

---

DEVELOPMENT AND RESEARCH ON INTELLIGENT  
MOBILE ROBOT SYSTEM

By  
FENGMING YANG

Bachelor of Engineering  
Southeast University  
Nanjing, China  
July, 1993

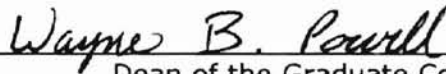
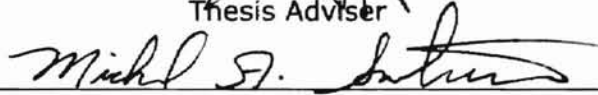
Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
in partial fulfillment of  
the requirements for  
the Degree of  
MASTER OF SCIENCE  
May, 2000

DEVELOPMENT AND RESEARCH ON INTELLIGENT  
MOBILE ROBOT SYSTEM

Thesis Approved:



Thesis Adviser



Dean of the Graduate College

## ACKNOWLEDGMENTS

I would like to sincerely thank my advisor Dr. Gary Yen for his enthusiastic support, constructive guidance, encouragement, as well as friendship throughout the time of acquaintance. I also want to thank Dr. Soderstrand and Dr. Johnson for being my committee members.

I would like to express gratitude to Phung Meesad, who is currently a PHD student in EE, for discussing with me on knowledge representation and fuzzy logic system parts of the research. My sincere appreciation also extends to Travis Hickey for his precious advices on my thesis.

I wish to express my appreciation to Dr. Soderstrand for lending us parts of their hardware. And give thanks to ECEN lab technician Lee Clark for helping us improve the lighting condition of the system.

I thank my parents for their encourage and understanding. I thank my elder brother for his constant guidance and faith in me. Finally, appreciation is extended to my wife for her love and patience.

## TABLE OF CONTENTS

<b>PART 1. DEVELOPMENT OF THE INTELLIGENT ROBOT SYSTEM .....</b>	<b>1</b>
<b>Chapter 1 System Introduction .....</b>	<b>2</b>
1.1 Hardware Parts.....	2
1.2 Software Parts.....	6
<b>Chapter 2 Computer Vision in Intelligent Robot System .....</b>	<b>8</b>
2.1 Video Capture.....	8
2.2 Image Processing .....	10
2.3 Pattern Recognition.....	18
2.4 Real Detection Effect .....	20
<b>Chapter 3 Lighting Problems .....</b>	<b>22</b>
3.1 Lighting Techniques.....	23
3.2 Brightness.....	23
3.3 Nearest Neighborhood Method to Solve the Lighting Problem .....	25
<b>Chapter 4 Serial Communication and Distributed Computing .....</b>	<b>30</b>
4.1 Serial Communication .....	30
4.2 Distributed Computing .....	32
<b>Chapter 5 System Evaluation and Research Perspectives .....</b>	<b>37</b>
5.1 Difficulties in Real Time Robot System Research .....	37
5.2 Possible Researches.....	38
5.3 An Introduction to the Windows Interface.....	38
<b>PART 2. LITERATURE REVIEWS AND PROPOSED RESEARCH .....</b>	<b>41</b>
<b>Chapter 6 Knowledge Representation.....</b>	<b>42</b>
6.1 Rule-Based Representation.....	42
6.2 Frame-Based Representation .....	44
6.3 Object Oriented Programming .....	44
<b>Chapter 7 An Introduction to Reinforcement Learning.....</b>	<b>46</b>
7.1 What is Reinforcement Learning .....	46
7.2 Q-Learning.....	46
<b>PART 3. EXPERIMENTAL RESULTS.....</b>	<b>51</b>
<b>Chapter 8 Fuzzy Behavior Controller.....</b>	<b>52</b>
8.1 Task Description .....	52
8.2 Why Fuzzy Logic System is Needed .....	53
8.3 Fuzzy Behavior Controller.....	54
8.4 Experimental Result .....	58

<b>Chapter 9 Knowledge Representation in Intelligent Robot System .....</b>	<b>59</b>
9.1 Frame Knowledge Representation in Robots Playing Soccer Game .....	59
9.2 Behavior-Based System .....	60
9.3 Finite State Machine Implementation .....	61
<b>Chapter 10 Robot Learning in Intelligent Robot System.....</b>	<b>62</b>
<b>Chapter 11 Summary and Future Work.....</b>	<b>65</b>
11.1 Summary and Contributions .....	65
11.2 Future Work.....	66
<b>References.....</b>	<b>68</b>

## LIST OF TABLES

Table 3.1 Nearest Neighborhood Method to Solve the Illumination Problem .....	27
Table 4.1 Operation of The CAsynSocket Class .....	35
Table 4.2 Overridable Notification Functions of the CAsynSocket Class.....	36
Table 7.1 Value Table of Q Learning .....	49
Table 10.1 Robot Learning .....	63
Table 10.2 Robot Learning Result .....	64

## LIST OF FIGURES

Fig. 1.1 Intelligent Robot System .....	3
Fig. 1.2 A Robot.....	3
Fig. 1.3 Hat of a Robot.....	4
Fig. 1.4 A Playground with 3 Robots and a Ball on it .....	5
Fig. 2.1 Video Capture Device Architecture .....	8
Fig. 2.2 Color Settings .....	11
Fig. 2.3 Binary Dilation with a Mask of 3 .....	12
Fig. 2.4 Binary Erosion.....	13
Fig. 2.5 Opening Operation.....	13
Fig. 2.6 Convex Set.....	14
Fig. 2.7 Convex Processing.....	15
Fig. 2.8 Component Labeling .....	16
Fig. 2.9 Pseudo Code for the Recursive Algorithm.....	16
Fig. 2.10 Pseudo Code for the Sequential Algorithm.....	17
Fig. 2.11 Size Filtering .....	18
Fig. 2.12 Real Detection Effect.....	20
Fig. 3.1 The RGB Intensities of a Pixel.....	24
Fig. 3.2 Increase the Brightness.....	24
Fig. 3.3 Over-lighting Causes Saturation of Red and Green Intensities .....	25
Fig. 3.4 Algorithm to Minimize Lighting Effect .....	26
Fig. 3.5 Real Image Detection Effect.....	29
Fig. 5.1 The GUI Interface of Intelligent Robot System.....	39
Fig. 7.1 The Standard Reinforcement-learning Model. ....	46
Fig. 7.2 Q-learning Algorithm.....	47
Fig. 7.3 A Grid World .....	48
Fig. 7.4 An Optimal Policy in the Grid World.....	48
Fig. 8.1 Diagram of Robot Moving from A to B .....	52
Fig. 8.2 The Input-output Diagram .....	54
Fig. 8.3 Membership Functions for Input Variable Distance .....	55
Fig. 8.4 Membership Functions for the Input Variable Angle.....	55
Fig. 8.5 Membership Functions for the Output Variable Speed.....	56
Fig. 8.6 Membership Functions for the Output Variable Turning Speed .....	56
Fig. 8.7 Rules of the Fuzzy System.....	57
Fig. 8.8 A Trajectory Record of Robot Moving to Position .....	58
Fig. 9.1 A Hierarchical Architecture of Robot Behaviors .....	60
Fig. 10.1 State Set and Action Set of Robot Learning to Shoot .....	62

## **Part 1. Development of the Intelligent Robot System**



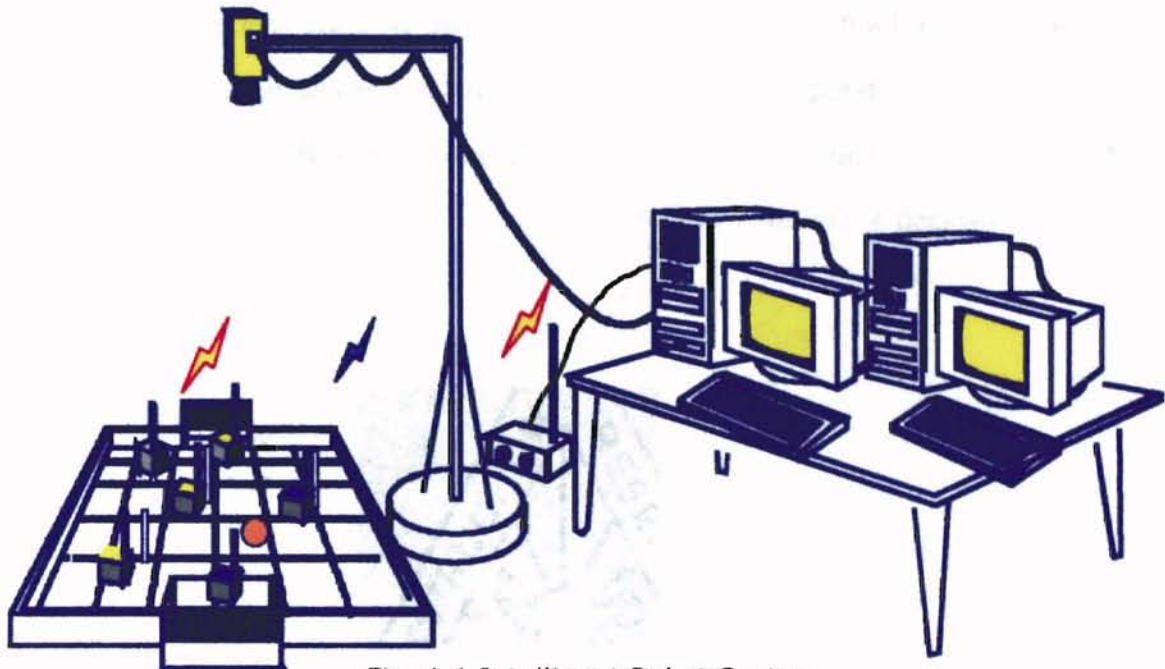
## **Chapter 1 System Introduction**

In artificial intelligence research, numerous ideas on learning paradigm were proposed. If we conduct a literature search using "machine learning" or "reinforcement learning" as keywords, we can get hundreds of hits. However, most of these ideas were implemented only by simulation at most. In artificial intelligence research, there is a big distance between theoretical possibility and real-world practice mainly due to the limit of available experiment testbeds. Theoretically, human can land on the Mars easily. However, billions of dollars have spent yet we haven't touched a rock on the Mars. In order to learn riding a bicycle, we need to have a bicycle. This inspired our effort to develop a real robot system for theoretical validation and experimental demonstration.

Robots playing soccer is a very good environment to engage in many different artificial intelligence research topics, such as robot learning, robots cooperation, and knowledge transferring. Therefore we bought a set of equipment and developed a research platform on it.

### **1.1 Hardware Parts**

Fig. 1.1 shows the system composition. The system consists of some robots, a playground on which robots can move around, a video camera which is used to monitor the activities of robots on the playground, two computers, two vision boards which are plugged in the computers, a wireless transmitter and a ball. The output of the video camera is split into two branches; each of them is connected to a vision board, which is plugged in a computer. The wireless transmitter is connected to a serial port of the computer. It is used to control the movement of robots by sending orders to them.



*Fig. 1.1 Intelligent Robot System*

This system was originally designed for robots playing soccer. People call it "MIROSOT". There is a world wide competition held annually. In the competition, different teams of robots come from around the world to play against each other. The equipments we bought meet the requirements of the competition. We can use it to take part in the competition and also use it for research purpose. We mainly use it for research.

In the following, we'll introduce the devices one by one briefly.

#### Robots



(a)



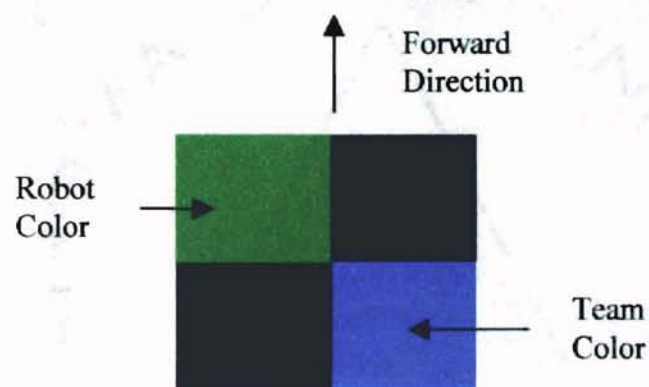
(b)



(c)

*Fig. 1.2 A Robot*

A picture of a robot is shown in Fig. 1.2. Fig. 1.2(a) shows the bird's view. Fig. 1.2(b) is the side view. Fig. 1.2(c) is the bird's view of the robot with a hat on it. Each robot has two wheels, left wheel and right wheel. They are independent of each other and can be controlled by the program separately. The speed of each wheel is scaled from -127 to +127. When the speed is negative, the robot will move backward.



*Fig. 1.3 Hat of a Robot*

Each robot has a different colored hat to identify it. A picture of a robot hat is shown in Fig. 1.3. On the hat, there are two different colored squares. We call the color of the right lower square the team color. The team color of two or three robots can be same, which means they belong to the same playing side. The color of the left upper square is called robot color. Different robot in the same team should have different robot color to identify itself. Robot color is used to identify the specific robot. The material for the hat should not be light reflecting. The surface of the hat should not be very smooth to avoid reflecting. The forward movement direction of a robot is also shown in the figure.

### Playground

The playground is painted to black. The surrounding borders are painted to white. The surface of the playground also should not be light reflecting. A picture of the playground is displayed in Fig 1.4.



*Fig. 1.4 A Playground with 3 Robots and a Ball on it*

#### **CCD camera**

The CCD Camera is made by Samsung. Its model number is SAC410NDX. It can output NTSC standard signal.

#### **Vision board**

The vision board should provide Microsoft windows programming driver. It provides full resolution of 640x480 dimension or 320x240 half dimension in 24-bit RGB format.

#### **Computers**

The two computers are connected together through the internetworking hub using TCP/IP protocol. One computer acts as master mode. It takes care of principal processing tasks. The other computer works as slave mode. It is mainly used to do

some supplementary work such as recording video. We'll discuss distributed computing later.

### **Wireless transmitter**

It is connected to one serial port of the computer to send wireless signal to robots. At present, the signal transmission is one way. Computers cannot accept data from robots.

We purchased the equipment (except the computers) from Micro Adventure Inc., (Korea).

## **1.2 Software Parts**

The software of the system is partitioned into the software on the robot and the main control software on the computers. The software on the robot is programmed using Assembler for the 89C52 microprocessor in the robot. The software on the computer is programmed using Visual C++ for Windows.

Since the software on the robot mainly receives orders from the computer and drives the motors to move around. We don't need to modify it. In addition software also comes with the equipment for the computer. It was programmed using VC 1.52. But most functions are not working. We need to ameliorate it entirely.

The video camera captures the video images, and sends them to the computer. The computer needs to perform image processing, pattern recognition, apply some algorithms to output orders by the wireless transmitter to control robots. Therefore, the main tasks of the development are divided into the following parts:

- Computer Vision, which includes video capture, image processing, and pattern recognition;
- Serial communication and distributed computing; and

- GUI Windows interface.

In the following chapters of Part One, we'll discuss them one by one.

## Chapter 2 Computer Vision in Intelligent Robot System

### 2.1 Video Capture

The vision board supports NTSC (USA) standards. Before we can use it on the PC, we should install a driver that comes with the board to support Microsoft Windows programming. Microsoft divides the architecture of Video capture device into four different logical channels: external in, video in, video out, and external out. The destination or source of each channel is the frame buffer that is part of the video-capture hardware [7]. The video capture device architecture is shown in Fig.2.1.

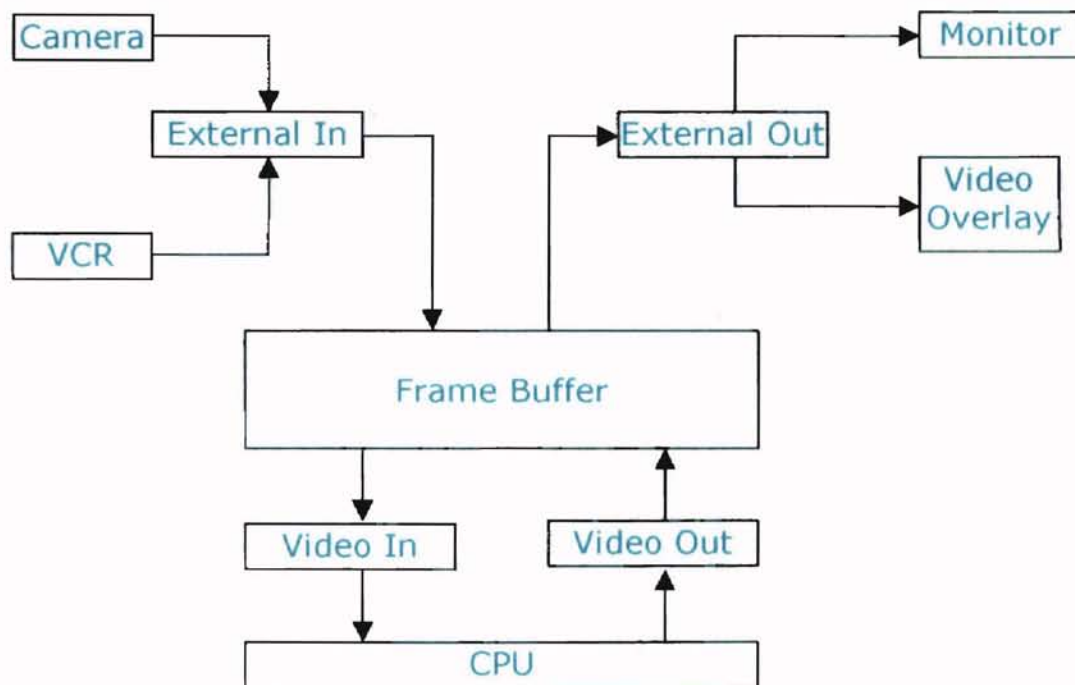


Fig. 2.1 Video Capture Device Architecture

The video capture channel (External In) is a source of video information placed in the frame buffer. The video source might be a video camera, video player, or television

tuner. The format of both the incoming signal and the data placed in the frame buffer is controlled by the video capture hardware.

The video capture device can display the frame buffer data by using the video display channel (External Out). In practice, this could be displayed a second monitor or a video overlay device.

The device driver and application will use the video in channel (Video In) to transfer the video data to application-supplied buffers.

The device driver and application can play captured data by using the video out channel (Video Out) to transfer data back into the frame buffer. Playback through this channel might be to review a sequence just captured or to play data from a file.

A capture driver can implement two methods for viewing an incoming video stream: preview mode and overlay mode. In overlay mode, video is displayed using hardware overlay. Using an overlay does not require CPU resources. In preview mode, frames are transferred from the capture hardware to system memory and then displayed in the capture window using GDI functions. The preview mode consumes substantial CPU resources.

In Microsoft Windows Visual Studio, there is a class `AVIcap`. We can use this class to program the video capture device. `AVIcap` provides a flexible interface for applications. It has two different callback functions for programmer to program the video data.

1. **Stream Callback:** `AVIcap` calls this procedure during streaming capture when a video buffer is filled. The capture window calls the callback function before writing the captured frame to disk. This allows applications to modify the frame if desired.



2. **Frame Callback:** AVICap calls this procedure when the capture window captures preview frames. The capture window calls the callback function before displaying preview frames. This allows an application to modify the frame if desired.

In our application, we used both stream callback and frame callback modes.

## 2.2 Image Processing

The video images are captured in 24-bit bitmap mode. Each 3-byte triplet in the bitmap array represents the relative intensities of blue, green, and red, respectively, for a pixel.

Currently, there are various image-processing techniques. Each of them can achieve a different effect. What we want to do is for the computer to identify different objects and their respective locations. This is actually a kind of machine vision or computer vision. For our purposes, we only apply the techniques relevant to realize machine vision. Usually, we need the following steps to realize computer vision:

- Color setting;
- Thresholding;
- Dilation and erosion;
- Convex processing;
- Component labeling;
- Size filtering;
- Center calculation; and
- Pattern recognition.

### 2.2.1 Color setting

Color setting is to designate the color ranges of different objects. Simply put, it is to identify "who is who". Color setting is a very important procedure in machine vision.

If color setting is not appropriate, we may misidentify an object, or confuse an object with another. To do color setting, we use mouse to point to an object on the video screen and circle an area as large as we can on the object. The color settings for this object will be the ranges from minimum RGB values to maximum RGB values in this area. Fig. 2.2 shows an example of color settings. By color setting, we define the color ranges of different objects. They act as the threshold values for thresholding in the next step.



*Fig. 2.2 Color Settings*

### 2.2.2 Thresholding

Thresholding is a kind of pixel operation. It is a method to convert color images or gray scale images into binary images. Binary image can represent the shape of the object better.

When a new frame of image comes, we find the entry point of the image data. We then apply each pixel of data to the color setting ranges. If this pixel falls into the range of a specific object, we classify it this specific object.

After thresholding, the color images are converted into binary images.

### 2.2.3 Dilation and Erosion

Dilation and erosion are two kinds of neighborhood operations on binary images. They work on the form or the shape of objects. Dilation operator can be expressed as:

$$G'_{mn} = \bigvee_{k=-R}^R \bigvee_{l=-R}^R M_{k,l} \wedge G_{m-k,n-l} \underline{\Delta} G \oplus M \quad (2.1)$$

Where " $\bigvee$ " and " $\wedge$ " denotes the logical *or* and *and* operation, respectively. They correspond to the binary " $\Sigma$ " and " $\Pi$ " operations. " $\oplus$ " denotes the binary convolution operation. The subscript  $(k,l)$  of matrices  $M$  and  $G$  are referred to as the pixel of  $k$ th row and  $l$ th column. The binary image pixel in  $G$  is updated into  $G'$  by convolving with a symmetric  $2R+1 \times 2R+1$  mask  $M$ .

What does this operation achieve? Let us assume that all the coefficients of the mask are set to 'one'. If one or more object pixels, i.e., 'ones' are within the mask, the result of the operation will be one; otherwise it is zero. Hence, the object will be dilated. Small holes or cracks will be filled and the contour line will become smoother, as shown in Fig.2.3.

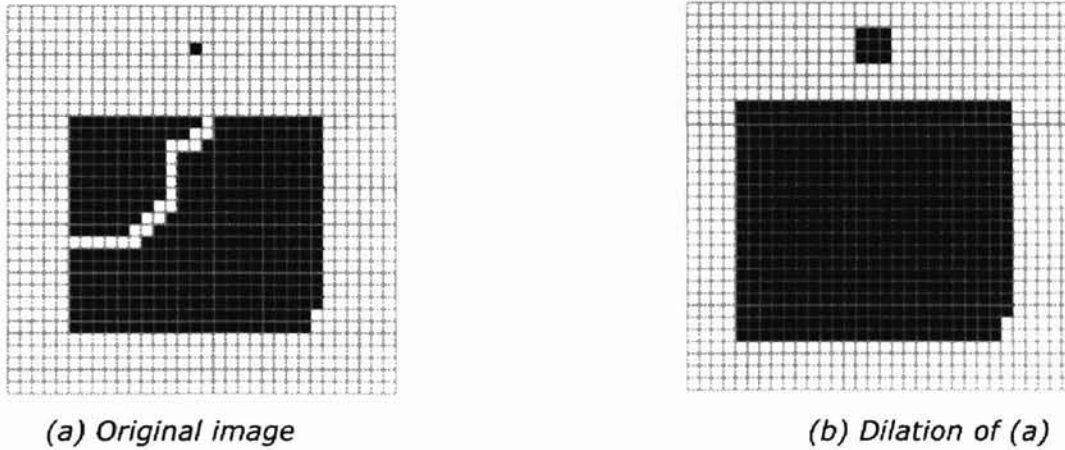


Fig. 2.3 Binary Dilation with a Mask of 3

Erosion has the opposite effect as dilation. In order to erode the object, we dilate the background by

$$G'_{mn} = \overline{\bigvee_{k=-R}^R \bigvee_{l=-R}^R M_{k,l} \wedge G_{m-k,n-l} \underline{\Delta} G \oplus M} \quad (2.2)$$

Where " $\oplus$ " denotes the dilation operation. Other symbols are the same as the dilation operation.

The result is only one if the mask is completely within the object. In this way, the object is eroded. Objects smaller than the mask will completely disappear, objects connected only by a small bridge will become disconnected, as shown in Fig. 2.4.

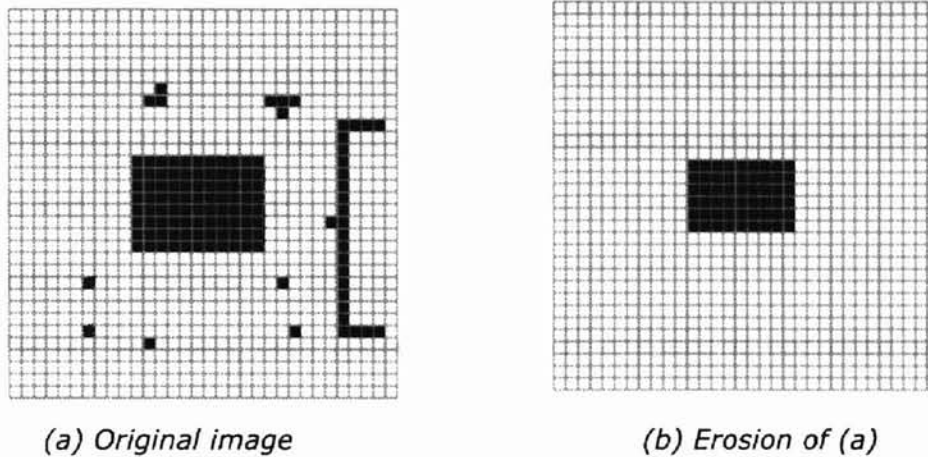


Fig. 2.4 Binary Erosion

The erosion operation is useful to filter out small objects. However it shows the disadvantage that all the remaining objects shrink in size. We can avoid this effect by dilation of the image after erosion with the same structure element. This combination of operations is called an *opening* operation.

$$G \circ M = (G \ominus M) \oplus M \quad (2.3)$$

where  $\circ$  denotes the *opening* operation.

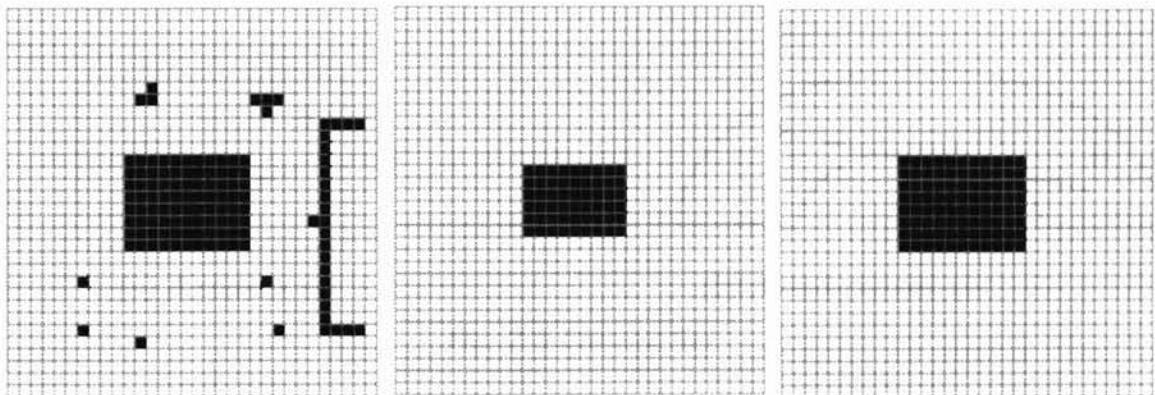


Fig. 2.5 Opening Operation

The opening sieves out objects that are smaller than the structure element, but avoids a general shrinking of the size. It is also an ideal operation to remove lines with a diameter that is smaller than the diameter of the structure element.

In contrast, dilation enlarges objects and closes small holes and cracks. General enlargement of the objects by the size of the structure element can be reversed by a following erosion. This combination of operation is called a *closing* operation.

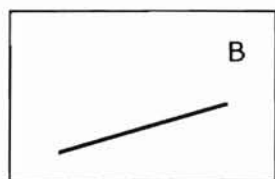
$$G \bullet M = (G \oplus M) \ominus M \quad (2.4)$$

Where  $\bullet$  denotes the *closing* operation

For our purposes, both *opening* operation and *closing* operation are useful. Closing operation is helpful for us to find the integral shape of objects. *opening* operation is helpful for us to remove the effect of white lines of the playground that are mis-detected by the computer.

If we use both in our intelligent robot system, we should use *closing* operation first in order to retrieve the integral shape of objects as good as possible. Then use the *opening* operation to remove the mis-detected white lines of the playground. However, it would increase the computational complexity requirement. At this moment, we only use the *closing* operation.

#### 2.2.4 Convex Processing



(a) A convex set



(b) A non-convex set

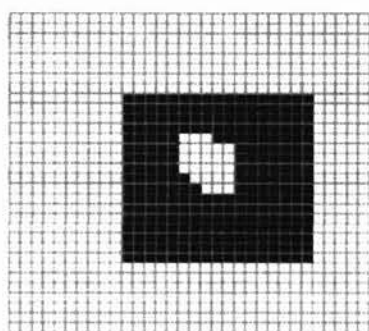
Fig. 2.6 Convex Set

In mathematics, if any two points in a set can be connected via a straight line that remains in the set, we call this set convex set. Fig. 2.5 (a) shows a convex set. Fig.

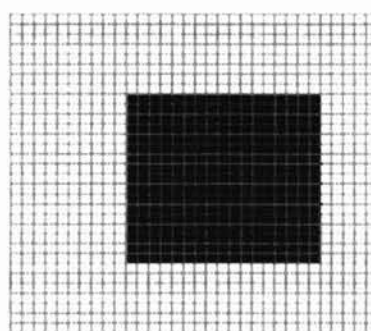
2.5(b) shows a non-convex set. In our intelligent robot system, the shapes of the robot team color, robot color, and ball, are all convex. We can use this character to improve our image processing effect.

In the dilation and erosion technique we discuss in the last subsection, if the holes or empty units in the binary images are too big, we couldn't fill in them even if we used dilation and erosion. But obviously, these holes in the center of the binary image belong to the object. By using this convex processing, we can fill in these holes.

In detail, for any detected component, we can find and record the four corners values of the binary image, i.e., upper left, upper right, lower left, and lower right values. We then label any pixels inside thesis 4 corners to this object. Fig. 2.7 shows the convex processing operation.



(a) Original image



(b) after convex processing

Fig. 2.7 Convex Processing

### 2.2.5 Component Labeling

A component is the smallest connected unit. In order for two points belong to a component, they have to be connected. If two points are not connected, they belong to different components. A component can be considered as an object in our image processing. In order to determine the size, shape of the object, we need to label the component a same mark. This is called component labeling. Component labeling is a very important process in computer vision. By labeling different component with

different marks, we can distinguish different objects of the same kind. Fig. 2.8 shows the component labeling operation. (This picture is copied from the Micro Adventure Company's on-line documents.)

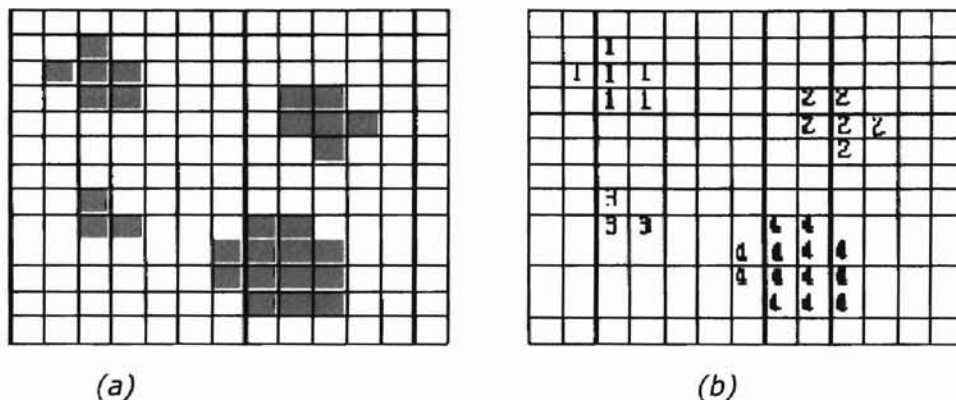


Fig. 2.8 Component Labeling

Basically, there are two commonly used component labeling algorithm: recursive algorithm and sequential algorithm.

#### Recursive component labeling algorithm

1. Scan the image to find an unlabeled unity valued pixel and assign it a new label L.
2. Recursively assign a label L to all its unity-valued neighbors.
3. Stop if there are no more unlabeled unity valued pixels or.
4. Go to step 1.

The pseudo code for the recursive algorithm is listed below:

```

Label(r,c)
Store(r,c,L);
If p[r][c-1] is 1 and unlabeled, label(r, c-1);
If p[r][c+1] is 1 and unlabeled, label(r, c+1);
If p[r-1][c] is 1 and unlabeled, label(r-1, c);
If p[r+1][c] is 1 and unlabeled, label(r+1, c);

```

Fig. 2.9 Pseudo Code for the Recursive Algorithm

The advantage of recursive algorithm is simple and effective. The disadvantage is it needs a large stack space since the function recursively calls itself. We need reserve and commit a large stack space when executing the algorithm.

#### Sequential component labeling algorithm

1. Scan the image from left to right and top to bottom.
2. If the pixel is unity valued, then
  - (a) If only one of its upper or left neighbors has a label, then copy the label.
  - (b) If both have the same label, then copy the same label.
  - (c) If both have different labels, then copy the upper pixel's label and enter the labels in an equivalence table as equivalent labels.
  - (d) Otherwise assign a new label to this pixel and enter this label in the equivalence table.
3. If there are more pixels to consider, then go to step 2.
4. Find the lowest label for each equivalent set in the equivalence table.
5. Scan the picture. Replace each label by the lowest label in its equivalent set.

The pseudo code for step 2 in the sequential algorithm is shown below.

```

K=1;
If p[r][c]=1 {
    if (p[r-1][c]=1 && p[r][c-1]=0)      label[r][c]=label[r-1][c];
    if (p[r-1][c]=0 && p[r][c-1]=1)      label[r][c]=label[r][c-1];
    if (p[r-1][c]=1 && p[r][c-1]=1)      label[r][c]=label[r][c-1];
    if (p[r-1][c]=0 && p[r][c-1]=0)    {
        label[r][c]=k;      k=k+1;}
}

```

*Fig. 2.10 Pseudo Code for the Sequential Algorithm*

#### 2.2.6 Size Filtering



Size filtering can effectively remove noise after component labeling. If objects of interest have sizes greater than  $T$ , all components below  $T$  are removed by changing the corresponding pixels' value to 0.

Both size filtering and erosion can remove noise. Erosion can remove some thin lines even if the sizes of the lines are greater than  $T$ , while size filtering can't. Size filtering can remove small objects that fill a whole mask of erosion, while erosion won't work in this case. Fig. 2.11 shows the size filtering operation. (This picture is also copied from Micro-Adventure Company's on-line documents.)

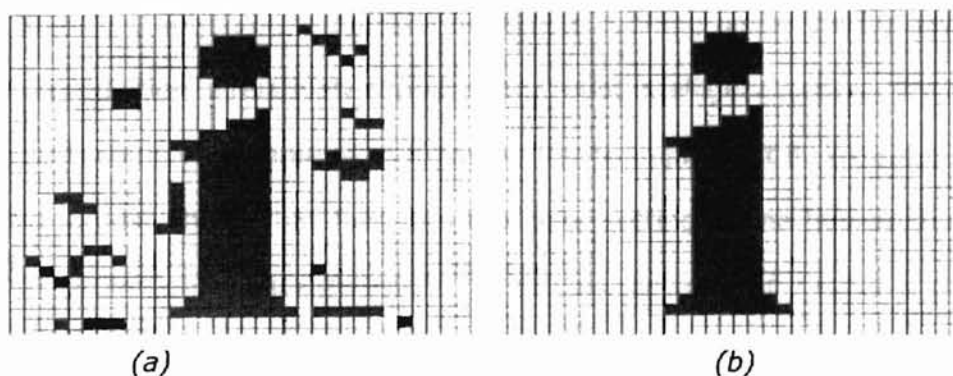


Fig. 2.11 Size Filtering

## 2.3 Pattern Recognition

### Object size and position

After all of the above steps, we finally come to decide where are the objects and what are they. To decide the location of an object, we need to find the size of the object first.

To decide the size of an object, we simply add all the pixels of the binary image together.

$$A = \sum_{i=1}^m \sum_{j=1}^n B[i, j] \quad (2.5)$$

Where  $A$  is the size of the object, and  $B[i, j]$  denotes the binary 1 or 0 of a pixel for the binary image. The position of the object is the center of the image area.

$$\bar{x} = \frac{\sum_{i=1}^n \sum_{j=1}^m jB[i, j]}{A}, \quad (2.6)$$

$$\bar{y} = \frac{\sum_{i=1}^n \sum_{j=1}^m iB[i, j]}{A}. \quad (2.7)$$

### Pattern recognition

Since there are only limited number of objects running under simple environment, pattern recognition is relatively simple in our robot intelligent system. We don't need to use such complex technique as Kalman filter to implement it. We simply take the detected largest object that falls into the color setting of the ball as the ball. The detected largest object that falls into of the color settings of a robot is the specific robot. Its team color is the closest team color to it. By the robot color position and team color position, we can detect the position of the robot  $[X_R, Y_R]$  and current facing direction  $[\theta]$  of the robot by

$$X_R = \frac{X_{RC} + X_{HC}}{2}, \quad (2.8)$$

$$Y_R = \frac{Y_{RC} + Y_{HC}}{2}, \quad (2.9)$$

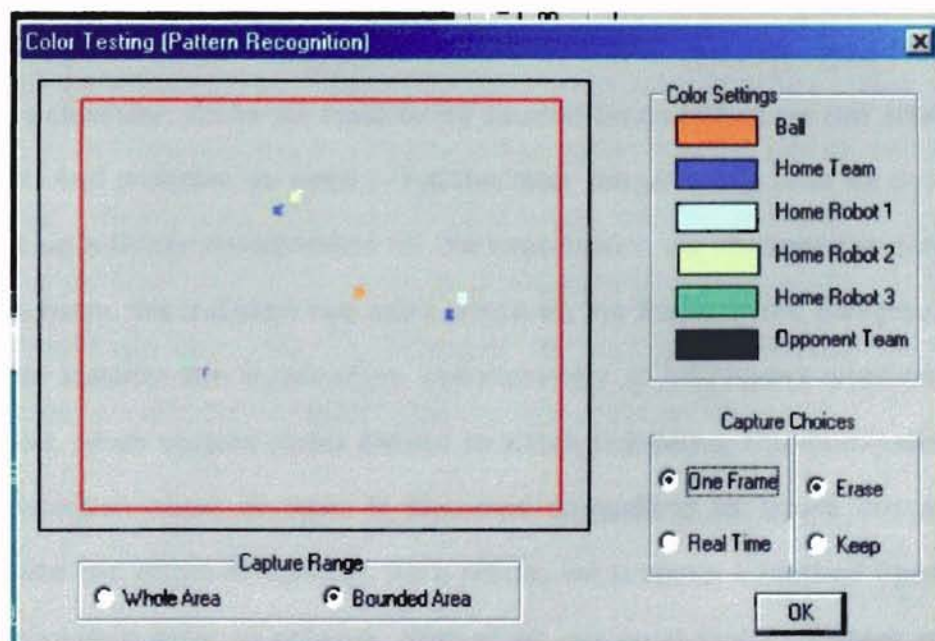
$$\theta = \tan^{-1}((Y_{HC} - Y_{RC}) / (X_{HC} - X_{RC})) - \pi / 4. \quad (2.10)$$

Where  $X_R$  is the X position of the robot,  $Y_R$  is the Y position of the robot,  $\theta$  is the angle of robot facing direction,  $X_{RC}$  is the X position of the robot color center,  $Y_{RC}$  is the Y position of the robot color center,  $X_{HC}$  is the X position of the robot home color center,  $Y_{HC}$  is the Y position of the robot home color center.

## 2. 4 Real Detection Effect



(a)



(b)

*Fig. 2.12 Real Detection Effect*

Shown in Fig. 2.12(a) is a copy of real video image. There are three robots on the playground. They all have the same team color. And each of them has a different

robot color to identify it. Located in the middle of the three robots is a red colored ball. Shown in Fig. 2.12(b) is the corresponding computer vision output. The right side column of the picture shows the color settings for different objects. The left side bigger square shows the whole field of video image. And the smaller square shows the bounded area that means we only detect and recognize the objects in this area. We can see from the picture that all objects of three robots and the ball are correctly identified. Their positions corresponds exactly the same positions as the video image. The detected objects are drawn as the same colors as their color settings. The lower parts of the picture 2.12(b) are some capture choices for doing experiment.

### ***Chapter 3 Lighting Problems***

After all these image processing techniques, the unstable illumination caused by uneven light shedding has to be resolved.

The effect of computer-detected objects is easily affected by the illumination (brightness). When the brightness changes, the output signal from the CCD camera will change. Correspondingly the RGB output values from the vision board will also vary. Therefore it affects the target detection process. The video image is significantly different if you turn on a lamp even far from the playground from leaving the lamp off. It can be quite different when a person is standing aside the playground from nobody is standing there.

In order to get a better detection performance, we need to adjust the lighting conditions carefully so that the brightness is evenly distributed at different areas of the playground, and make effort to keep it stable. We also need to do the color setting carefully. Often we need to try several times before we can achieve an ideal output. And probably we need to set the color ranges every time we do experiment. To set up a better environment for the experiment, we changed the screen of lights on the room. We installed two extra lamps on the frame of the playground. All these were to stabilize the illumination. Unfortunately all this hadn't ameliorate much. In addition, when visitors stand around to watch the demo, the illumination will affect the detection more or less. It becomes compelling to figure out a method to minimize the effect of lighting. As a result, we propose a method based on a least approximation error to solve it. First of all, we need first to discuss some lighting techniques and what is brightness, and how it affects the detection performance.

### 3.1 Lighting Techniques

There are two basic lighting techniques: front-lighting and back-lighting. Front-lighting is to put the lighting sources in front of the object. Back-lighting is to put the lighting sources behind the object. Front-lighting is also divided into omni-directional illumination and directional illumination. Omni-directional illumination provides a uniform, omni-directional illumination that can eliminate shadows on the objects within the scene. Directional illumination can highlight surface texture. This technique is often used for special purposes [1]. Back-lighting is often used to analyze the shape of the object. As a result, the omni-direction front-lighting technique is applied to the intelligent robot system.

### 3.2 Brightness

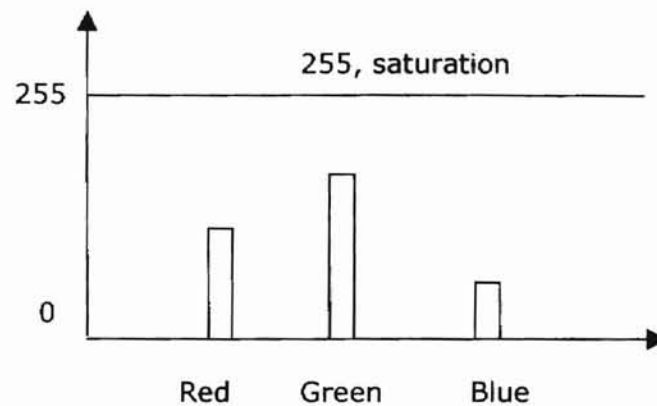
Changing the illumination causes changing the brightness of the image. Brightness is the pixel intensity value stored in the image array. It is like signals' DC component in electrical engineering. Decreasing brightness can be thought of as the simple subtraction of a constant from all pixel intensity values stored in the image array. The brightness can also be increased by adding a constant. In general, the brightness modification operation can be expressed as

$$P' = A + P \quad (3.1)$$

Where  $P'$  is the pixel value after enhancement,  $P$  is the pixel value before enhancement, and  $A$  is the enhancement factor (constant)

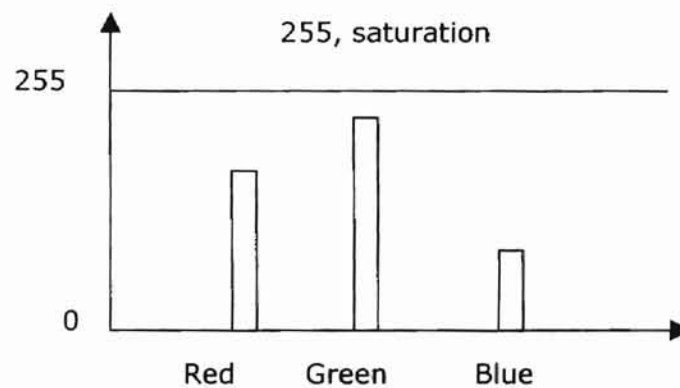
In our robot system, the image is represented using 24-bit Bitmap. Every 8 bits represents the intensities of red, green and blue. The red, green and blue intensity has a range of 0 to 255. 0 is the value when the illumination is totally vanished or the color intensity is not present in the image. 255 is the value when the brightness

is so strong that saturation occurs, or the color intensity prevails overwhelming in the image. Fig. 3.1 shows the brightness value of RGB for a pixel in the usual case.



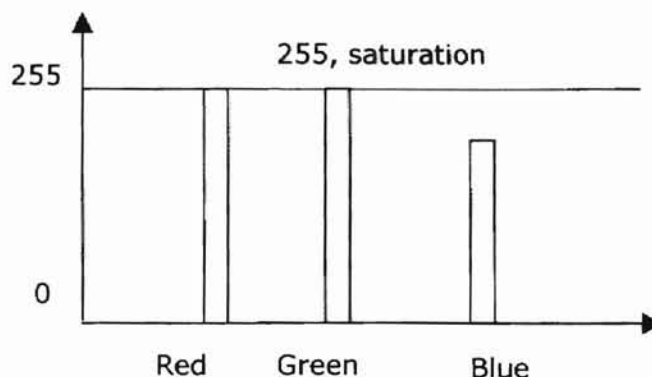
*Fig. 3.1 The RGB Intensities of a Pixel*

When the brightness increases, the RGB intensities all increase the same value (Fig. 3.2).



*Fig. 3.2 Increase the Brightness*

When the brightness increases more, saturations of red and green occur (Fig. 3.3). When the brightness increases even more, saturations of all red, green and blue could occur. This doesn't mean the object will change color when the illumination changes. It is because the CCD camera will always output 1 when the illumination is too strong. This actually causes over-lighting problem. In this case, the computer cannot detect the differences of different colored objects. A good illumination should avoid any saturation of red, green, and blue.



*Fig. 3.3 Over-lighting Causes Saturation of Red and Green Intensities*

### 3.3 Nearest Neighborhood Method to Solve the Lighting Problem

To some extent, the lighting unsteadiness problem is inevitable. If we put too much effort to build a more idea environment, it would be fortune costing and would not be practicable. So I thought about how to solve this problem by software or algorithm. Human and animals can identify an object even if the lighting experiences some appreciable change. On one side, they do this by adjusting the pupils of the eyes. On the other hand, human eyes identify an object by comparing it with their background. If the lighting of the objects changes, the lighting of the background also changes accordingly. For example, a red colored object is placed in a green environment. It is still red compared to its environment even if the illumination becomes stronger or weaker.

The essential problem in our system is, when the illumination changes, the brightness of the image changes. A pixel originally falls into the thresholding ranges may no longer falls into the thresholding ranges. This causes a mis-identify problem. Our approach to solve this problem is that, if the lighting condition changes, fox example, the illumination becomes weaker, the background lighting intensity also decrease. Although the object color no longer falls into the thresholding ranges, it is



still closer to this object's thresholding values of than the background and other objects. So it occurred to me that we should also mark a background dark color thresholding ranges and the background white color thresholding ranges. Instead of using the ranges of color settings of objects, we adopt the weighted average of a color setting range as the thresholding value. By doing this, a range such as 51 to 86 is represented as a value such as 70. We then calculate the distances of the RGB value of the pixel from each of these thresholding values. We add the RGB distances of each pixel together. And we classify the pixel to the object that has the minimum distance sum of all the objects (it includes background dark color and background white color of the playground.). A pseudo code to implement this is shown in Fig. 3.4.

```

MINI_SUM=INFINITE;
INDEX=0;
FOR I=1; I< NUMBER_OF_OBJECTS; I++
{
D_R=ABSOLUTE VALUE (RED - RED_RANGE[I]);
D_G=ABSOLUTE VALUE (GREEN - GREEN_RANGE[I]);
D_B=ABSOLUTE VALUE (BLUE - BLUE_RANGE[I]);
DIS=D_R+D_G+D_B;
IF DIS< MINI_SUM {
        MINI_SUM=DIS;    INDEX=I;
    }
}
OBJECT = INDEX;







```

*Fig. 3.4 Algorithm to Minimize Lighting Effect*

By applying this algorithm, the detection effects were significantly improved. Not only can we detect a more accurate shape of the object, but also, the effect of the light imbalance is minimized. It was found there was little if not at all whether a person standing aside the playground or not.

Later, I found that using squared sum of the color distance errors should be more reasonable, because square maximizes effect of bigger distance of a specific RGB value. We tried this. It did get better results.

Table 3.1 Nearest Neighborhood Method to Solve the Illumination Problem

Candidate Objects	Home Team Color	Robot 1 Color	Robot 2 Color	Background Dark Color	Background White Color
Object Color Settings					
RGB values	75 151 181	10 246 122	150 238 250	49 57 65	203 209 215
Candidate Pixel					
Pixel RGB Values	60 226 88				
Color Distances	15 75 93	50 20 34	90 12 162	11 169 23	143 17 127
Squared Sum of Color Distances	14499	4056	34488	29211	36867
Classified Object		√			

If we divide the sum of the squared distance errors by 3, we'll have the Nearest Neighborhood Method (NNM). It means we always select the object that has the least mean distance. In mathematics, it can be represented as:

$$C = \min_o \left( \frac{1}{3} \sum_{i=1}^3 (V(i) - V_o(i))^2 \right) \quad (3.1)$$

Where  $C$  is the classified object,  $o$  is the candidate objects,  $V$  is the intensity RGB value of a pixel,  $V(1)$  is the red intensity of the pixel,  $V(2)$  is the green intensity of

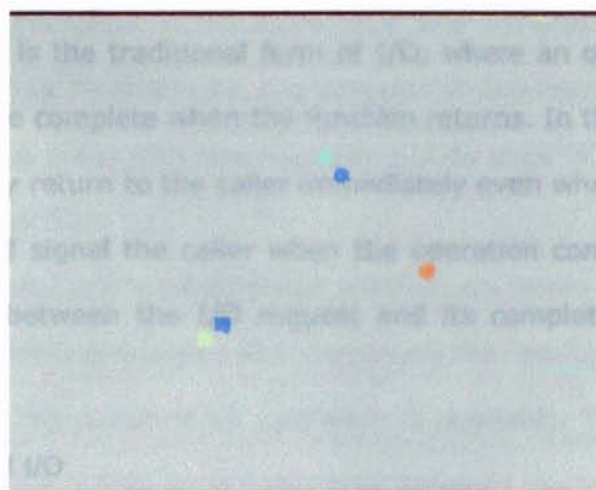
the pixel,  $V(3)$  corresponds to the blue intensity of the pixel, and  $V_T$  is the correspondingly thresholding values for one object. For any one pixel, we calculate the distances of its RGB from the thresholding values of an object, square it, and calculate the mean value. We then calculate the mean values for all candidate objects. The classified object  $C$  would be the one that has the minimum mean square value. Shown in Table 3.1 is an example of using NNM to solve the illumination problem.

Shown in Fig. 3.5 is a picture of the detected objects.

In comparison with Fig.2.9, the effect in this figure is much improved. The shapes detected are more accurate approximation of real objects. Furthermore, in Fig.2.9, the color settings were carefully designated and the lighting was adjusted to its best effect, it was the best performance we could get before we used the LMS method. If we moved the robot or the ball from one corner to another corner, we were not sure we still could get the same good result. Comparably, the effect of Fig.3.4 is the usual performance we can achieve after we used the fuzzyfication method. There is almost no more difference whether a person or several persons are standing besides the playground.



(a)



(b)

*Fig. 3.5 Real Image Detection Effect*

## **Chapter 4 Serial Communication and Distributed Computing**

### **4.1 Serial Communication**

In our system, the orders to the robot are sent to the wireless transmitter through the serial communication port of the computer. Serial communication in Win32 has significant difference from serial communication in Win16. Here we talk it briefly.

Reading from and writing to communications ports in Win32 is very similar to file input/output (I/O) in Win32. In fact, the functions that accomplish file I/O are the same functions used for serial I/O. I/O in Win32 can be accomplished in two ways: overlapped or non-overlapped [5].

*Non-overlapped I/O* is the traditional form of I/O, where an operation is requested and is assumed to be complete when the function returns. In the case of *overlapped I/O*, the system may return to the caller immediately even when an operation is not yet finished and will signal the caller when the operation completes. The program may use the time between the I/O request and its completion to perform some "background" work.

#### **4.1.1 Non-overlapped I/O**

Non-overlapped I/O is very straightforward, though it has limitations. An operation takes place while the calling thread is blocked. Once the operation is complete, the function returns and the thread can continue its work. This type of I/O is useful for multithreaded applications because while one thread is blocked on an I/O operation, other threads can still perform work. It is the responsibility of the application to serialize access to the port correctly. If one thread is blocked waiting for its I/O operation to complete, all other threads that subsequently call a communications API will be blocked until the original operation completes. For instance, if one thread

were waiting for a **ReadFile** function to return, any other thread that issued a **WriteFile** function would be blocked.

One of the many factors to consider when choosing between non-overlapped and overlapped operations is portability. Overlapped operation is not a good choice because most operating systems do not support it. Most operating systems support some form of multithreading, however, multithreaded nonoverlapped I/O may be the best choice for portability reason.

#### 4.1.2 Overlapped I/O

Overlapped I/O is not as straightforward as nonoverlapped I/O, but allows more flexibility and efficiency. A port open for overlapped operations allows multiple threads to do I/O operations *at the same time* and perform other work while the operations are pending. Furthermore, the behavior of overlapped operations allows a single thread to issue many different requests and do work in the background while the operations are pending.

In both single-threaded and multithreaded applications, some synchronization must take place between issuing requests and processing the results. One thread will have to be blocked until the result of an operation is available. The advantage is that overlapped I/O allows a thread to do some work between the time of the request and its completion. If no work *can* be done, then the only case for overlapped I/O is that it allows for better user responsiveness.

An overlapped I/O operation has two parts: the creation of the operation and the detection of its completion. Creating the operation entails setting up an **OVERLAPPED** structure, creating a manual-reset event for synchronization, and calling the appropriate function (**ReadFile** or **WriteFile**). The I/O operation may or may not be completed immediately. It is an error for an application to assume that a request for an overlapped operation always yields an overlapped operation. If an

operation is completed immediately, an application needs to be ready to continue processing normally. The second part of an overlapped operation is to detect its completion. Detecting completion of the operation involves waiting for the event handle, checking the overlapped result, and then handling the data. The reason that there is more work involved with an overlapped operation is that there are more points of failure. If a non-overlapped operation fails, the function just returns an error-return result. If an overlapped operation fails, it can fail in the creation of the operation or while the operation is pending. You may also have a time-out of the operation or a time-out waiting for the signal that the operation is complete.

In our robot system, we use overlapped I/O operation. The thread sends the data to the I/O port buffer, then returns immediately to process other tasks.

## 4.2 Distributed Computing

### 4.2.1 The Needs for Distributed Computing

As shown, robots intelligent control system is a real time control system. The stream video is captured and sent to the computer to do image processing, pattern recognition, algorithm evaluation and orders transmitting. Both the image processing and control algorithm evaluation are time consuming.

When we are doing experiments, we need to observe how the robots are behaving. We can't capture all the phenomena at one time when we are doing real time experiments. Besides the battery of the robot may deplete. The experiment is expensive. We can't repeat an experiment for many times easily. It is to our advantage to record the experiment (video) for later analysis. If we use the same computer to do the video recording work, it would definitely slow down the processing significantly.

Distributed computing is a good alternative to solve this question.

#### 4.2.2 Implementation Possibility

Windows 95 includes several mechanisms that support distributed computing. Typically, distributed computing means that a computing task is divided into two parts. The first part runs on the client computer and requires minimal resources. The other part of the process runs on the server and requires large amounts of data, number crunching, or specialized hardware.

Another type of distributed computing spreads the work among multiple computers. For example, one computer can work on a complex analytical problem that would take a month to solve. But with distributed computing, 50 computers could work on the same analytical problem simultaneously and solve it in less than a day.

In both cases, a connection between computers at a process-to-process level allows data to flow in both directions. Windows 95 includes the following inter-process communication (IPC) mechanisms to support distributed computing: Windows Sockets, Remote Procedure Calls (RPC), NetBIOS, named pipes, and mailslots. We'll only use Winsock for our purposes. For other mechanisms, please refer to Microsoft Online Documentation[6].

#### 4.2.3 Windows Sockets

Windows Sockets is a Windows implementation of the widely used U.C. Berkeley Sockets API, the *de facto* standard for accessing datagram and session services over TCP/IP [6]. Non-NetBIOS applications must be written to the Sockets interface to access Microsoft TCP/IP protocols. Applications written to the Sockets interface include FTP and SNMP. In Windows 95, sockets support is also extended to IPX/SPX.



Windows Sockets in Windows 95 is a protocol-independent networking API tailored for use by programmers using the Windows family of products. Windows Sockets is a public specification that aims to do the following:

1. Provide a familiar networking API to programmers using Windows or UNIX,
2. Offer binary compatibility between heterogeneous Windows-based TCP/IP stack and utility vendors, and
3. Support both connection-oriented and connectionless protocols

In TCP/IP, the internetworking address is the IP address of the workstation and the software process address is the port number. Source and destination IP address and port numbers are fields in the TCP/IP packet structure.

Class **CAsyncSocket** provides an object-oriented abstraction for programmers who want to use Windows Sockets in conjunction with MFC (Microsoft Fundamental Class). The main functions for using Winsock programming are listed below [22].

Table 4.1 Operations of the CAsyncSocket Class

<a href="#">Accept</a>	Accepts a connection on the socket.
<a href="#">AsyncSelect</a>	Requests event notification for the socket.
<a href="#">Bind</a>	Associates a local address with the socket.
<a href="#">Close</a>	Closes the socket.
<a href="#">Connect</a>	Establishes a connection to a peer socket.
<a href="#">IOCtl</a>	Controls the mode of the socket.
<a href="#">Listen</a>	Establishes a socket to listen for incoming connection requests.
<a href="#">Receive</a>	Receives data from the socket.
<a href="#">ReceiveFrom</a>	Receives a datagram and stores the source address.
<a href="#">Send</a>	Sends data to a connected socket.
<a href="#">SendTo</a>	Sends data to a specific destination.
<a href="#">ShutDown</a>	Disables <b>Send</b> and/or <b>Receive</b> calls on the socket.

Table 4.2 Overridable Notification Functions of the CAsyncSocket Class

<a href="#">OnAccept</a>	Notifies a listening socket that it can accept pending connection requests by calling <b>Accept</b> .
<a href="#">OnClose</a>	Notifies a socket that the socket connected to it has closed.
<a href="#">OnConnect</a>	Notifies a connecting socket that the connection attempt is complete, whether successfully or in error.
<a href="#">OnOutOfBandData</a>	Notifies a receiving socket that there is out-of-band data to be read on the socket, usually an urgent message.
<a href="#">OnReceive</a>	Notifies a listening socket that there is data to be retrieved by calling <b>Receive</b> .
<a href="#">OnSend</a>	Notifies a socket that it can send data by calling <b>Send</b> .

## **Chapter 5 System Evaluation and Research Perspectives**

By detecting the positions of objects, the computer provides a feedback to the robots. It can tell what is the current state of each robot, and the state of the ball. It also provides a way to monitor the performance of a robot behavior. This lays the basis for research.

### **5.1 Difficulties in Real Time Robot System Research**

However, there are some difficulties on conducting experiments with real time robot system.

1. Real time data collection & processing, real time control

The image processing and robots control are real time. If our program breaks down, or we just pause the program to debug some problems, since the robots may be still running, we may lose control of the robots. This may cause damage of robots.

2. Difficult to embed algorithms into the program

Since the program is coded using Visual C++, compared to Matlab, it is more difficult to embed algorithms in the program. Matlab provides numerous math functions and array operations etc. Unfortunately, we have to program the functions ourselves.

3. Difficult to do numerous and repetitive experiments

In order to do repetitive experiments, we may need to restore the states before another trial. In our system, we need to put the robot(s) back exactly the same position as last time or even the same facing direction. This is often exhausting and frustrating.

4. Hardware failure

Long time running and occasional collision with the borders may cause the robots to fail. To some extent, we can't afford to do experiment again and again in fear of robots' failure.

Robots consume DC power provided by a battery. A fully charged battery can only support a robot to run for 20 minutes or less. A robot would consume notable power even when it is standing by due to hardware design defect. This is another kind of hardware limitation.

## 5.2 Possible Researches

Despite these difficulties discussed above, we can deploy the system to do many interesting researches and experiments. We may have three robots playing soccer against three robots, and to research on cooperation among intelligent agents. We may train two robots to push a heavy box to a destination following a designated curve. This is another kind of cooperation research. We may have one robot playing soccer to research on robot learning.

Our program embedded all needed interfaces appropriate for these researches. All we need to do is adding modular algorithms to the existing applications.

## 5.3 An Introduction to the Windows Interface

Pasted in Fig. 5.1 is a snapshot of our Windows interface of the research platform.

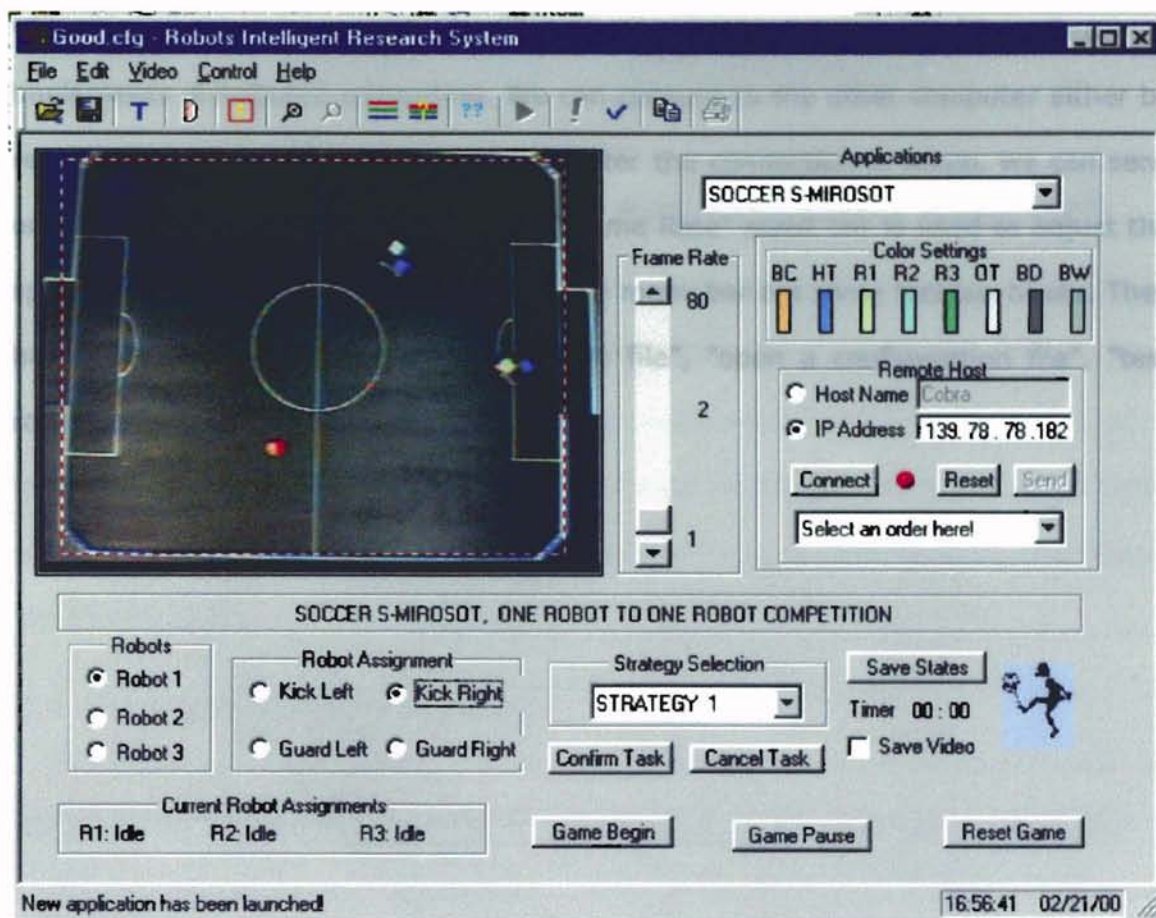


Fig. 5.1 The GUI Interface of Intelligent Robot System

The left upper part of the interface is the video camera window. It is used to observe the activities of robots and the ball on the spot. The right upper "Applications" combo box is used to select different applications based on this platform. It currently has MIROSOT, PUSHING BOX, and SMALL MIROSOT application items. When a different item is selected, a specific dialog will replace the dialog on the lower part of the interface. Currently shown is the SMALL MIROSOT application. Below the "Applications" combo box is the "Color Settings". The designated color settings are shown here. "BC" means the color of the ball. "HT" denotes the home team color. "R1", "R2", "R3" correspond to the colors of robot 1, robot 2, and robot 3 respectively. "OT" means the opponent's team color. "BD" indicates the background dark color of the playground. "BW" represents the background white color, i.e. the

white lines in the playground. The "Remote Host" square below the "Color Settings" implements distributed computing. We can connect to the other computer either by referring its name or by its IP address. After the connection is setup, we can send orders to control the other computer. "Frame Rate" scroll bar is used to adjust the speed of video capturing. Docked below the menu bar are some tool bar boxes. They have the functions of "save configuration file", "open a configuration file", "test robot", "set color", "test color", etc.

## **Part 2. Literature Reviews and Proposed Research**



## **Chapter 6 Knowledge Representation**

There are different kinds of knowledge representation techniques: rule-based representation, frame-based representation, multiple context representation, model-based representation, and blackboard representation [13]. Rule-based representation and frame-based representation are two most widely used techniques. We also used rule-based representation and frame-based representation in our intelligent robot system for playing soccer. Next we'll talk briefly about these two techniques. Please refer to related books for other knowledge representation techniques [13].

### **6.1 Rule-Based Representation**

Rules are conceptually represented as IF/THEN statements with the logical form:

IF <predicate> THEN <consequent>

Using such statements, knowledge engineers formulate the knowledge they obtain from the experts into sets of such rules. The inference engine then analyzes and processes these IF/THEN rules in one of two ways: backward or forward. In backward-chaining the inference engine works backward from hypothesized consequence to locate known predictions that would provide support. In forward-chaining, the inference engine works toward from known predicates to derive as many consequents as possible [13].

A simple example:

Consider a simple example of a set of rules that relates the type of day and where one would be.

Rule 1:       IF     IT IS A SUNNY DAY  
              AND I DON'T HAVE CLASS

```

                THEN I PLAY TENNIS
Rule 2:        IF    I PLAY TENNIS
                THEN I WOULD BE IN THE TENNIS COURT    **
Rule 3:        IF    IF IS A SUNNY DAY
                AND  I HAVE CLASS
                THEN I WOULD BE IN CLASS              **
Rule 4:        IF    IT IS A RAIN DAY
                AND  I HAVE CLASS
                THEN I WOULD BE IN CLASS              **
Rule 5:        IF    IT IS A RAIN DAY
                AND  I DON'T HAVE CLASS
                THEN I WOULD BE IN THE LIBRARY        **

```

The star marked lines are the consequents. A backward chaining inference engine might operate like this: first select a possible hypothesis, I WOULD BE IN THE LIBRARY, then check whether its predicates are correct. If the predicates are evaluated to TRUE, then this consequent is as the consequent of the rules. For forward-chaining inference engine, no hypotheses are provided because the rules are not used to try to derive the truth of any particular consequent. Rather, they are used to derive all possible consequents that can be derived from a set of predicates (actually from a set of values that cause one or more predicates to evaluate to TRUE).

One thing we need to do when using rule-based knowledge representation is pattern matching. Inference engines use a pattern-matching algorithm that enables them to match a new data value with only those rules that reference that data value. Thus, predictions need be tested only for those rules that might be affected by the new value.

## 6.2 Frame-Based Representation

Frame-based representation provides a mechanism for structuring certain types of knowledge in a knowledge base. The types of knowledge that can suitably be structured using a frame-based organization can range from collections of related facts, to relationships between such collections, to rule-based and even procedural representations of knowledge.

A frame can be viewed as a collection of related information about a topic. This information may be factual or procedural (e.g. data or functions). A frame may be taken to represent a class of similar objects; other frames, representing subclasses or specific instances of those objects, can be formed from the initial class frame. Rule-based and procedural knowledge representation can operate efficiently on frame-based representation.

The information or fields in the data structure of frame is called attributes, which, together with their values, form a description of an object. For example, a frame for a person might include the following attributes:

PERSONAL

Name

Age

Height

Weight

...

## 6.3 Object Oriented Programming

Object-oriented programming is a programming paradigm that uses frames and inheritance to build programs that model the user's perception of the world more

closed than other types of programs and consequently are more easily adaptable to small changes in the world. Developing an object-oriented program begins with a domain analysis that identifies the objects to be modeled and their behavior.

In the object-oriented program the objects are modeled by data structures with associated procedures. Some of the characteristics typically exhibited by object-oriented programs are:

- . Inheritance – Objects are organized into classes that exhibit similar behavior.
- . Encapsulation – Object A communicating with Object B. A is said to be a client of B, may request services provided by B without knowing how B will perform those services.
- . Polymorphism – The same message may be interpreted differently by different objects.

We used object-oriented programming Visual C++ to program our intelligent robot system.

## Chapter 7 An Introduction to Reinforcement Learning

### 7.1 What is Reinforcement Learning

Reinforcement learning refers to a family of algorithms inspired by human and animal learning[23]. It is about learning what to do, how to make decisions. Its objective is to discover a policy, i.e. a mapping from states of the environment to available actions, so as to maximize the average reward per step.

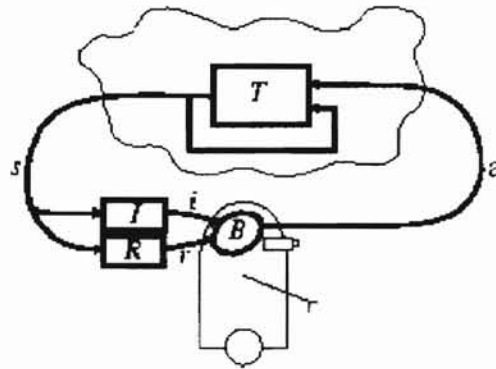


Fig. 7.1 The Standard Reinforcement-learning Model.

In the standard reinforcement learning model, an agent is connected to its environment via perception and action, as depicted in the Fig.7.1. On each step of interaction the agent receives an input,  $i$ , some indication of the current state,  $s$ , of the environment; the agent then choose an action,  $a$ , to generate as output. The action changes the state of the environment, and the value of this state transition is communicated to the agent through a scalar reinforcement signal,  $r$ . The agent's behavior,  $B$ , should choose actions that tend to increase the long-run sum of values of the reinforcement signal. It can learn to do this over time by trial and error, guided by a wide variety of algorithms.

### 7.2 Q-Learning

Q-Learning was proposed by Watkins in 1989 [2] . It is an importance breakthrough in reinforcement learning. It is also the most widely used algorithm in reinforcement learning. Its simplest form, one step Q-learning is defined as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (7.1)$$

Where  $t$  is the discrete time step,  $s_t$  is state at  $t$ ,  $a_t$  is action at  $t$ ,  $\alpha$  is step-size parameter,  $\gamma$  is the discount-rate parameter,  $r_{t+1}$  is reward at time step  $t+1$ ,  $Q(s_t, a_t)$  is the value of taking  $a$  in state  $s$ .

```

Initialize  $Q(s, a)$  arbitrarily
Repeat (for each episode):
    Initialize  $s$ 
    Repeat (for each step of episode):
        Choose  $a$  from  $s$  using policy derived from  $Q$  (e.g. greedy)
        Take action  $a$ , observe  $r, s'$ 
         $s \leftarrow s'$ 
    Until  $s$  is terminal
  
```

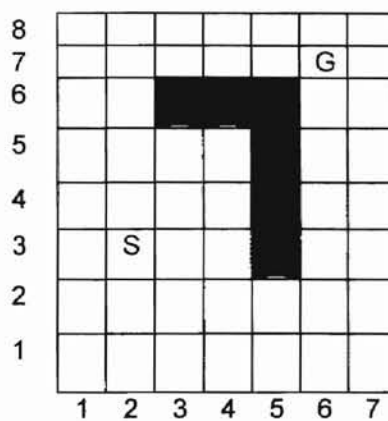
*Fig. 7.2 Q-learning Algorithm*

The process of Q-learning is shown in Fig. 7.2.

Next we will give out an example to show how Q-learning works.

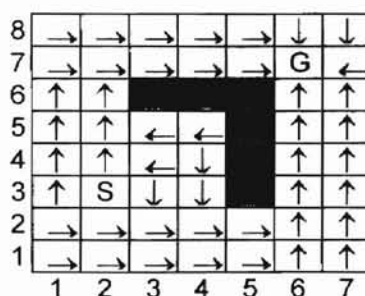
A "grid world" is often used to illustrate reinforcement learning (Fig.7.3). Imagine a robot initially in cell (2,3). The robot receives input vector (x1, x2) telling it what cell it is in; it is capable of four actions, n, e, s, w moving the robot one cell up, right, down, or left, respectively. It is rewarded one negative unit whenever it bumps into the wall or into the block cells. For example, if the input to the robot is (1,3), and the robot chooses action w, the next input to the robot is still (1,3) and it receives a reward of -1. If the robot lands in the cell marked G (for goal), it receives a reward

of +10. Let's suppose that whenever the robot lands in the goal cell G and gets its reward, it finishes an episode. And it is immediately transported out to some random cell, and the quest for reward continues.



*Fig. 7.3 A Grid World*

A policy for our robot is a specification of what action to take for every one of its inputs, that is, for every one of the cells in the grid. For example, a component of such a policy would be "when in cell (3,1), move right." An optimal policy is a policy that maximizes long-term reward. One way of displaying a policy for our grid-world robot is by an arrow in each cell indicating the direction the robot should move when in that cell. An optimal policy in the grid world is shown below:



*Fig. 7.4 An Optimal Policy in the Grid World*

The Q-learning procedure requires that we maintain a table of  $Q(x,a)$  values for all state-action pairs. In the grid world that we described earlier, such a table would not be excessively large. We might start with random entries in the table.

A portion of such an initial table might be as follows:

Table. 7.1 Value Table of Q Learning

$X$	$a$	$Q(x,a)$	$r(x,a)$
(2,3)	w	7	0
(2,3)	n	4	0
(2,3)	e	3	0
(2,3)	s	6	0
(1,3)	w	4	-1
(1,3)	n	5	0
(1,3)	e	2	0
(1,3)	s	4	0

Suppose the robot is in cell (2,3). The maximum Q values occurs for  $a=w$ , so the robot moves west to cell (1,3) – receiving no immediate reward. The maximum Q value in cell (1,3) is 5, and the learning mechanism attempts to make the value of  $Q((2,3),w)$  closer to the discounted value of 5 plus the immediate reward (which was 0 in this case). With a learning rate parameter  $\alpha=0.5$  and  $\gamma=0.9$ , the Q value of  $Q((2,3),w)$  is adjusted from 7 to 5.75. No other changes are made to the table at this time step.

The learning problem faced by the agent is to associate specific actions with specific input patterns. Q learning gradually reinforces those actions that contribute to positive rewards by increasing the associated Q values. Typically, as in this example, rewards occur somewhat after the actions that lead to them- hence the phrase delayed-reinforcement learning. One can imagine that better and better approximations to the optimal Q values gradually propagate back from states producing rewards towards all of the other states that the agent frequently visits. With random Q values to begin, the agent's actions amount to a random walk through its space of states. Only when this random walk happens to stumble into rewarding states does Q learning begin to produce Q values that are useful, and even then, the Q values have to work their way outward from these rewarding



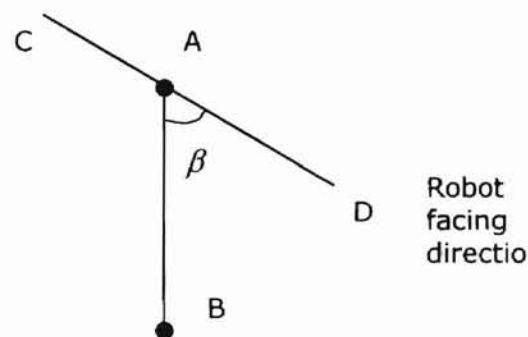
states. The general problem of associating rewards with state-action pairs is called the temporal credit assignment problem - how should credit for a reward be apportioned to the actions leading up to it? Q learning is, to date, the most successful technique for temporal credit assignment.

## **Part 3. Experimental Results**

## Chapter 8 Fuzzy Behavior Controller

### 8.1 Task Description

Moving the robot to some designated position is the most commonly expected behavior in intelligent robot research system. In order to shoot a ball, the robot needs to move to right behind the ball before it can shoot. In order to guard, it also needs to move the position to block the ball. It is also one of the most important behaviors. We need the robot to stop exactly the designated position, and also the robot needs to move at a possible high speed. The performance of the relocation affects heavily the performance of the next behavior.



*Fig. 8.1 Diagram of Robot Moving from A to B*

Fig. 8.1 shows a diagram of the robot moving. Suppose the robot is currently at position A, and it hopes to move to position B. Line CD is robot's current facing direction. Since the robot can move either forward or backward. We can always find the angle of robot facing direction line CD and the destination direction AB between 0-90. The robot needs to turn to the right angle while moving.

Usually, when the distance between the robot and its destination is farther, the robot's moving speed should be higher. When the distance becomes shorter, the speed

should slows down so that the robot can stop exactly at the destination. And when the angle difference is bigger, the turning speed is bigger.

The task is to figure out how to decide the robot forward moving speed and the turning speed so that it can reach the destination fast and precisely.

## 8.2 Why Fuzzy Logic System is Needed

It is difficult to find a formula for the real robot system. Actually we were using a formula shown below before we switched to fuzzy system:

$$\textit{forwarding} = \textit{dis} / 4, \quad (8.1)$$

$$\textit{turning} = \textit{angle} / 9, \text{ and} \quad (8.2)$$

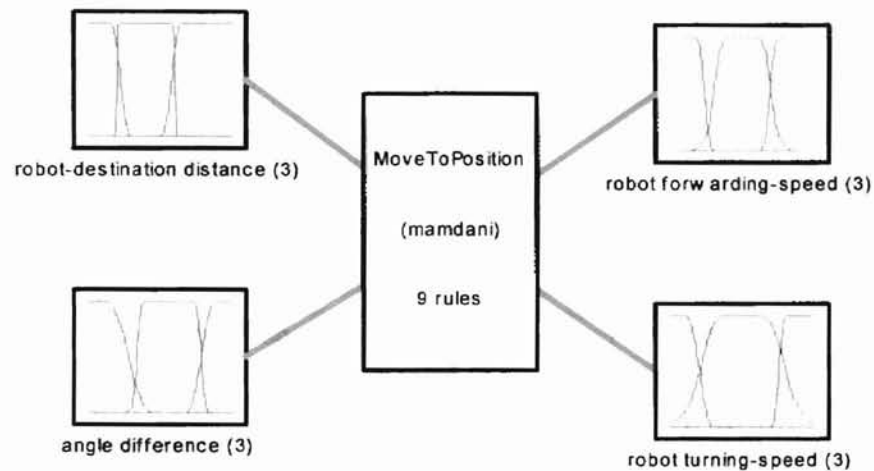
$$\textit{speed} = \textit{forwarding} + \textit{turning} \quad (8.3)$$

Where *dis* is the distance between the robot and its destination, *angle* is the angle difference of robot facing direction and destination direction, *speed* is the real output speed of the robot wheel. *speed* consists of forwarding speed *forwarding* which is *dis* / 4 and turning speed *turning* which is *angle* / 9. From the formula, we can see that, the forwarding speed is proportional to *dis* and the turning speed is proportional to *angle*.

This formula does work in some range of *dis* and *angle*. An important problem is that when *dis* becomes very big, the forwarding speed will also become very big and remain constant correspondingly. The robot will probably thrust forward a distance before it can turn to the right angle. The resulting trajectory should be very winding. Effort has been spent trying to find a better formula. For example, I tried to use the squared root of the *distance*. But all these didn't help much. We can't find a formula to represent the whole complex process.

Due to these reasons, fuzzy logic is a good alternative.

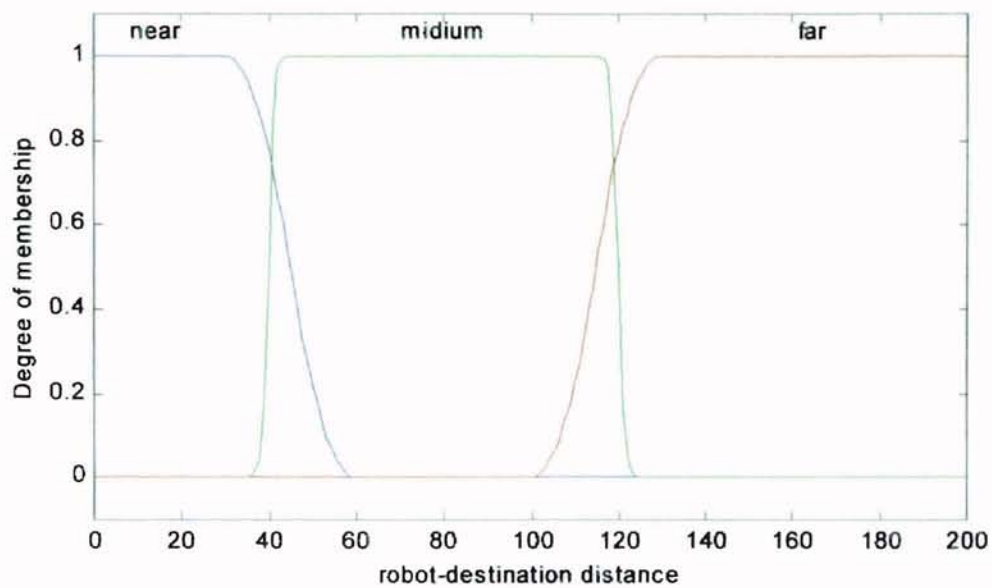
### 8.3 Fuzzy Behavior Controller



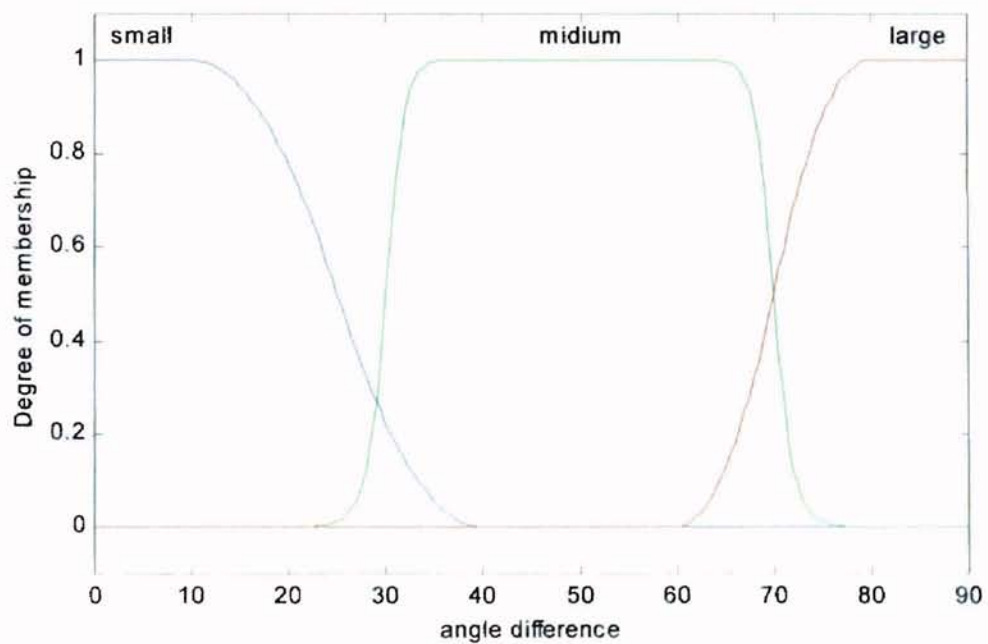
System MoveToPosition: 2 inputs, 2 outputs, 9 rules

*Fig. 8.2 The Input-output Diagram*

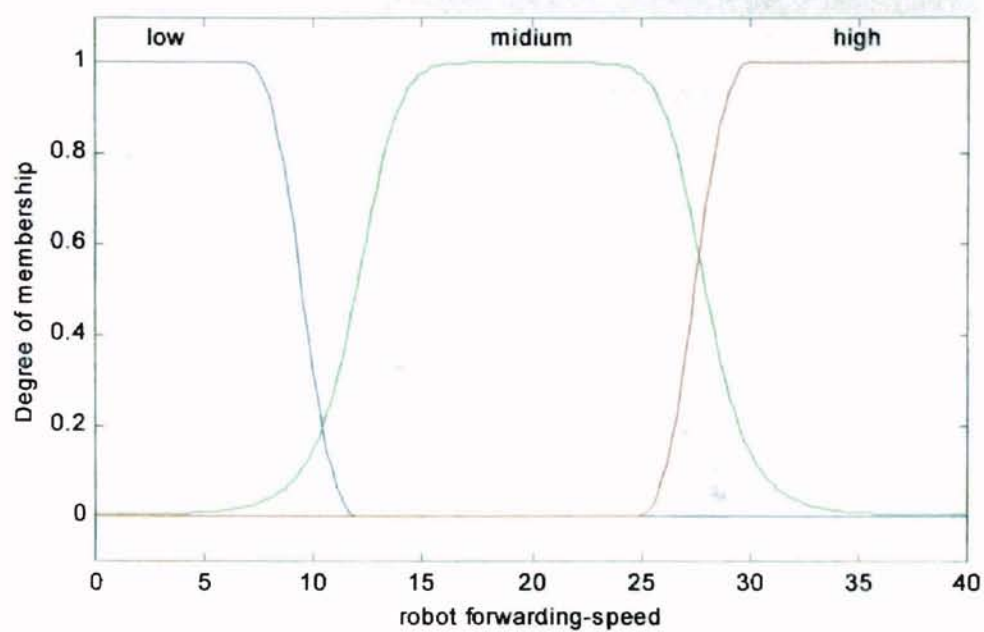
The input-output diagram is shown in Fig 8.2. In our fuzzy system, we have 2 inputs, 2 outputs and 9 rules. The 2 inputs are: *distance* which is the distance between the robot current position and destination and *angle* which is the angle difference between the robot facing direction and destination direction. The 2 outputs are *forwarding* which is robot forward moving speed and *turning* which is the robot turning speed. Each of these inputs and outputs has 3 membership functions. The membership functions of the 4 variables are heuristically shown in Fig. 8.3 to Fig. 8.6 respectively.



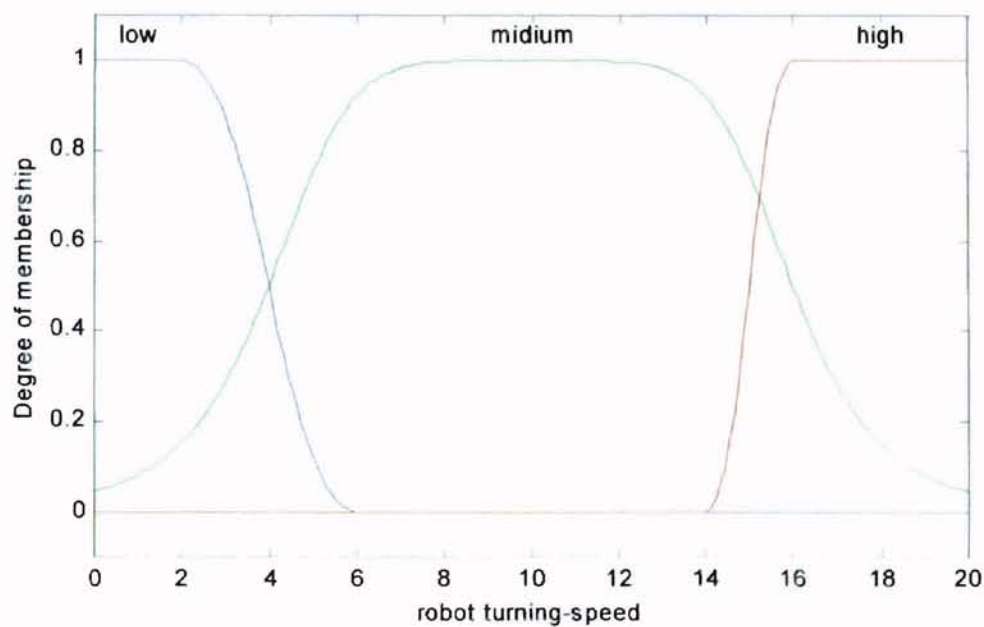
*Fig. 8.3 Membership Functions for Input Variable Distance*



*Fig. 8.4 Membership Functions for the Input Variable Angle*



*Fig. 8.5 Membership Functions for the Output Variable Speed*



*Fig. 8.6 Membership Functions for the Output Variable Turning Speed*

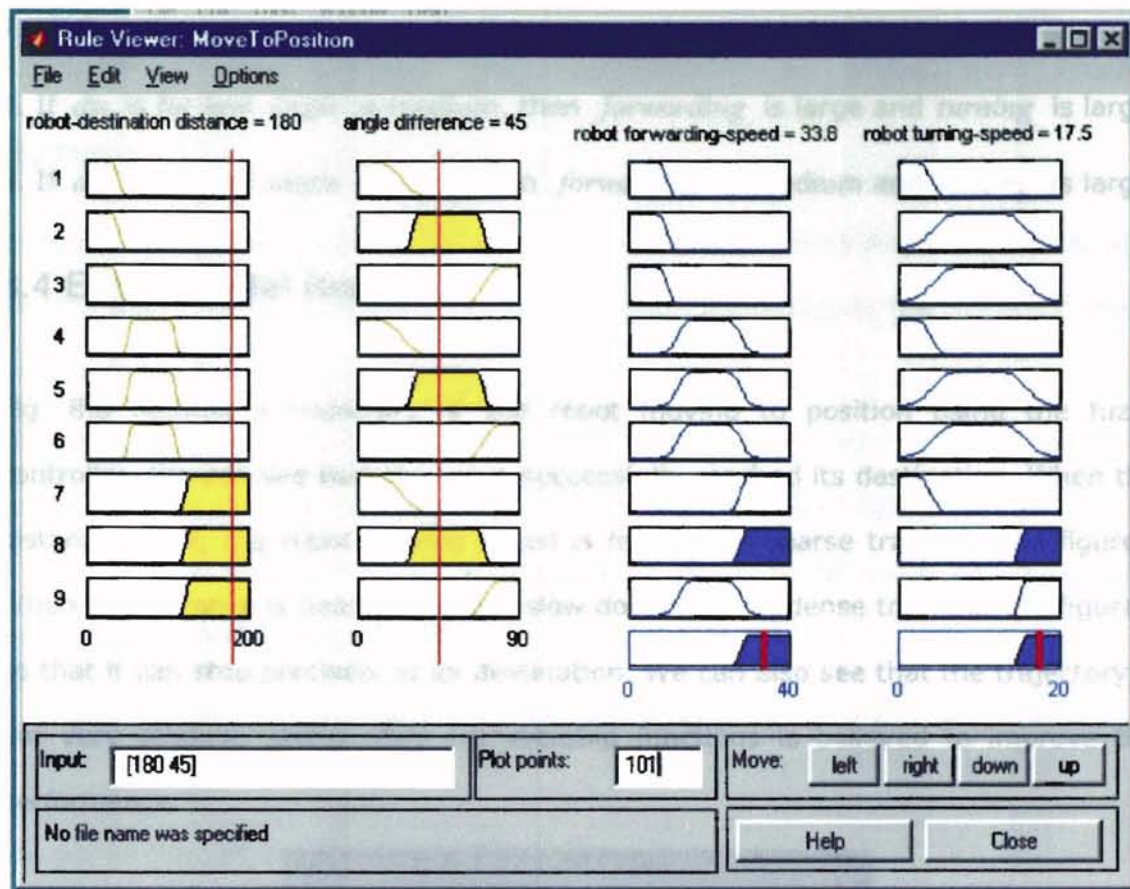


Fig. 8.7 Rules of the Fuzzy System

Fig. 8.7 displays all the rules for the fuzzy system. The 9 rules are:

1. If *dis* is near and *angle* is small, then *forwarding* is low and *turning* is low.
2. If *dis* is near and *angle* is medium, then *forwarding* is low and *turning* is medium.
3. If *dis* is near and *angle* is large, then *forwarding* is low and *turning* is medium.
4. If *dis* is medium and *angle* is small, then *forwarding* is medium and *turning* is low.
5. If *dis* is medium and *angle* is medium, then *forwarding* is medium and *turning* is medium.
6. If *dis* is medium and *angle* is large, then *forwarding* is medium and *turning* is medium.



7. If *dis* is far and *angle* is small, then *forwarding* is large and *turning* is small.
8. If *dis* is far and *angle* is medium, then *forwarding* is large and *turning* is large.
9. If *dis* is far and *angle* is large, then *forwarding* is medium and *turning* is large.

## 8.4 Experimental Result

Fig. 8.8 records a trajectory of the robot moving to position using the fuzzy controller. We can see that the robot successfully reached its destination. When the distance is far, the robot moving speed is fast (more sparse traces in the figure). When the distance is near, the speed slow down (mover dense traces in the figure), so that it can stop precisely at its destination. We can also see that the trajectory is not very straight. Using more membership functions is believed to improve the performance.

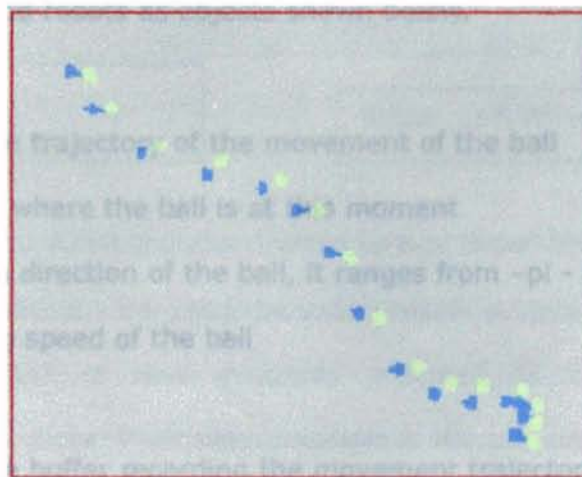


Fig. 8.8 A Trajectory Record of Robot Moving to Position

## **Chapter 9 Knowledge Representation in Intelligent Robot System**

Robots playing soccer games are kind of transferring human knowledge to robots. How to select means of knowledge representation are critical to the game playing in case of flexibility and expandability.

In the robot soccer game playing, we mainly used frame based knowledge representation for the attributes of robots and the ball and rule based knowledge representation for controlling the robots. It is programmed using object-oriented programming method Visual C++.

### **9.1 Frame Knowledge Representation in Robots Playing Soccer Game**

We define the ball and robots as objects shown below.

BALL

old positions[]; // the trajectory of the movement of the ball

current positions; // where the ball is at this moment

angle; // the moving direction of the ball, it ranges from  $-\pi$  -  $\pi$

speed; // the moving speed of the ball

ROBOT

old positons[]; // The buffer recording the movement trajectory of the robot

current position; // The current location of the robot

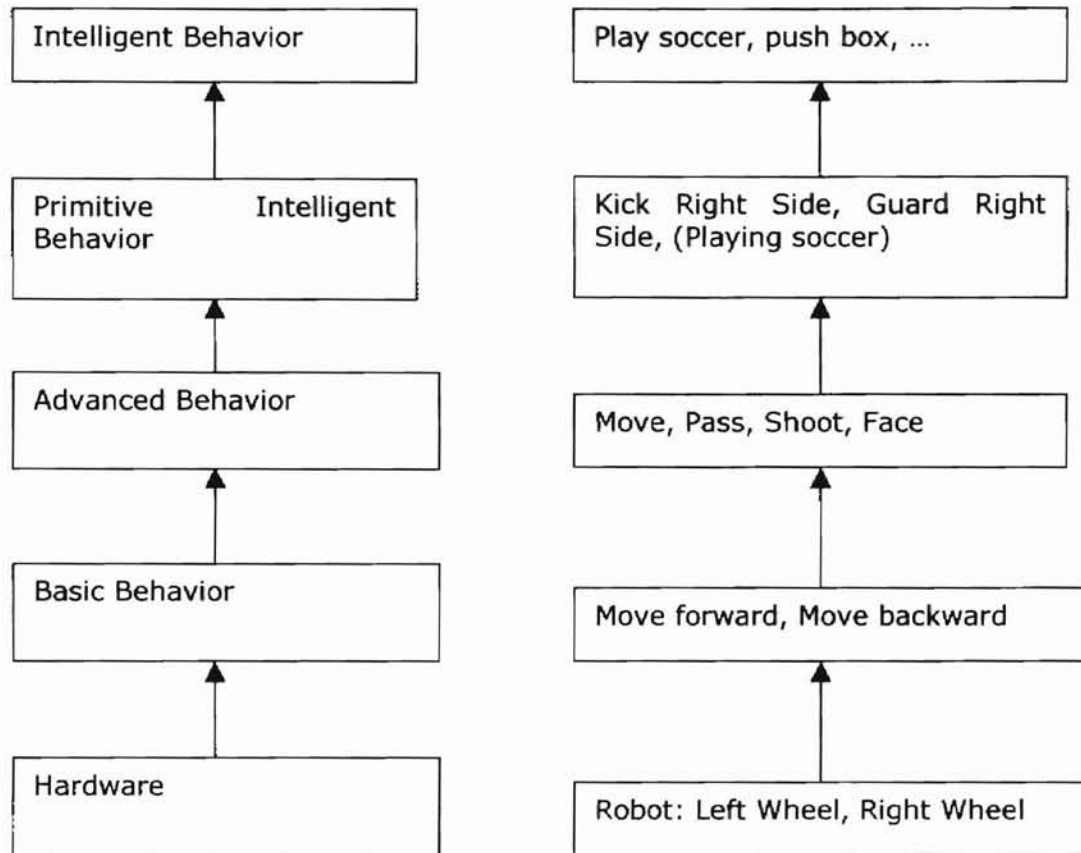
angle; // The current facing direction of the robot

destination positon; // The destination where the robot is moving to

command; // The current order the robot needs to execute.

The game playing is kind of detecting these variables and controlling them using the control algorithms.

## 9.2 Behavior-Based System



*Fig. 9.1 A Hierarchical Architecture of Robot Behaviors*

In robots playing soccer, we used behavior-based system. The architecture of behavior-based system is now generally accepted as an efficient basis for autonomous mobile robots. Their main principle is the achievement of desired goals by activating an appropriate sequence of behaviors. As in the OSI 7 layers in telecommunication, we also divide the robot behavior into a hierarchical architecture. Each higher layer behavior is implemented by a sequence of lower level behaviors. The highest layer is intelligent layer, which may be only an assignment to some robot(s) to do something. Human should only give robots this orders in the future if robots are in practical use. The lowest layer is hardware layer or physical layer.

### 9.3 Finite State Machine Implementation

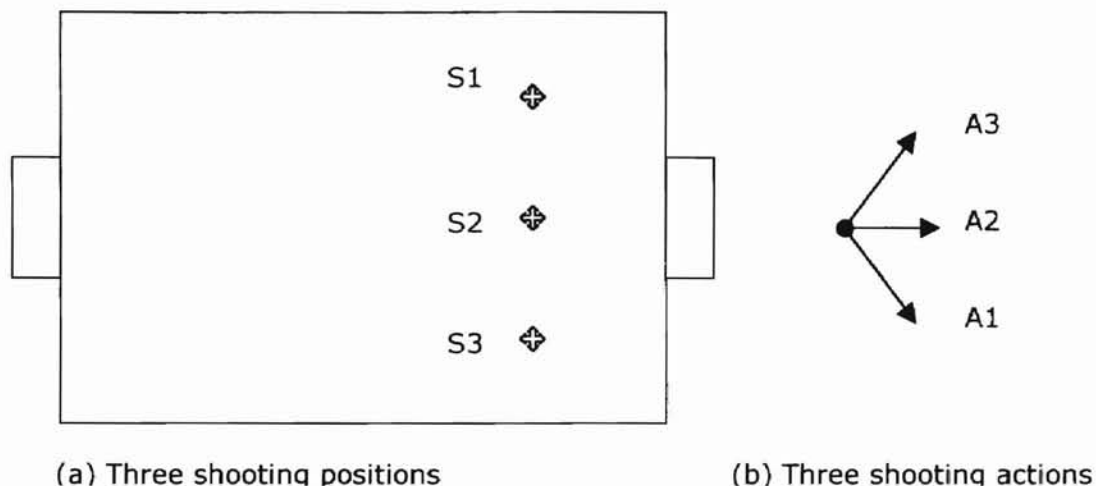
The robot control program runs on a PC and is realized as a finite state machine for the lower layers of the hierarchy.

At the beginning of the development, and the learning proces, there maybe only a few states. The learning proces involves adding more states to the finite state machine. More states means more complete consideration and more precise control. The deciding of the current state and the state transition is affected by the "situation". These decisive factors of the "situation" are:

- . Perceivable objects in the environment of the robot and their suspected or recognized states. In our system, it may be where is the ball, where and how fast it is moving.
- . The static characteristics of the environment (e.g., as stored in a map), even if they cannot be perceived by the robot's sensors at the given moment. In our robot system, it may be the location of the goal.
- . The state of the robot.
- . The repertoire of available behaviors and the abilities of the robot to change the present situation in a desired way by executing appropriate behaviors.
- . The goal of the robot, i.e. permanent goals (kicking right side, guarding right side) and transient goals emerging from the actual mission description (destination, corridor to be used, ... ) or directly imposed by the human operator. By using these mechanisms, we could successfully achieve the game playing.

## Chapter 10 Robot Learning in Intelligent Robot System

A simple learning of robot is implemented.



*Fig. 10.1 State Set and Action Set of Robot Learning to Shoot*

We train the robot to learn shooting the right side goal. Fig. 11.1 shows the robot learns to shoot. We can put the ball on three different locations, S1, S2, and S3 (Fig. 11.1 (a)). Each time the robot is put right behind the ball facing right side. So there are altogether 3 different states. In each state, there are three shooting actions to choose from for the robot, A1, A2, and A3 (Fig. 11.1 (b)). Shooting action 1 means the robot's left wheel runs at a higher speed than the right wheel, so that the ball will be kicked to right forward direction. Shooting action 3 means the robot's right wheel runs at a higher speed than the left wheel, so that the ball will be kicked to the left forward direction. Shooting action 2 means both the right and left wheels move at the same speed, so that the ball will be kicked to ahead. The task for the robot is to learn which shooting action to use under each different shooting position. To human, this is trivial. We know exactly how to shoot in each case. What we want to do is to teach the robot learn by itself.

After each shooting, we observe the next state of the ball. If the ball moves closer to the middle point of the goal, we think it enters a state which have a better value. If farer, we think it enters a worse state. If it scores a goal, we give it a reward of 10, otherwise 0. Since we are doing real robot experiments, sometimes even if the robot uses the right action, it is possible to miss the goal. In this case, we can get a table looking like below:

Table 10.1 Robot Learning

State	Actions	Value	Selected Action	Value of New Entered State	Reward	New Value
S1	A1	11				11
	A2	12				12
	A3	13	*	13	0	12.35
S2	A1	11				11
	A2	12				12
	A3	13	*	13	0	12.35
S3	A1	11				11
	A2	12				12
	A3	13	*	15	0	13.25

There are 3 different states S1, S2 and S3 in the table. Under each state, there are 3 different actions A1, A2 and A3. For each action-state pair, we have a value. We can initialize the values of states randomly. In this case, we initialize all three states of the values to 11, 12, and 13 respectively. We evenly distribute the learning for the 3 states and we use greedy policy in the learning. So if the robot is in state 1 in the first episode, since action 3 has the largest value 13, it will adopt action 3. Obviously the ball will be kicked away from the goal. The robot enters a worse state. We assume the value of new state just entered is equal to the value of this state. Using

$\alpha = 0.5$  and  $\gamma = 0.9$  in equation (8.1), we got a new value of 12.35. The robot is punished. Under state 2, the robot will also select action 3. Its value is also changed to 12.35 as a punishment. Only under state 3, the robot adopts action 3. It enters a better state. We assume it is always larger than 2 of the old value. In the case of 13, it should be 15. Although the robot selected the correct action, it still missed the goal. So the reward is 0. But it still can learn from entering a better state. Its value changed to 13.25. If it scored a goal, the new value should change to 18.25 instead. The robot keeps learning like this. We take each episode as the robot learns once under each different state.

After 10 episodes, we get,

Table 10.2 Robot Learning Result

State	Action	Value
S1	A1	16.8
	A2	10.8
	A3	10.6
S2	A1	11
	A2	25.7
	A3	11.7
S3	A1	11.0
	A2	12.0
	A3	38.0

We can see that the robot has converged to the correct decisions, i.e. in state 1, it always select action 1; in state 2, it always select action 2; in state 3, it always select action 3.

## ***Chapter 11 Summary and Future Work***

### **11.1 Summary and Contributions**

In this study, a mobile robot system was developed and preliminary study was conducted on it. In the system, the activities of robots are monitored by a video camera. The video camera captures the video images and sends them to the computer for analysis. The computer accepts every frame of image and uses related image processing techniques to realize target recognition and target tracking. This kind of computer vision provides a feedback for the computer to know where the interested objects are. The computer then uses the information to control the robots intelligently. Distributed computing is also implemented to facilitate the research.

The contributions of this work are summarized below:

1. Developed intelligent mobile robot system research platform.

Object-oriented programming (OOP) language, Visual C++, was used to program the system. It is easy to integrate knowledge, and easy to expand. The system was designed to dedicate to conducting research on it. Considerations were taken to facilitate experiments, such as computer vision effect monitoring, and video recording.

2. Solved the lighting problem

Lighting unstableness problem had significantly impacted the quality of robot performance. Nearest Neighborhood method is used to remedy the problem.

3. Implemented a basic behavior controller using fuzzy logic system.

Fuzzy control is a novel approach to solve some non-linear control problem when formulas to solve the problems are hard to find. We successfully used fuzzy logic to implement a basic behavior controller: moving robot to a designated position.



4. Realized game-playing by rule-based and frame-based knowledge representation.

To make robots playing games actually is how to transfer expert's knowledge to robots. We used rule-based and frame-based knowledge representation to realize robots playing game.

5. Designed a simple learning process.

In our research, by using Q-learning algorithm, we designed a simple real robot learning experiment: robot learning to shoot. The results show that robots can learn some simple knowledge by themselves.

## 11.2 Future Work

This study addresses many issues related to development and research on mobile robot system. There are several issues that need further investigation and refinement.

The robot hardware is bought from a Korea company. The price was high. In the future, we should design and build our own robots ourselves. On one hand, the cost would drop down. On the other hand, when there are possibly some problems on the robots, we can fix them by ourselves.

In the development of the platform, complex algorithms were used for image processing to implement computer vision and for solving the lighting unstableness problem. The target detection rate is only several frames every second. The control of robots slows down significantly. Robots can only run at a low speed. Some work should be conducted to improve the image processing speed by using other possible algorithms.

Vision on board and brain on board robot system should be considered as one of the future developments.

In this study, only a simple learning is implemented. In the future, we should consider more extensive self-learning of robots. We can further design many more interesting experiments based on this research platform. If we use our imagination, I believe we can dig a lot out of this platform.

## References

- [1] G. J. Awcock and R. Thomas, *Applied Image Processing*, McGraw-Hill Inc, New York, 1996.
- [2] Thaddeus J. Kowalski and Leon S. Levy, *Rule-Based Programming*, Kluwer Academic Publishers, Boston, 1996.
- [3] Shyi-Ming Chen, A knowledge acquisition scheme for rule-based systems, *Computer, Communication, Control and Power Engineering, Proceedings, TENCON'93.*, 1993 IEEE Region 10 Conference on, Vol 4, pages: 282-286
- [4] Bernad Jahne, *Digital Image Processing, Concepts, Algorithms, and Scientific Applications*, Third Edition, Springer, 1995.
- [5] Microsoft, *Serial Communication in Win32*, MSDN online documents.
- [6] Microsoft, *IPC and Windows 95*, MSDN online documents.
- [7] Microsoft, *Video Capture Driver Architecture*, MSDN online documents.
- [8] Graefe, V and Bischoff, R, A human interface for an intelligent mobile robot, *Robot and Human Cimmunication*, 1997. RO-MAN '97. Proceedings., 6<sup>th</sup> IEEE International Workshop on, 1997, pages: 194-199
- [9] Morita, T; Aramaki, S.; Kurono, S.; Kagekawa, K., A knowledge representation for the communication between robots. *Robot and Human Communication*, 1993. Proceedings., 2<sup>nd</sup> IEEE International Workshop on, pages: 308-313.
- [10] Micro Adventure, *Documents*, Micro Adventure Online documents, 1997.
- [11] Mark Beale and Howard Demuth, *Fuzzy system toolbox for use with Matlab*, PWS Productivity Tools, 1994.
- [12] Wei Chen, *Morphological image pyramids for automatic target recognition*, PHD thesis, 1997, Oklahoma State University.
- [13] John Walters and Norman R. Nielsen, *Crafting knowledge-based systems*, A Wiley-Interscience Publication, John Wiley & Sons, New York, 1988.

- [14] Jonathan H. Connell and Sridhar Mahadevan, Robot learning, Kluwer Academic Publishers, Boston, 1993.
- [15] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning, The MIT Press, Cambridge, Massachusetts, London, England, 1998.
- [16] Hee Rak Beom, A Sensor-Based Navigation for a Mobile Robot Using Fuzzy Logic and Reinforcement learning, IEEE transactions on systems, man, and cybernetics, vol. 25, No. 3, March 1995, pages: 464-477.
- [17] Sutton, Learning to predict by the methods of temporal differences, Machine Learning, 1988, Vol 3, pages 9-44.
- [18] Mance Harmon, An on-line reinforcement learning tutorial, <http://www-anw.cs.umass.edu/~mharmon/rltutorial/tut.html>.
- [19] Leslie Pack Kaelbling, Reinforcement learning: A survey, <http://www.cs.brown.edu/people/lpk/rl-survey/rl-survey.html>, on line document.
- [20] Roy Turner, Context-sensitive reasoning for autonomous agents and cooperative distributed problem solving, Department of Computer Science, University of New Hampshire, Durham, NH03857, USA.
- [21] Sendip Sen, Individual Learning of Coordination Knowledge", Department of Mathematical & Computer Science, University of Tulsa
- [22] Microsoft, CAsyncSocket class, MSDN online documents
- [23] Valter M. Van Buijtenen, et at, "Adaptive Fuzzy Control of Satellite Attitude by Reinforcement Learning", IEEE Transactions on Fuzzy Systems, Vol 6, No2, May 1998.
- [24] Edmund Durfee, "Distributed Problem Solving and Multi-Agent System: Comparisons and Examples", EECS Department, University of Michigan, Ann Arbor, MI 48109

- [25] Roy Turner, "Context-sensitive Reasoning for Autonomous Agents and Cooperative Distributed Problem Solving", Department of Computer Science, University of New Hampshire, Durham, NH03857 USA [rmt@unh.edu](mailto:rmt@unh.edu)
- [26] Dongil CHO, "Developing a Multi-Agent Model for Distributed Knowledge Systems", Graduate School of Systems Management, The University of Tsukuba. [Cho@gssm.otsuka.tsukuba.ac.jp](mailto:Cho@gssm.otsuka.tsukuba.ac.jp)
- [27] Patrick McDonnell, "A Reinforcement Learning Approach to Support Setup Decisions in Distributed Manufacturing Systems", Industrial and Manufacturing Engineering, Penn State University, PA 16802 [pjm@wimpy0.psu.edu](mailto:pjm@wimpy0.psu.edu)
- [28] Hagan, et al, " Neural Network Design", PWS Publishing Co. 95.
- [29] Powerpoint presentations for MIROSOT SYSTEM, Micro Adventure Co. Korea.

VITA

Fengming Yang

Candidate for the Degree of  
Master of Science

Thesis: DEVELOPMENT AND RESEARCH ON INTELLIGENT MOBILE ROBOT SYSTEM

Major Field: Electrical Engineering

Biographical:

Education: Received Bachelor degree in Electrical Engineering from Southeast University, Nanjing, China in July 1993; received Master of Engineering Degree in Electrical Engineering from Tsinghua University, Beijing, China in July 1996; completed requirements for the Master of Science degree at Oklahoma State University in May, 2000.

Professional Experience: Senior Software Engineer, Huawei Technologies Co. Ltd, Shenzhen, China, from September 1996 to June 1998.