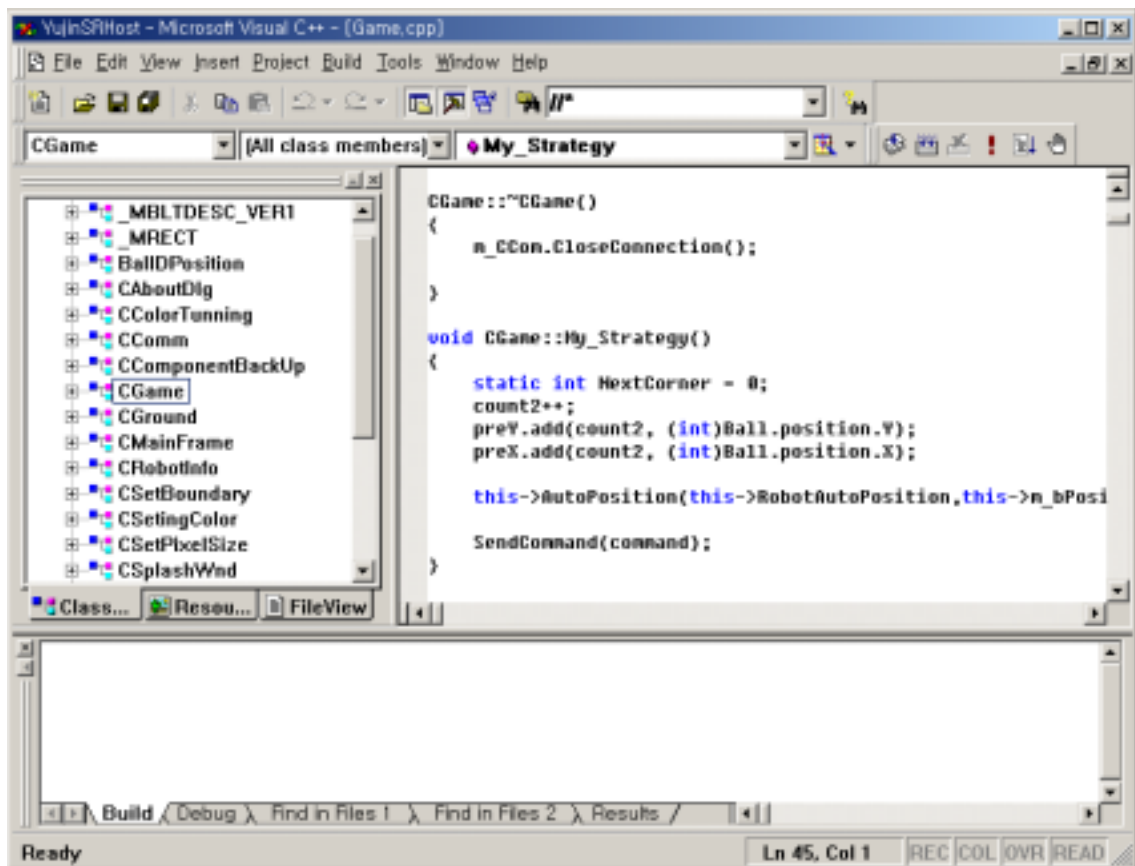


Figure 5.7 Game Class

First of all, to play a game, if you click game button, then Game process() will be called in View Class and after Vision calculation, a function My Strategy() will be called. In this function, there're functions about strategy for the robots' movement and cooperating movement.

Therefore, when you make a strategy function, you can make to call in My Strategy function. In the My\_Strategy() function, there is called of Auto Position() function. Auto Position can be described as following.



```

void CGame::AutoPosition(int AutoPositionGameMode,BOOL AttackOrDefense)
{
    switch(AutoPositionGameMode){
        case KICKOFF:          .....
            break;
        case PENALTYKICK:      .....
            break;
        case GOALIEKICK:       .....
            break;
        case FREEKICK:         .....
            break;
        case FREEBALL1:        .....
            break;
        case FREEBALL2:        .....
            break;
        case FREEBALL3:        .....
            break;
        case FREEBALL4:        .....
            break;
        case GAME:             .....
            break;
    }
}

```

Except for Game, the rest is set for the robots to move in the selected position. This is because it is automatically set for the robots to move in the selected positions in case of penalties or starting point in the actually soccer robot game. When you execute the program, optional buttons of AutoPosition shown on Game menu can be each case.

In the optional menu of “Auto Position”, if Game is selected, the function of Gamealgorithm() in Game Class. Cases inside this function are set to operate some movements in each situation in the actual game. This strategy can be same as a set-play easily men plays in the soccer game. After this situation is finished, Case is set automatically as Defense mode and at this time, general strategy is selected.

```

void CGame::Gamealgorithm(int GameMode)
{
    double dx1, dy1, dx2, dy2;
    // First Kicker is HOME1
    switch(GameMode){
    case KICK_OFF:
        dx1 = Ball.position.X-MyRobot[HOME1].position.X;
        dy1 = Ball.position.Y-MyRobot[HOME1].position.Y;
        if(dx1*dx1+dy1*dy1 <6.0*6.0){
            MyRobot[HOME1].VelocityLeft = 34;
            MyRobot[HOME1].VelocityLeft = 25;
            Velocity(HOME1);
            DeffensePosition(HOME2, Ball.position.X, Ball.position.Y);
            Goalie(HGOALIE);
        }
        else{
            GameMode = DEFENSE;
            InitRole();
            Goalie(HGOALIE);
        }
        break;          .....
    case DEFENSE:
        GameMode = DEFENSE;
        Attack(HOME1, HOME2);
        Goalie(HGOALIE);
        break;
    }
}
}

```

Finally, we finished to look into make up of strategy of “YujinSRHost” ( Game Class).

Next, let’s see the function which No.1 robot of our team moves to the center of the ground after game starts. First of all, to make this function, you have to add it in Game Class. The way to add it, it is same as Figure 5.8. In the Class View, select Game Class and click right button of your mouse, and select Add Member Function.

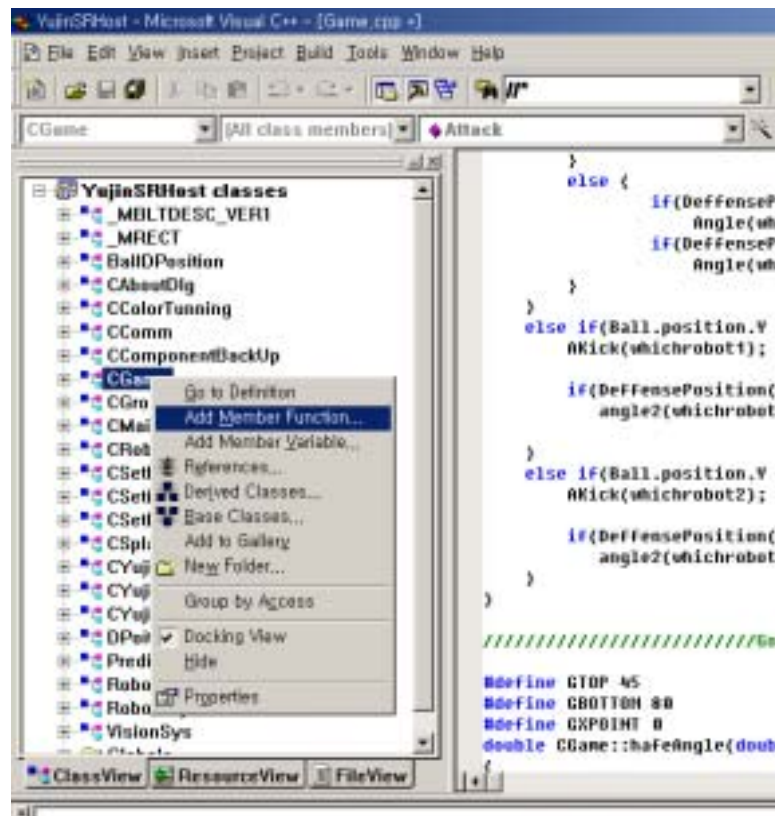


Figure 5.8 Member Function

Make a function of HomeRobot1MoveToCenter() as a type of Void.

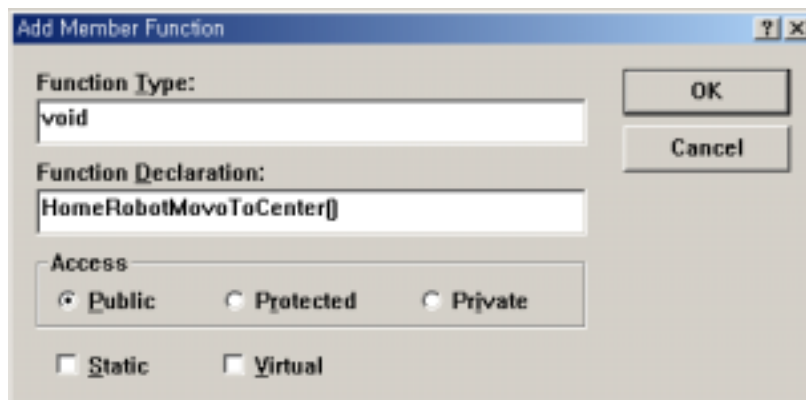


Figure 5.9 Making HomeRoboMovetoCenter() function in Void type

If you click “Ok” button, the function HomeRobotMoveToCenter in Game class. Inside of this function, calling DefensePosition function, and making a point to the center for No. 1 robot. The center point of the ground is X = 85, Y=65, so you can select this point.

```

void CGame::HomeRobotMoveToCenter()
{
    int x_position,y_position;
    x_position = 85;
    y_position = 65;
    DeffensePosition(HOME1 , x_position, y_position);
}

```

When you click Game Start button, My\_Strategy() function should be called inside to execute Game Start.

The execution of above function will be processed if you do as following.

```

void CGame::My_Strategy()
{
    static int NextCorner = 0;
    count2++;
    preY.add(count2, (int)Ball.position.Y);
    preX.add(count2, (int)Ball.position.X);
    HomeRobotMovoToCenter();          //< ===== Call function

    SendCommand(command); //Send data to transmitter
}

```

Put the key“F7” to compile, then execute by putting the key “Ctrl + F5”.

After Color Setting, if the position of the robots can be found in vision program, the robots can be moved to the center of the ground as soon as you put Game Start button.

The way to make another function is exactly same procedure as above in order. So you can make any function you want as it does.

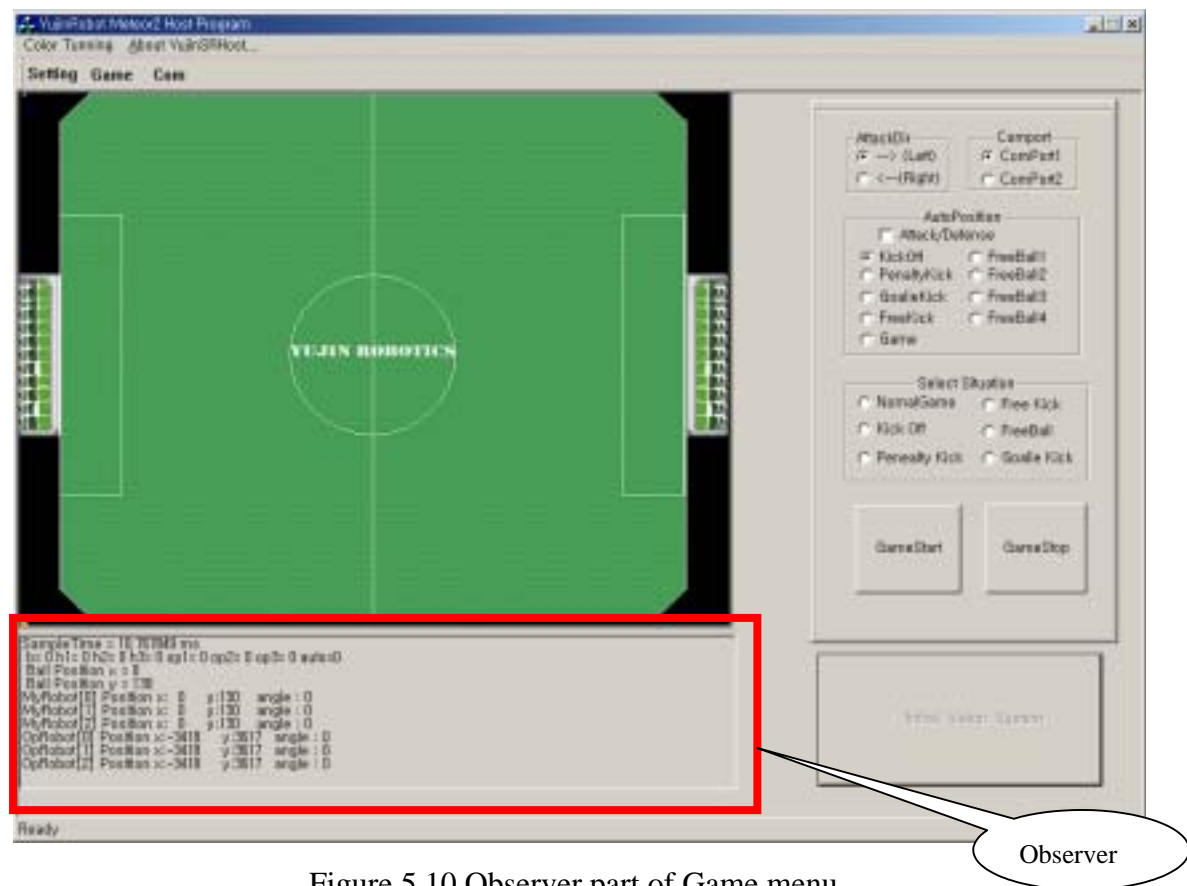


Figure 5.10 Observer part of Game menu

If the ball and the robots can be seen in Vision program, the observer part shows “1” and if not, it shows “0”. Sometimes vision program can find them in some area, but not in some area. It’s because the color setting for the robots is not set completely well. The color you already set can be changed according to the condition of illumination, so you can check time to time.

If you look into the displaying part, there’re positions of each robots, angles, and their values. These shows they are inside of Cstring.Format() content. Here the time means a Sample time once per a time processing of vision system.

If you check if the robots move exactly after you code a new function, put some values in the variable can check the execution of new function.

Call a variable of Game Class as Public, and put each different values in this variable according to its executing part, then you can check if they move in some part or not. In case if initial values in the variable not input, its value is set as “0”. Let’s make an example with Gamealgorithm() function. Assumed there’re Gamealgorithm() function explained above, let’s try to check if the robots move correctly in all cases.

Call a variable used in display as Debug, and declare it in Game Calss.

```

class CGame
{
public: //
    int  Debug;      .....
}

```

Also, with this variable, let them have each different values putting inside of each case of Gamealgorithm() function, so that this variable can be output in Display part.

```

void CGame::Gamealgorithm(int GameMode)
{
    double dx1, dy1, dx2, dy2;
    switch(GameMode){
    case KICK_OFF:      .....
        Debug = 0;
        break;

    case PENALTY_KICK:      .....
        Debug = 1;
        break;

    .....

    case DEFENSE:      .....
        Debug = 5;
        break;

    }
}

```

```

void CYujinSRHostView::GameProcess(bool EvenOrOdd)
{
    .....

    strInfomation.Format("SampleTime = %f ms \r\n b= %d h1= %d h2= %d
h3= %d op1= %d op2= %d op3= %d auto=%d \r\n",float((m_nTime-
_nTime1)*1000), findFlag[6],findFlag[0],findFlag[1],findFlag[2], findFlag[3],
findFlag[4], findFlag[5], m_nRobotPosition, m_CGame.Debug);

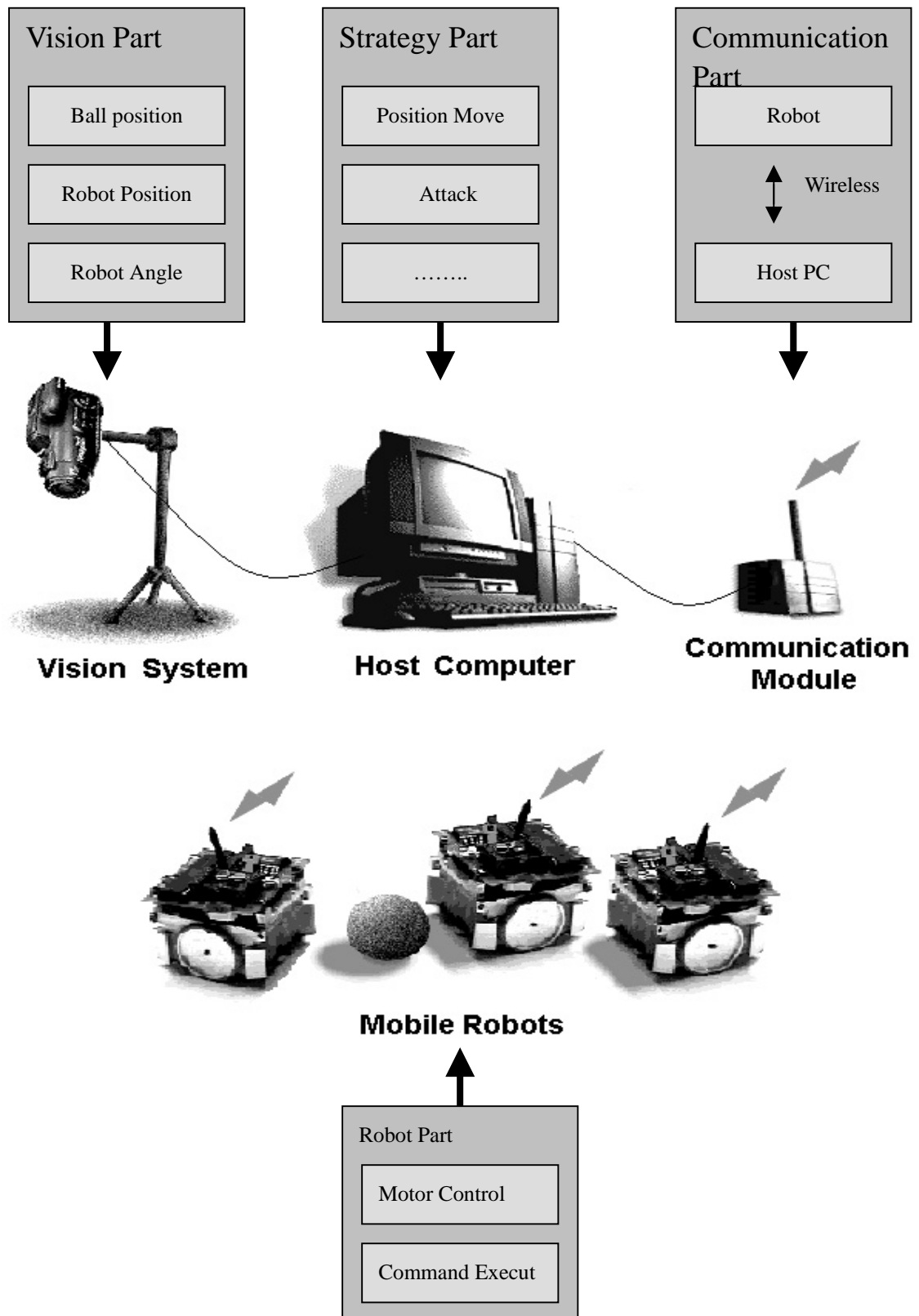
    .....
}

```

In this way, when you run the program and look into Debug values in Display part, you can see which part it is not executed in Gamealgorithm() function. If Debug values are output as expected, but the robots do not move correctly, output different information in Display part and think why. At this time, the information you can use are position of the ball, robot, and their angles. And if you make the robots move after some calculation during this way, you can also output the calculated values in Display part.

As above explanation, it is convenient to declare variables to be used for Robot control and Debugging separately in Game Class. If you declare a variable to debug in Gamealgorithm() function, and use in strInfomation.Format() in YujinSRHostView.Cpp, there can be much difficulty. Therefore to make it simple, declare a variable to be used in Game Class, use this file in CYujinSRHostView.Cpp using m\_Cgame as Class one, so you can refer values of a variable in CYujinSRHostView.Cpp for robot controll.

## 6. Robot Control Program





## 6.1 Movement of Robot

### 6.1.1 A principle of Robot's Movement

PWM(Pulse Width Modulation) means a method to use by control of the ratio between High Voltage and Low voltage, conditioned as one period for sample Time. “Duty rate” means a ratio between high and low voltage, and user can control correct or converse direction of DC motors with this rate.

The most important purpose to use PWM method is because of generally used for its convenience to substitute control of Duty rate as showing the values of Digital 0~255.

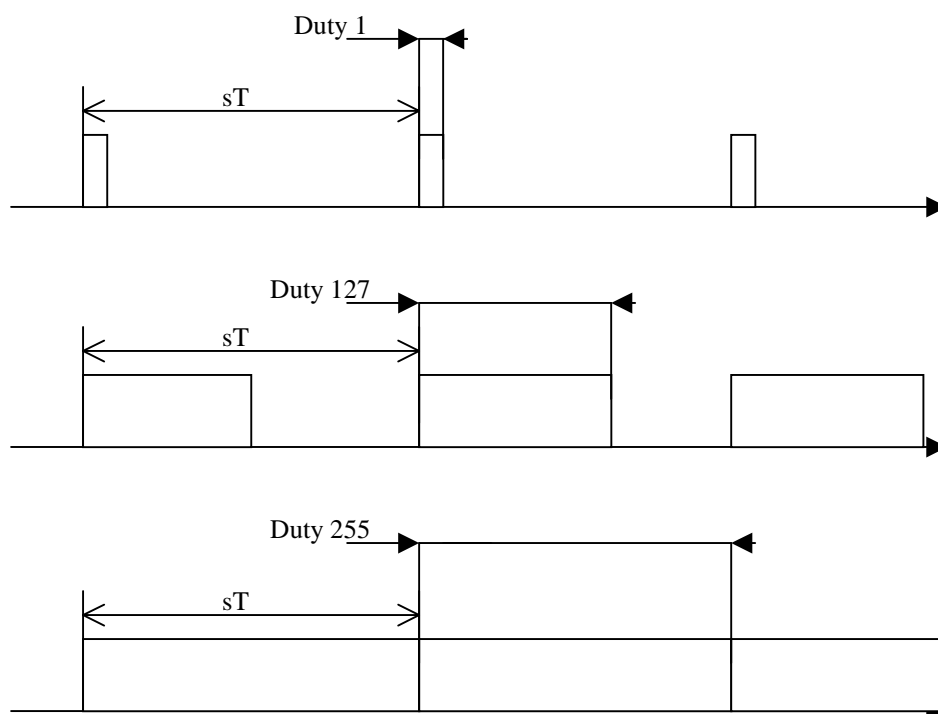


Figure 6.1 Duty rate

Therefore transmitted voltage to a motor by Duty Rate of PWM, user can control of the correct or converse direction of the motor.

Figure 6.2 describes a circuit to transmit voltage to a motor. This is also called as “H Bridge Driver Circuit Map” because of its shape of alphabet H.

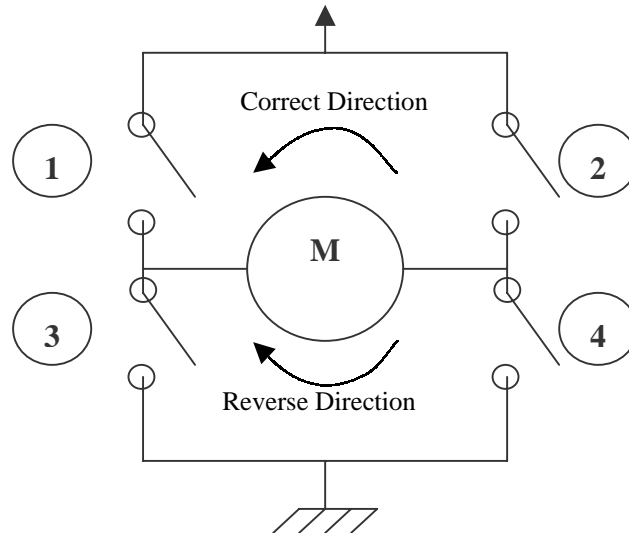


Figure 6.2 H-bridge Circuit Map for each direction

Switch No. 1, 4 ON: Correct rotation

Switch No. 2, 3 ON: Reverse rotation

Determined the direction in correct or reverse to the motor, finally the robot starts to move by its wheels.

### 6.1.2 Controller of a Robot

There're 3 different programs included in a robot, a program receives commands of Host\_Computer, the other program counts the numbers of Encoder, and another program controls the motor's movement by speed commands from Host\_Computer. Figure 6.3 is a drawing of block for the movement of the motor included in a robot.

A robot controller is made of PID controller. This PID controller can be easily realized. And each initial of PID has its concept. P has as proportion, I as integration, and D as differentiation.

So what a robot moves by a speed command received from the Host Computer can be a broad frame of the program for a robot.

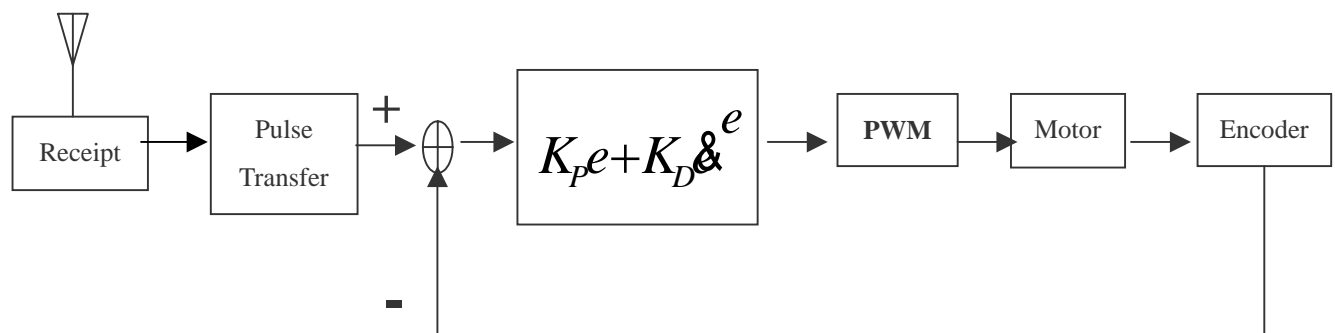


Figure 6.3 Block for Internal Controller for a Robot

### 6.1.3 Prior Controller

A prior controller has its function to calculate a speed command to move a robot. This controller means a program for strategy/tactics by calculating the position of a target and the current position of a robot so that it can play an actual soccer game.

As you can see on Figure 6.4 as below, Host Computer has a prior controller.

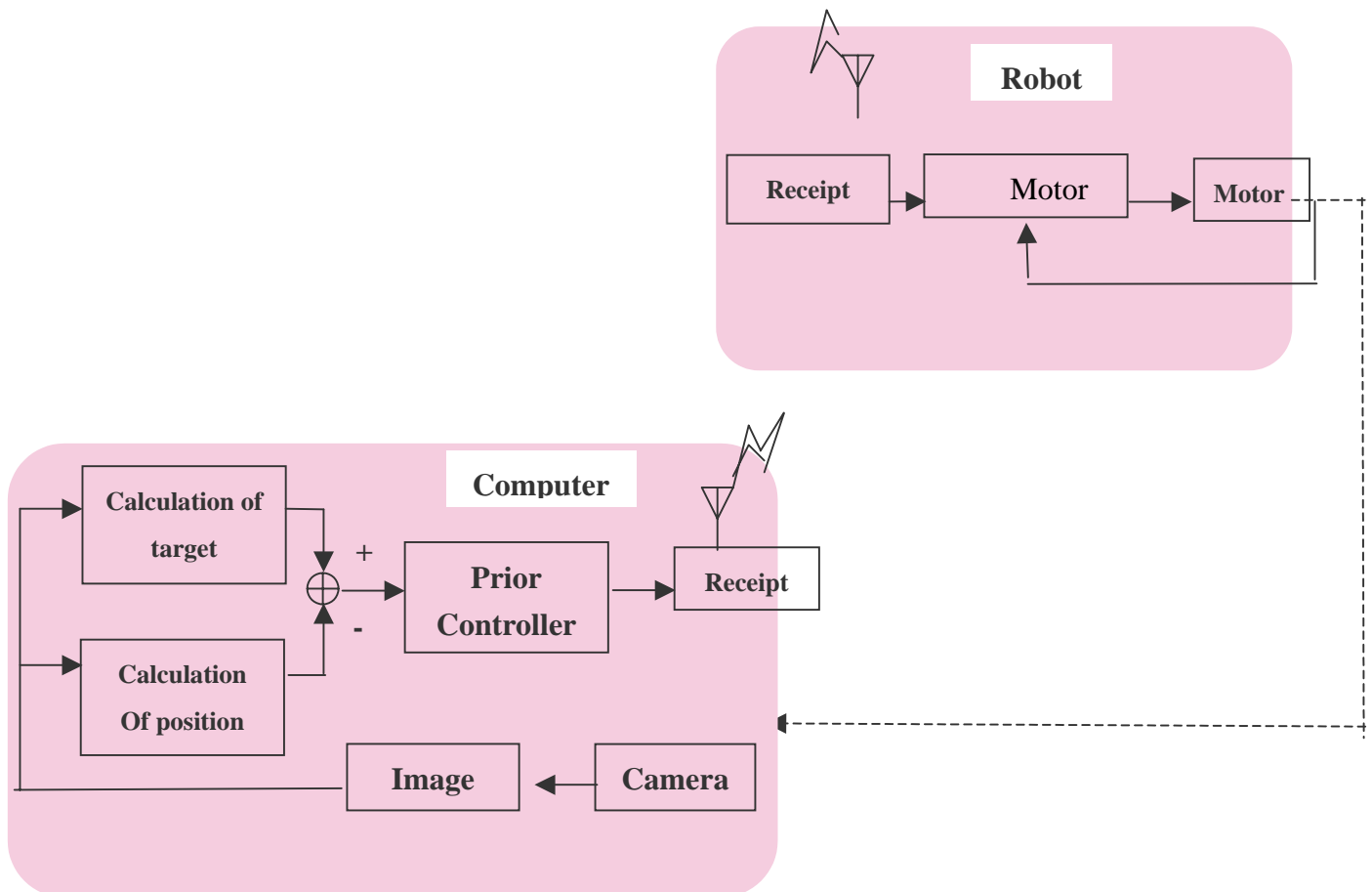


Figure 6.4. Block for a Total Controller

### 6.1.4 Description of Position for a Soccer Robot

Our Soccer Robot YSR-A operates its movement with 2 wheels. Figure 6.5 describes a position of the robot that moves with 2 wheels. So if described only as axis X & Y for the field, the direction of the robot can be represented as below figure.

Then, if you want to show the position of the Robot, it is also available to describe with axes of  $(y, x, \theta)$ .

If informed about the angle and the position of the center of a robot, it is also possible to make a controller which makes a robot moved to a desired position. By controlling the linear speed or for an angle to rotate, a user can move the robot to any position.

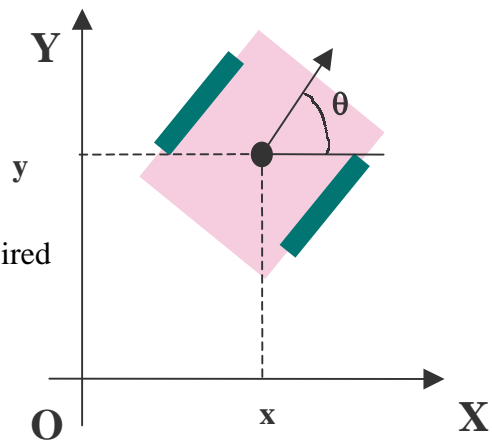


Figure 6.5 Coordinate System of A Robot

#### A Formular for Linear Velocity ( $v_C$ ) / Angular speed ( $\omega_C$ )

Linear Velocity of each wheel:  $v_L, v_R$

Prior Velocity: 
$$v_C = \frac{v_L + v_R}{2}$$

Angular Velocity: 
$$\omega_C = \frac{v_L - v_R}{D}$$

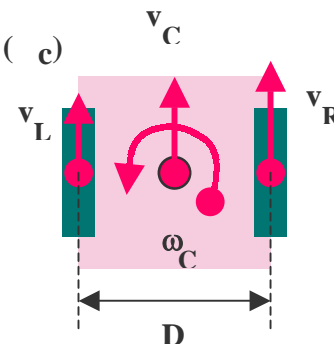


Figure 6.6 Relation for Prior and Angular Speed

## 6.2 Strategy Part

### 6.2.1 Robot Control Program

#### 1.1 Function of Velocity

Function Velocity transforms a speed command of a motor as a form to transmitter. All the values in this function can be the same as the values of PWM.

```
void CGame::Velocity(int whichrobot)
{
    int vl=MyRobot[whichrobot].VelocityLeft;
    int vr=MyRobot[whichrobot].VelocityRight;

    if ( vr > 125 ) vr = 125;      //Velocity Limte
    if ( vl > 125 ) vl = 125;
    if ( vr < -125) vr = -125;
    if ( vl < -125) vl = -125;

    vl += 127;                    // Velocity Range 0 ~ 255.
    vr += 127;                    // Translate Pwm Level

    command[0] = 0x00;
    switch(whichrobot){
    case HOME1 :
        command[1] = (unsigned char)vr;    //Save Communication array
        command[2] = (unsigned char)vl;
        break;
    case HOME2:
        command[3] = (unsigned char)vr;
        command[4] = (unsigned char)vl;
        break;
    case HOME3:
        command[5] = (unsigned char)vr;
        command[6] = (unsigned char)vl;
        break;
    case HOME4:
```

```
        command[7] = (unsigned char)vr;
        command[8] = (unsigned char)vl;
        break;
    case HGOALIE:
        command[9] = (unsigned char)vr;
        command[10] = (unsigned char)vl;
        break;
    }
    command[11] = 0x00;

    MyRobot[whichrobot].VelocityLeft = vl-127; //Robot Pwm Velocity
    MyRobot[whichrobot].VelocityRight=vr-127;
```

## 1.2 Function of Angle

Function Angle is for facing the robot to a desired angle.

This function makes a robot to move to the angle calculated between the angle of the robot and a desired angle.

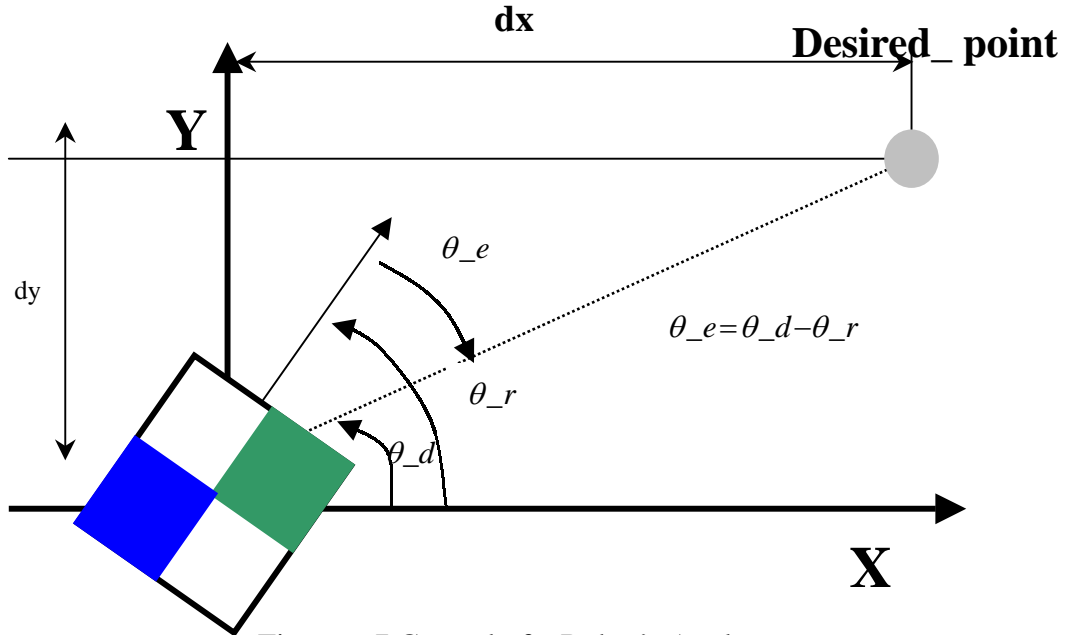


Figure 6.7 Control of a Robot's Angle

Figure 6.7 describes a fomular to control a robot. Conditioned the distance about poing X as dx and poing Y as dy, By using a function aTan2, we can calculate the angle( $\theta_d$ ) for a robot to face. As above figure, conditioned the current angle of the robot as an angle( $\theta_r$ ), will become an error( $\theta_e$ ) extracted the current angle of the robot from the desired angle.

$$\text{Angular Velocity } \omega_c = \frac{vl - vr}{d}$$

$$\text{Linear Velocity } vc = \frac{vl + vr}{2}$$

$$Vc = 0$$

$$\omega_c < 0 \text{ or } \omega_c > 0$$

Formula 6.1



As Formula 6.1, the robot rotates if a user controls its direction with only an angular speed in the condition that there's no linear speed. To rotate a robot, the direction between right motor and left motor should be in the opposite, so the calculation for this is same as following.

$$\begin{aligned} V_L &= -K_P \cdot \theta_e \\ V_R &= K_P \cdot \theta_e \end{aligned}$$

By using above principle, this is the function of Angle.

In this function, there is included a variable named AngleOfHomeRobot[whichrobot] saved the robot's angle. The angles can be described from 0~360 degrees.

```
void Angle(int whichrobot, int desired_angle)
{
    int theta_e, vl, vr;
    double Kp=15./90;
    theta_e = desired_angle - AngleOfHomeRobot[whichrobot]; //  $\theta_e = \theta_d - \theta_r$ 

    while(theta_e > 180) theta_e -= 360; //range of  $\theta_e$  -180~180
    while(theta_e < -180) theta_e += 360; // limited to

    vl = (int)(- Kp.*(double)theta_e); //  $V_L = -K_P \cdot \theta_e$ 
    vr = (int)( Kp.*(double)theta_e); //  $V_R = K_P \cdot \theta_e$ 

    Velocity(whichrobot, vl, vr); //Call Velocity function and save values for speed.

    if(FlagHomeRobotFound[whichrobot] == FALSE) //in case not found a
robot's WhichRobotStop(whichrobot); //position and stop the robot
}
```

The values of Kp is not decided, so set up the values by decreasing the errors to see the movement of the robot.

**The prior controller of YSR-A only uses P controller and controls the robot.**

### 1.3 Function of Position

The function Position is the one to let the robot moved to the desired position.

The function Angle only lets the robot rotated to a desired position, but the function Position makes the robot both faced and moved to a desired position.

Therefore both functions Position and Angle as basic movements in Soccer Robot System affect a failure or a win of the game.

The function Position needs to use both linear speed ( $V_c$ ) and angular speed ( $\omega_c$ ) to move the robot to a desired position.

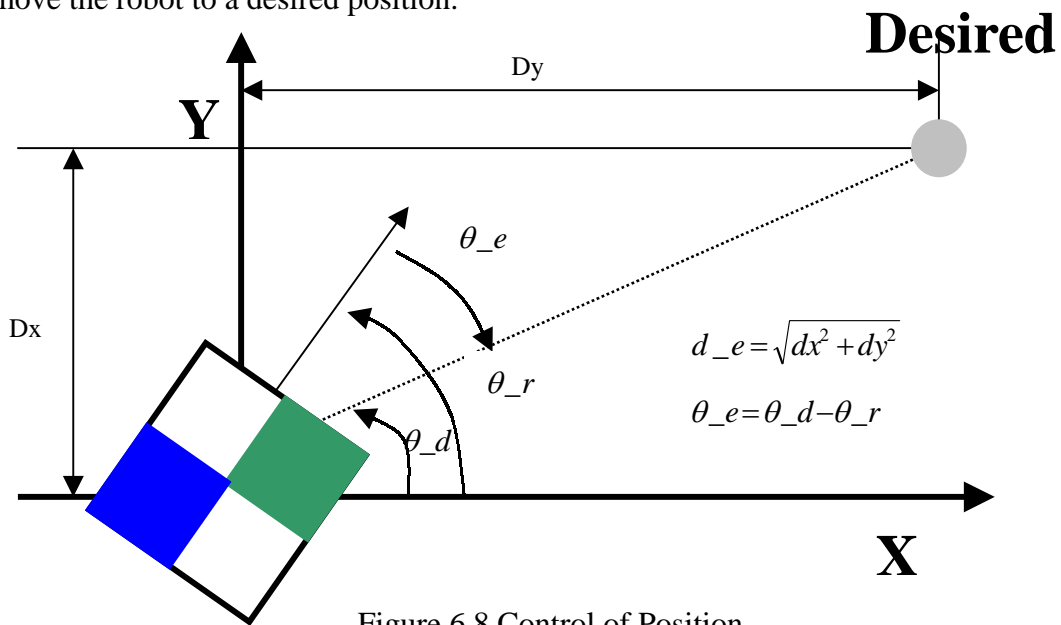


Figure 6.8 Control of Position

Figure 6.8 includes a formula about the error of distance except for the error of the function Angle and Position. For a robot to move to a desired position, the base line dy of dx is the height by Pythagorean Theroem and we can see the following formular.

$$d_e = \sqrt{dx^2 + dy^2}$$

$$\theta_e = \theta_d - \theta_r$$

The basic formula of control of Position should be calculated both speed about the linear and angle at the same time. That is, the formula to be calculated at the same time is same as below.

Based on the above formula, we can conclude the function of Position.

$$V_L = K_{Pdis\ tan\ se} \cdot d_e - K_{Pangle} \cdot \theta_e$$

$$V_R = K_{Pdis\ tan\ se} \cdot d_e + K_{Pangle} \cdot \theta_e$$

PositionOfHomeRobot[which][0](value of point X for a robot),

PositionOfHomeRobot[which][1](value of point Y for a robot) used in the functions of strategy/tactics describes each points of the robot.

All of the points of the robot mean the actual actual field, and this is represented as a value of x=0~150 y=0~130.

In the game of 3 vs 3, 3 robots are used in each team. If input 0, 1, 2 in the space named 'which robot', the points of each robot Home1, Home2, Home3 can be shown.

```
bool CGame::DeffensePosition(int whichrobot, double x, double y)
{
    int desired_angle=0, theta_e = 0, vl, vr;           // variable  $d_e, \theta_e$  statement
    double dx, dy, d_e, Ka=0.12, Kd=0.85;              //Kd =  $K_{Pdisanse}$ 
                                                         //Ka =  $K_{Pangle}$ 

    dx=x-PositionOfHomeRobot[whichrobot][0];          //distance difference of axis X
    dy=y-PositionOfHomeRobot[whichrobot][1];          //distance difference of axis Y
    d_e=sqrt(dx*dx+dy*dy);                             //by Pythagorean Theorem
                                                         //distance difference of robot and target position

    if(dx==0 && dy==0)                                  //prevents dx to be divided as '0'
        desired_angle = 90;
    else
        desired_angle = (int)(180./M_PI*atan2((double)(dy),(double)(dx)));//extracting  $\theta_d$ 

    theta_e = desired_angle-AngleOfHomeRobot[whichrobot]; //  $\theta_e = \theta_d - \theta_r$ 

    while(theta_e > 180)                                //limit of the range of  $\theta_e$  as -180~180
        theta_e -= 360;
    while(theta_e < -180)
        theta_e += 360;

    vl = (int)(kd*d_e - (Ka*theta_e));                 //control input of left wheel
    vr = (int)(kd*d_e + (Ka*theta_e));                 //control input of right wheel

    Velocity(whichrobot, vl, vr);                      //save the calculated speed of robot
}
```

## 1.4 Function of Goalie

To play a game, the more important robot is a goal-keeper robot than an attack robot. A goal-keeper robot takes most of the part of defense.

The role of the goal-keeper is to kick the ball inside of the goal area, and prevents the ball which the opponent's robot shoots. The strategy of an attack robot is also important, but if the goal-keeper robot does not play its role, the team cannot win the game even though the attack robot did so many shoots.

The function for Goalie included in **the strategy** has mainly movements about 3 different areas.

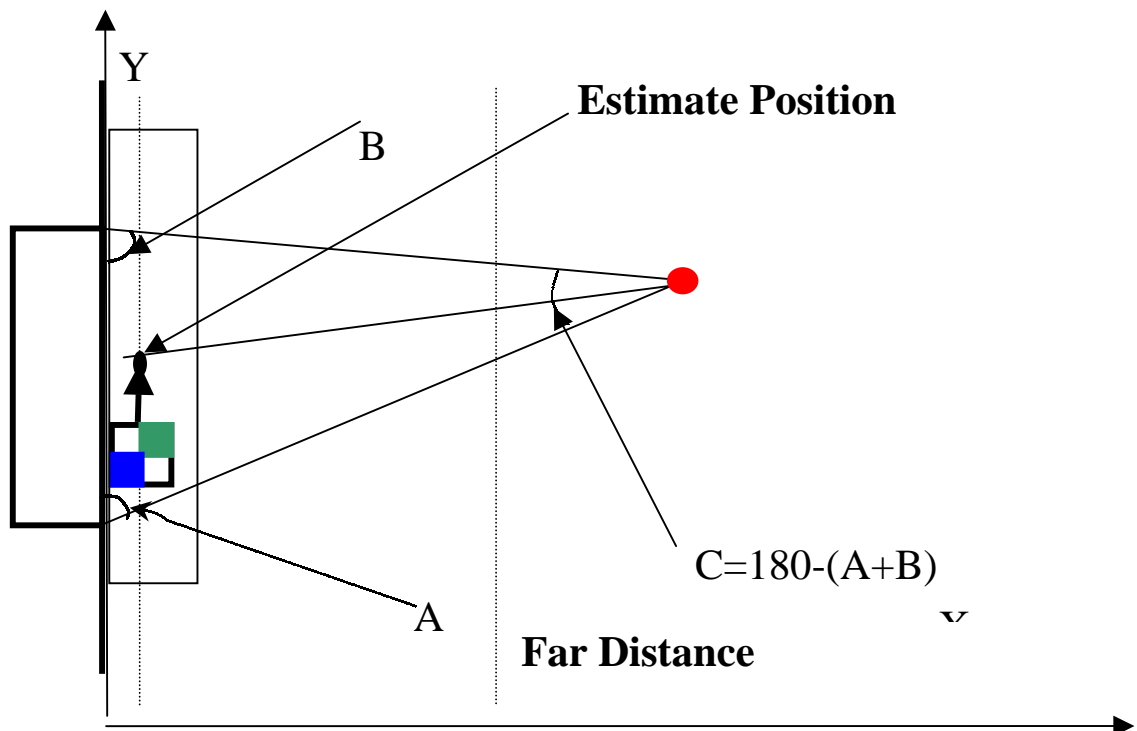
1. In case of the ball far from our goal line
2. In case of the ball near from our goal line
3. In case of the ball under or above of our goal line.

First,

1. In case the ball is far from our goal line

In this case, it is not necessary for a goalie to move sensitively according to each point Y of the goalie. For the goalie to prevent the ball entered into the goal line, as you see below figure, this function can calculate the angle between the ball and the both side of the goal line, and set the goalie stopped in front of the goal area from the center of the calculated angle. The goalie can position the expected angle in which the opponent's robot shoots the ball.

A goal-keeper robot can only move in the goal area. So we can move the robot by fixing point X and changing point Y as the approach angle of the ball.



Conditioned A as the angle between bottom line of the goal area and the ball, B as an angle between the top line of the goal area and the ball, and C as an angle of adding A and B, the angle can be extracted as above figure because the total add of a triangle should be 180 degrees.

The expected position for the ball to be approached can be a tangent between the line of C as divided as 2 and the perpendicular line from the angle and line for the robot to move from the goal area. This function is used to move the robot to the position to prevent the ball to the goal line when the opponent's robot approaches with the ball to our goal area far from. distance X of robot position

```
void CGame::Goalie(int whichrobot)
{
    double estimate_x, estimate_y, dx, dy, d_e; //Estimate position of point X,Y variables
                                                statement

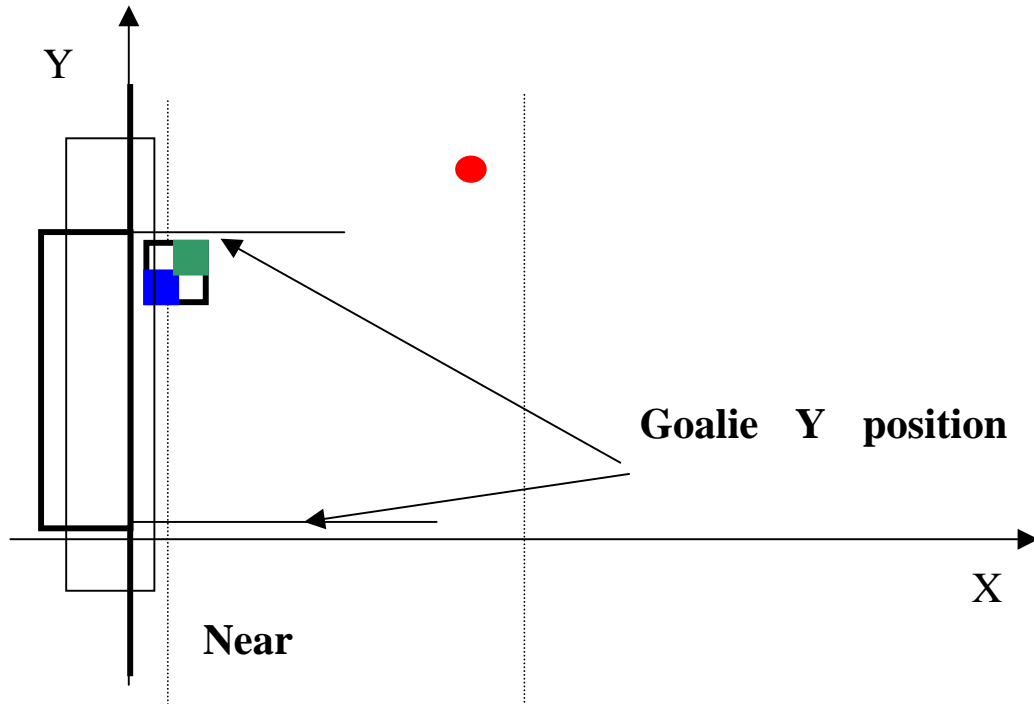
    if(PositionOfBall[0] > 100.)                //in case point X of ball is far
    {
        estimate_x = 0 + G_OFFSET;              //fix point X of robot to some point
        estimate_y = (int)(PositionOfBall[1] -   //get point Y in expected position of ball
            ( tan((hafeAngle(PositionOfBall[0], PositionOfBall[1])/180.0)*M_PI)
            * (PositionOfBall[0]-15)));

        dx = estimate_x - PositionOfHomeRobot[whichrobot][0]; //distance X of robot position
        dy = estimate_y - PositionOfHomeRobot[whichrobot][1]; // distance Y of robot position
        d_e = sqrt(dy*dy + dx*dx);                          //get the distance error

        if(d_e < 3 && G_OFFSET+8 > PositionOfHomeRobot[whichrobot][0] &&
            G_OFFSET-8 < PositionOfHomeRobot[whichrobot][0]) //in case of expected position
        {
            AngleOfPosition(whichrobot, estimate_x, 130);
        }
    }
    else //in case of no expected position
        GoaliePosition(whichrobot, estimate_x, estimate_y);
    return;
}
```

**In case the ball is near from our goal area, the goalie prevents the ball to move to the point Y instead of moving the robot to the expected position.**

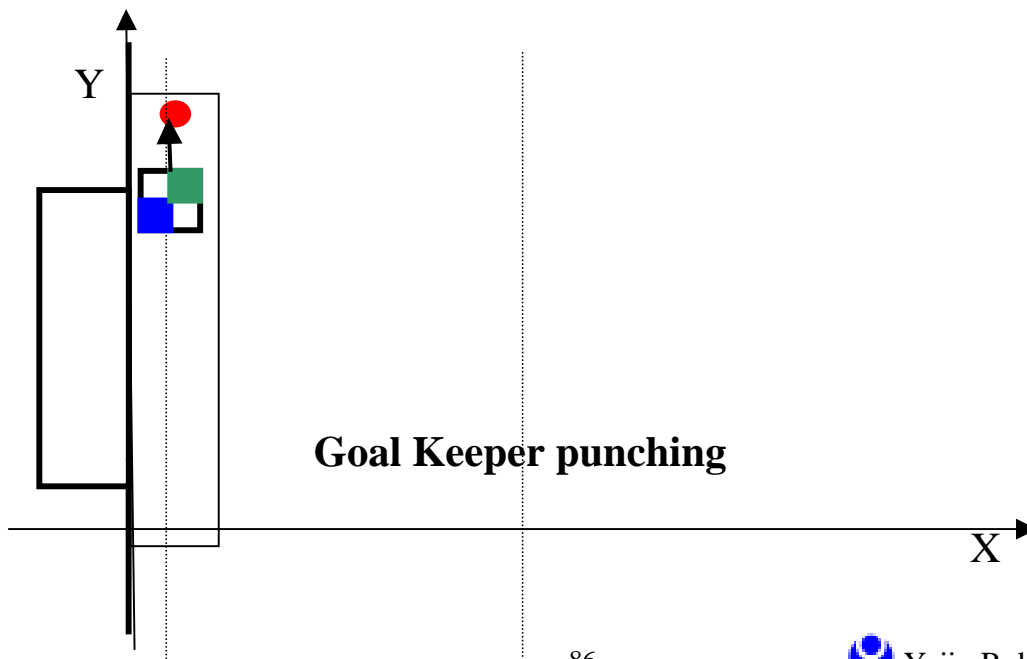
Therefore make the position of Y for the robot and point Y for the ball. If the goalie goes to the outside of the goal area (as below figure) to prevent the ball, make the goalie not to go over to the outside of the Goalie line. If a user does not select the limit for the goalie to move according to the poing Y for the ball, the goalie can move to the bottom or top line of the field. So it is the most effective to select the limit area of the Goalie to the goal line of area.



Finally, to make sure one thing is that in case the ball is near the bottom line or top line of the goal area. In this case, if the ball stays more than 10 seconds

in the goalie area, it can be a penalty kick, so prevents the ball not to stay there more than 10 seconds.

In most case, the ball can be entered there bounced or rolled by the side-wall of the field. So you should have to keep this in mind.



```

else
{
    estimate_x = G_OFFSET; //fixing point X
    estimate_y = PositionOfBall[1]; //Point Y as same

    if((PositionOfBall[1] > 80 && PositionOfBall[1] < 100) && //above the goal area
        (PositionOfBall[0] > 0 && PositionOfBall[0] < 10)) //status of goalie punching
    {
        if(GoaliePosition(whichrobot, estimate_x+2, estimate_y+8)) //above the ball position
        {
            //push the ball
            AngleOfPosition(whichrobot, estimate_x, 130); //fix the angle
        }
    }
    else if((PositionOfBall[1] > 30 && PositionOfBall[1] < 45) //above the goal area
        && (PositionOfBall[0] > 0 && PositionOfBall[0] < 10)) //status of goalie punching
    {
        if(GoaliePosition(whichrobot, estimate_x+2, estimate_y-8)) //under the ball position
        {
            //push the ball
            AngleOfPosition(whichrobot, estimate_x, 130); //fix the angle
        }
    }
    else{
        //Not punching status, the robot near the ball
        if(estimate_y > 80) //Limit above the point Y
            estimate_y = 80;
        else if(estimate_y < 50) //Limit above the point X
            estimate_y = 50;
        if(GoaliePosition(whichrobot, estimate_x, estimate_y)) //move the robot
        {
            AngleOfPosition(whichrobot, estimate_x, 130); //in case no distance error
        } //stay fixing the angle
    }
    return;
}

```

## 6.3 Robot Control Program

### 6.3.1 PID Controller

PID controller is mostly used because of the easiest way to embody among many controlling ways. Generally, a controller means a function to move some object to a desired position. Let's see an example for something to be controller, and look up into PID controller next.

Ex)

Let's make an example for the beginner of driving to drive the car to the center of the road in case the first line. The driver would not know about the correct speed either how much to change the handle.

This driver drives his/her car in the first line to change the handle if the car is inclined to the right or to the left. Before the driver have some experience, he/she will go through trial and error. Later if have some experience of driving, he/she will make the car correct in the first line.

In above example,

Object of control: the car

Target of control: Making the car correct in 1<sup>st</sup> line

As above, the object and target will be decided and each human being would drive a handle of the car if one has to possess hands to drive a car, eyes to recognize the 1<sup>st</sup> line, and brain to think and command of handling of the car.

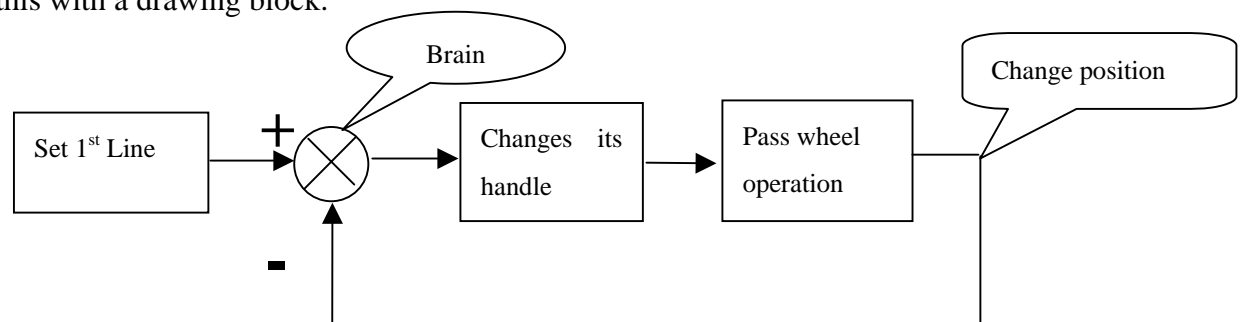
In this point, there can be one question. What is the difference between the beginner and the experienced driver even though both have above 3 essential things???

The difference comes from the process of the human being's brain. There will be a difference for the volume to pass the power to one's hands for the commands to change the handle.

As above procedure of a human being's driving, PID controller can be same applied for the object of all controls. Both the experiences drive has eyes, hands, and a brain and also the beginner has same things. In this same manner, PID controller can be made exactly same, but does not operate in the same movement.

The reason comes from the difference of compensation of the controller.

Contents explained above means totally a Feed Back System. Let's figure out through this with a drawing block.

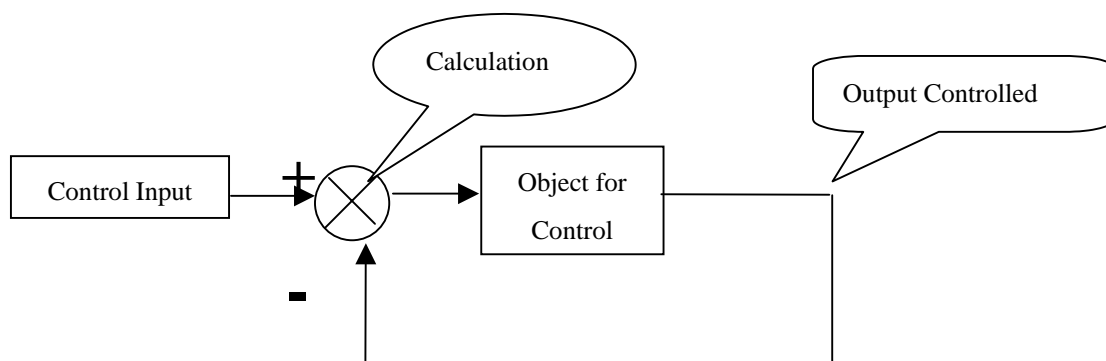




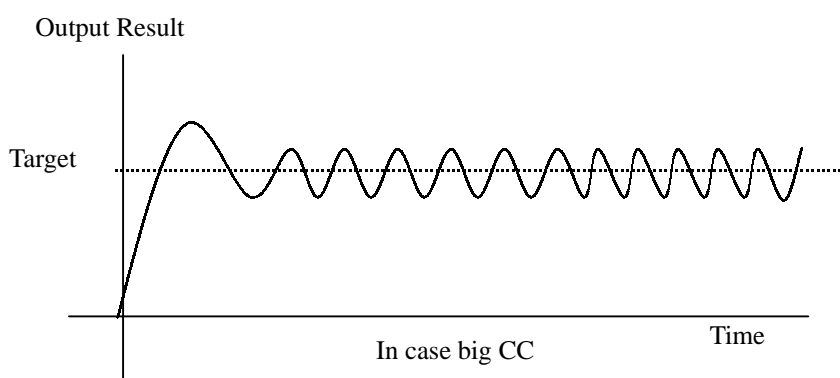
Changed as the position of the car finally in above drawing block, you will find that those values get returned to be included as a source to decide the controlling of the handle. Feed Back System means to be included a calculation to control the result of the set object.

To use PID controller in this **Feed Back System**, first, make a frame of the controller and set up a compensation coefficient according to it. **Compensation Coefficient** means that make it same between final output and input, and this is the most important purpose of this controller. If the output and input are not the same, the result of output is called as **Control Error**. About the values of this error, it should be reflected for how much the handle should be turned to compensate if the output and input are the same. Then we can extract **Compensation Coefficient** selected according to turning the handle.

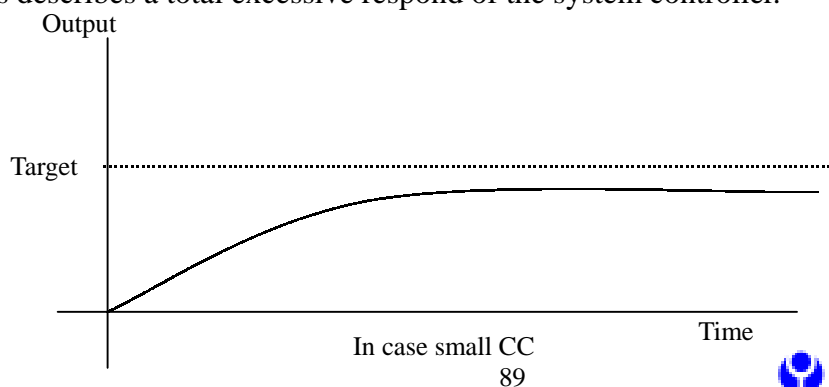
A prior controller of soccer robot uses P controller among PID controllers. Embodied as P controller, same as following.



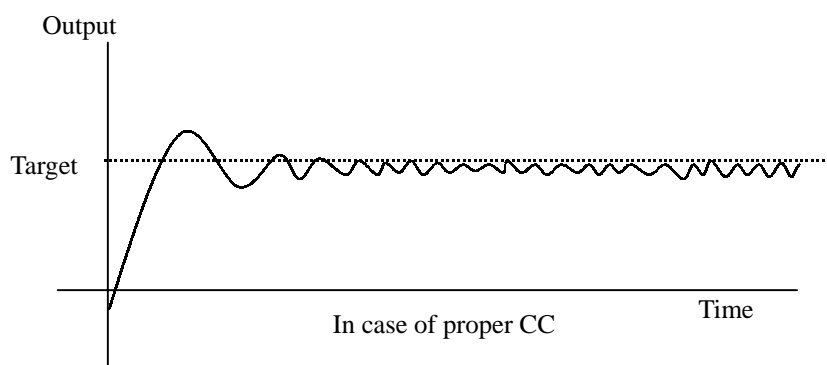
Using P controller, the values of the output of the **Compensation Coefficient** can be described as below.



As above figure, the output values are sloping after approaches to a target. This describes a total excessive respond of the system controller.



Above figure describes a rather low output not arrived to a target. This means the system controlled do not arrive to the normal status because of being slow.



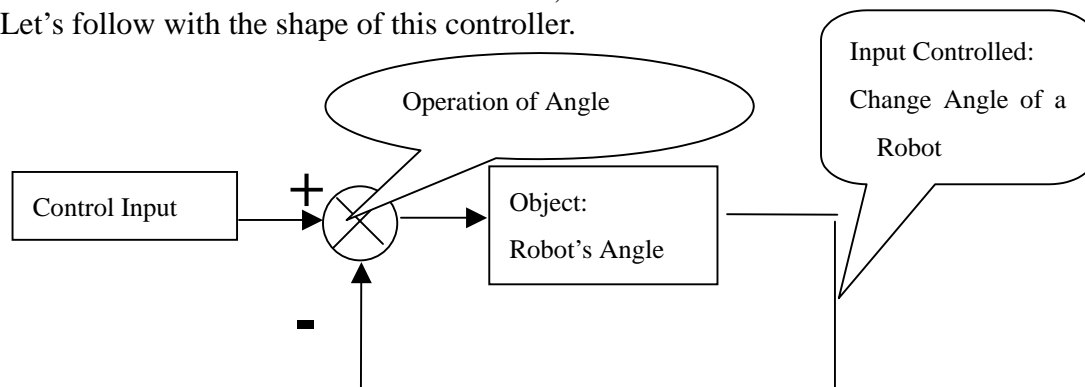
Above figure describes that the output values approaches a target in proper condition. However in case of using only P controller, the values do arrive almost to a target, but not to the point. This is called “Residal Error”. To remove this residual error, I integration control should be added and this is call as PI controller. D controller is used in case making quickly into proper condition about the sensitive response. In case of using PID controller by all, the curve seems to be idealistic. However you don’t have to use all of them. PID controllers can be used separately according to its character of each own system through only PI, PD, or P controller. A controller had better be simpler, so this will depend on your own control system.

### 6.3.2 Setting Error Gain

If you did already understand about PID controllers, let's practice how to use set up Gain between the function Angle and Position.

#### 1) Setting Gain of function Angle

The function Angle makes the robot faced to a desired angle. According the angle of the robot, there can be a variable named 'Ka' as a program code. All controllers of functions included are made of P controller, so it forms such as it. Let's follow with the shape of this controller.



Above operation as explained of the function Angle can change the angle of the robot by controlling the speed of both wheels.

Ka means a kind of Gain and the object output for the function Angle will be the angle to be faced.

The operation is made of numeral formulas, Gain Ka calculated as invariables is delivered to the both wheels of the robot as a speed value.

```
void CGame::My_Strategy()
{
    static int NextCorner = 0;
    count2++;
    preY.add(count2, (int)PositionOfBall[1]);
    preX.add(count2, (int)PositionOfBall[0]);

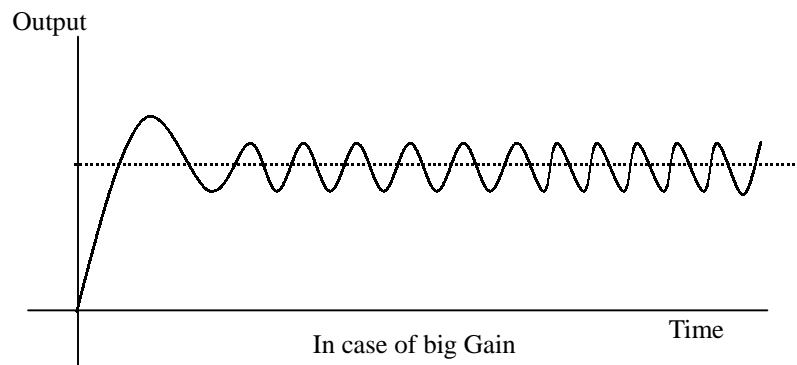
    Angle(HOME1, 90);           ← Call the function
    SendCommand(command);
}
```

Then, let's start to operate the program with the value of Ka as '1'.

```
void Game::Angle(int whichrobot, int desired_angle)
{
    .....
    double Ka=1;           ← convert of Ka values
    .....
    vl = (int)(-Ka*(double)theta_e);
    vr = (int)(Ka.*(double)theta_e);
    .....
}
```

After setting the robot color of Home1, put [Ready] and then click [Start] button so that the program is processing to check the condition of the robots.

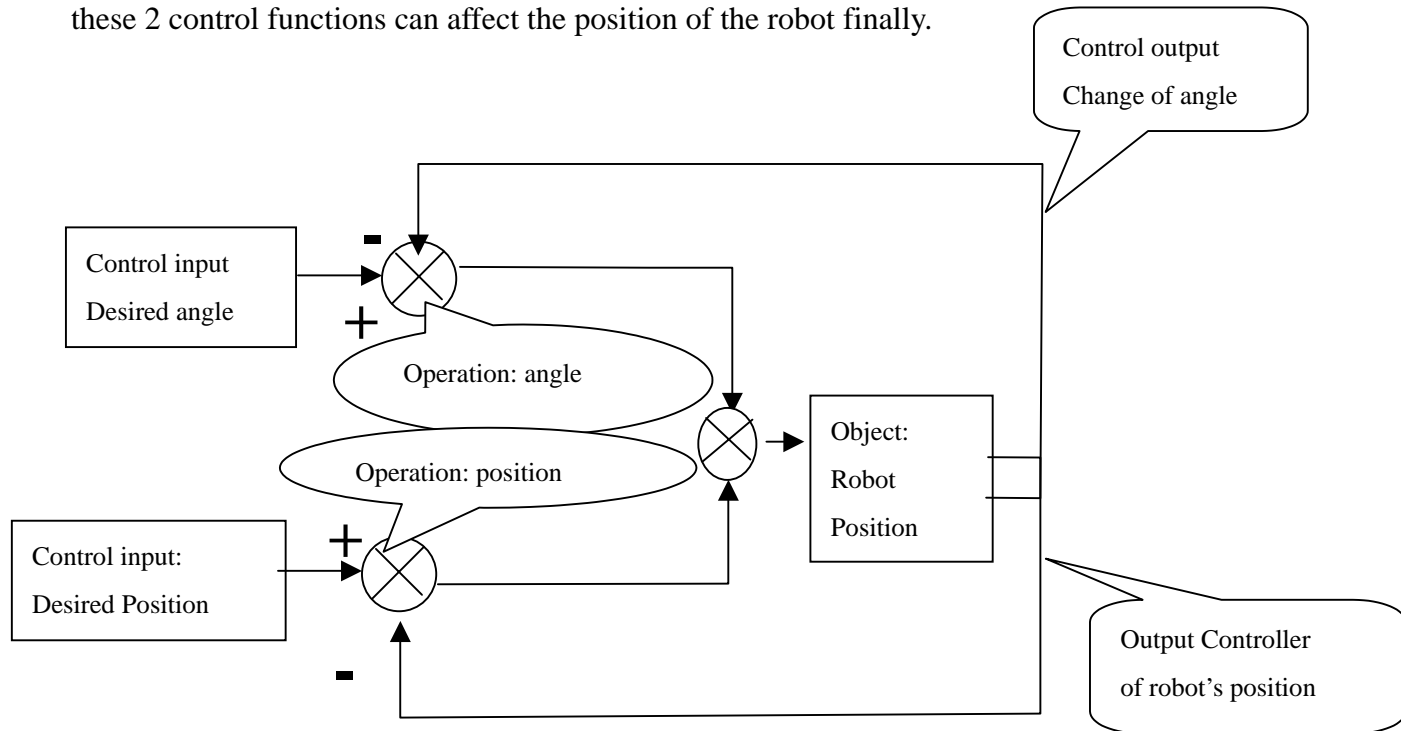
Then, you will find that the robots will be screened shaken. The following graph can be described as its response.



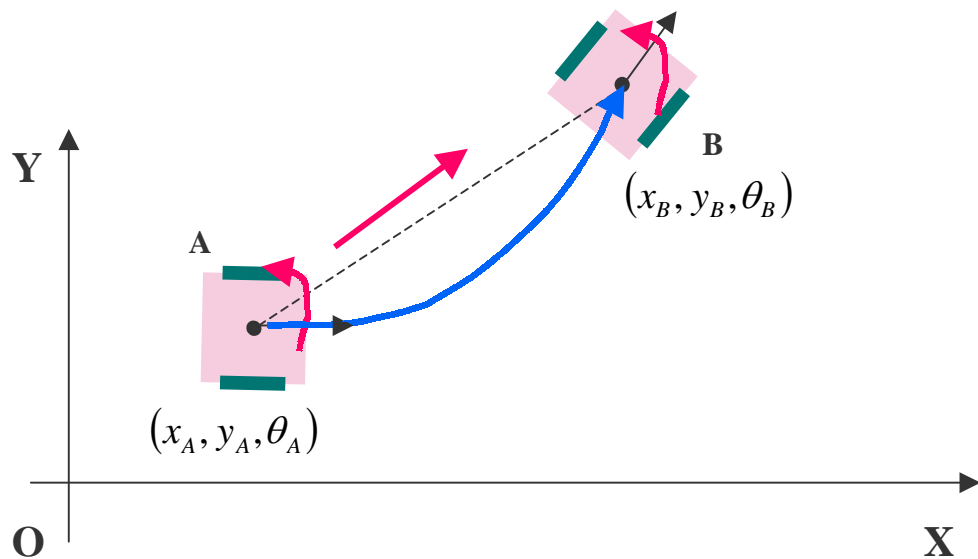
Decreased the values of Ka slowly, let's find the correct values of Ka for the robot to rotate quickly at most in the stable condition of the robot which is not vibrated.

## 2) Setting Gain for a function Position

The function of Position needs to control ahead and angular speed both. That is, these 2 control functions can affect the position of the robot finally.



As above frame, two controllers are made up. That is, the sum of control between the position and the angle can be a value for the speed of a robot. The function Position controls the angle and position of a robot at the same time. To move the robot to a desired position, it can rotate to the angle and go straight ahead.



To move a robot from point A to point B, the robot will be moved drawing a blue circle by controlling the angle and position of the robot with using a suggested controller.

There are 2 different controllers and there should be also 2 Gains about these controllers so that we can use them.

```
void Game::My_Strategy()
{
    static int NextCorner = 0;
    count2++;
    preY.add(count2, (int)PositionOfBall[1]);
    preX.add(count2, (int)PositionOfBall[0]);

    Position(HOME1, 65, 75);
    SendCommand(command);
}
```

← Call the function

In the above function, Kd means Distance and Gain, Ka means Angle and Gain. The movement of a robot depends on if amended Kd and Ka. First, in case Gain of the angle is small and Gain of the position is large, the robot has a big tendency to go straight.

Therefore the robot won't move to the desired position and only from the front to the back in a large range. In the opposite, in case the Gain of the angle is too large and the Gain of the position is too small, the robot moves slowly and fits its angle shaken so much. Therefore let's control the values of these 2 Gain.

First, set the Gain of the position as 0.01, and the Gain of the angle as 1. After watching the movement of the robot, set the Gain of the angle less and less to the proper point.

```
void Game::Position(int whichrobot, double x, double y)
{
    .....
    Kd = 0.01;
    Ka = 1.0;

    vl = (int)(Kd*d_e - (Ka*theta_e));
    vr = (int)(Kd*d_e + (Ka*theta_e));
    .....
}
```

If you already set the Gain of the angle properly, set the Gain of the position. By controlling of the values from 0.01 slowly, set the most proper Gain.

Way of Gain as explained above is just one of many methods to set up, so there're other ways to. That is, there're ways to set up the Gain by control program, or by simulation program, etc. Let's make up the system for the robots to operate their best performances by using any way of use as you want.

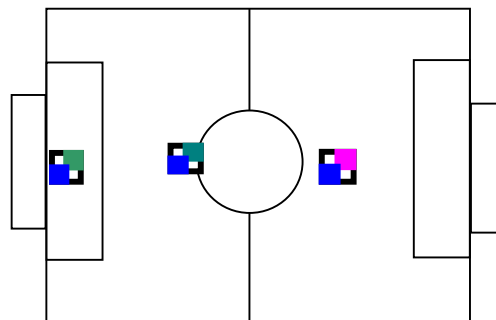
## 6.4 Strategy

### 1) What is strategy?

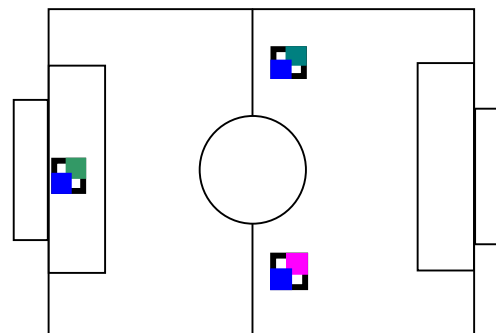
Robot Soccer is not a game playing with one by one robot, but a match at which robots more than 3 cooperate and compete which team has more efficient algorithm. Basic function of the robots' movement is also necessary, but more important thing is to how the robots cooperate well and materialize in the match like the attack robot cannot be the one for the victory.

In case of attack, the host computer sends a command to pass the ball to a robot where no obstacle is if the robots cannot dribble or shoot because of the defense robots of the opponent team. This series of movement can be an example of each team's strategy.

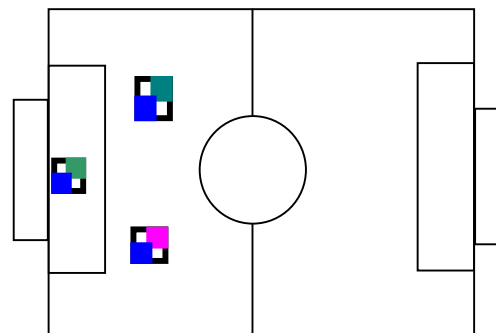
Such as this manner above, it will be more efficient to make a formation in order for robot soccer to control the movement of robots. Assumed that playing with 3 robots per one team make a basic formation, it will be the same as following.



1) 1-1-1 (attack –defense-goalie)



2) 2-0-1 (attack -defense -goalie)



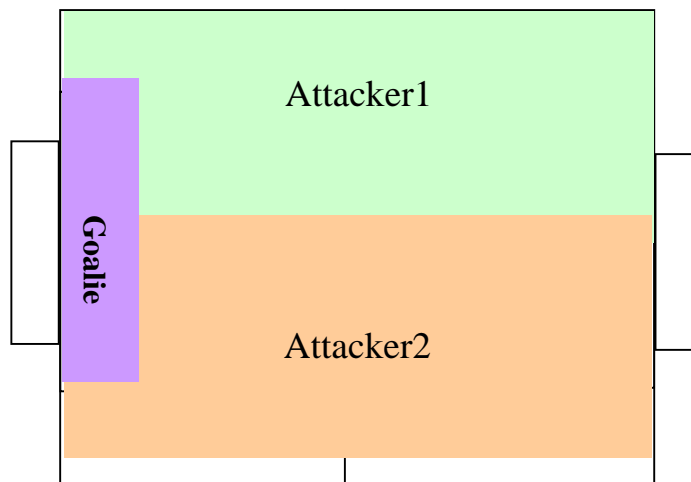
1) 0-2-1 (attack -defense -goalie)

- 1) No. 1 formation is to prepare sudden attack of the opponent robot arranging the local defense robot and attack robot ahead in the positions of attack, defense, and goalie.
- 2) No. 2 formation is to play attack-centered 2 robots for attack and 1 goalie
- 3) No. 3 is to play defense-center 2 robots for defense and 1 goalie.

It would be efficient to change above basic formations mixed properly according to the position of the ball.

## 2) Construction of Strategy

Let's look into how "Miro99.cpp" strategy is organized.



Above figure means the construction of strategy inside "Miro99.cpp" program. This is programmed for the robots to attack only in the selected position of the ball playing local defense. Describes as a program for above figure, it's same as below,

```
void Game::Attack(int whichrobot1, int whichrobot2)
{
    if(PositionOfBall[0]>151.0 || PositionOfBall[0]<-1.0)
        StopRobot();

    if(PositionOfBall[1] > 55.0){
        Kick(whichrobot1);
        Position(whichrobot2, PositionOfBall[0]-20.0, 40.0);//17.0
    }
    else if(PositionOfBall[1]<75.0){
        Kick(whichrobot2);
        Position(whichrobot1, PositionOfBall[0]-20.0, 90.0);//17.0
    }
}
```

There will be more ways to embody strategies/tactics except of this. Let's make a soccer robot system moving actively more and more using many different ideas and information.