VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Han Lin

BEHAVIOR DESIGN OF NAO HUMANOID ROBOT: A CASE OF PICKING UP A BALL AND THROWING INTO A BOX

Technology and Communication

2015

FOREWORD

This is my undergraduate thesis in Degree Programme in Information Technology, at Vaasan Ammattikorkeakoulu, University of Applied Sciences. I would like to take this opportunity to show my great gratitude to all who help me in the process of completing my thesis.

First of all, I would like to express my deepest appreciation to my supervisor, Dr. Yang Liu. During my study at VAMK, University of Applied Sciences, he has given me a lot of guidance in almost all my specialized courses. Besides, during my thesis period, he inspired me with many new ideas, which led me to have deeper academic research. And when I had problems, he could always provide me with constructive advices, so I could find the solution successfully. More importantly, he gave me the chance to access to the field of Nao robot, which really made me learn a lot in both theory study and practical application.

In addition, my sincere thanks also go to the staff at Vaasan Ammattikorkeakoulu, including Dr. Chao Gao, Dr. Ghodrat Moghadampour, Mr. Jani Ahvonen, Mr. Santiago Chavez Vega, Mr. Antti Virtanen, Mr. Jukka Matila, Dr. Smail Menani and all others staff who have taught me knowledge in the field of embedded system with patience. It is their help and guidance that broadens my horizons and makes me complete my Bachelor’s degree.

Last but not the least, I am very thankful for my parents who supported me to study in Finland and encouraged me to challenge myself in the process of completing the thesis. Also all my friends, including Lu Wei, in Botnia RoboCup laboratory should be appreciated for their help to my thesis. Hope they can obtain great achievement in the future.

Han Lin

Vaasa, Finland

20/04/2015

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Degree Program in Information Technology

ABSTRACT

Author	Han Lin
Title	Behavior Design of Nao Humanoid Robot: a Case of Picking up the Ball and Throwing into the Box
Year	2015
Language	English
Pages	80 + 4 Appendices
Name of Supervisor	Yang Liu

This thesis introduces the behavior design of the Nao humanoid robot: a case of picking up the ball and throwing into the box. In this thesis, red ball recognition, strategy design for tracking the ball, picking up the ball and finding the box as well as the animation design are involved.

In this thesis, Hough circle transform is utilized to detect the center of the ball precisely in the view of the robot in the vision system. In addition, probability theory is applied in designing and optimizing the strategy of picking up the ball autonomously. More importantly, the strategies of tracking the ball dynamically and calculating the distance to the box are designed based on mathematical models. And key frames in timeline are used to design the animations. Therefore, the robot can achieve this project successfully.

This project was completed by using Python in Windows platform. And Eclipse was used as the environment for programming Python. Besides, OpenCV and Python were used to recognize the red ball. In addition, Choregraph was used to design the animations and Matlab was used to find the location of the box. The robot is the newest Nao robot which is version 5. In this thesis, the implementation method was mainly software engineering which involves planning, developing, debugging and testing.

Moreover, achieving the autonomy of the robot, which is to make the robot behave like a real human being, tends to be a popular field. And in this project, the robot achieves the ability of picking up the ball and throwing the ball into the box autonomously. And it can be concluded that mathematical theory support is very significant to achieve the autonomy of the robot.

Key words Nao robot, vision system, strategy design, animation design

CONTENTS

LIST OF ABBREVIATIONS

PC	Personal Computer
RoboCup	Robot Soccer World Cup
V5	Version 5
V	Volt
Ah	Ampere Hour
DCM	Device Communication Manager
SDK	Software development kit
IP	Internet Protocol
3D	three dimension
DHCP	Dynamic Host Configuration Protocol
IDE	Integrated Development Environment
GUI	Graphical User Interface
Fps	Frame per second
OpenCV	Open Source Computer Vision Library
BSD-licensed	Berkeley Software Distribution license
VGA	Video Graphics Array
RGB	Red Green Blue
PNG	Portable Network Graphics
HSV	Hue Saturation Value

LIST OF FIGURES AND TABLES

LIST OF APPENDICES

1 INTRODUCTION

1.1 Purpose of Thesis

This thesis introduces design and implementation details about how to program the humanoid Nao robot to make it pick up a red ball and throw it into a box. The thesis is mainly about the strategies of finding the red ball and the landmark of the box, precise identification of the red ball and animation design with timeline.

1.2 Overview Structure of Thesis

This thesis consist of eight chapters. The first chapter introduces the background information and technical reviews of this thesis as well as the motivation of this thesis. The second chapter generally describes the overview structure of the Nao application, it explains the logical connection between each part of the application, and it explains different movements of the robot in this project. The third chapter presents the vision module of this application. In this chapter, the implementation details about how to achieve the precise recognition of the red ball is introduced. Then, Chapter 4 is the strategy design part which includes finding ball strategy and finding Nao mark strategy. Chapter 5 introduces the implementation details which contain setting the environment for Python, Eclipse and OpenCV, and animation design as well as a few codes explanation. Then, Chapter 6 provides the overview of future research, Chapter 7 is the conclusion of this thesis.

1.3 Brief introduction of Nao

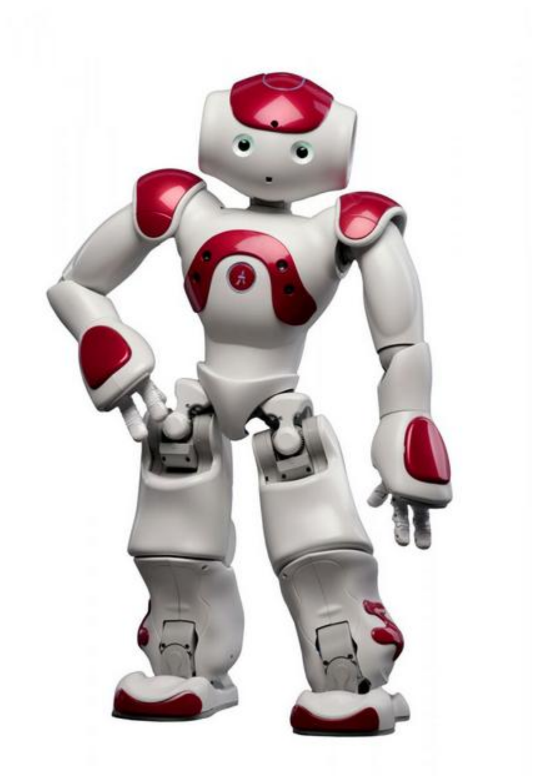


Figure 1. Humanoid Nao robot /1/

As shown in **Figure 1**, Nao is a 58-cm tall humanoid robot from Aldebaran Robotics. These robots are medium-sized open architecture robots. This robot is able to move, recognize people and communicate with human beings. Now Nao robots tend to be more and more popular around world; Naos are being used around world for educational and research purpose in over 480 universities. /2/, /3/

The Nao robots communicate with the PC through a cable or wireless networks. In addition, different Nao robots can interact with each other by using infrared sensors, camera, microphone, wireless network and speaker. Besides, DCM conducts the communication between the devices of Nao such as the actuator and sensors. /3/

The Nao software is based on Gentoo Linux, and it supports all kinds of programming language such as Python, C++, Java and .Net Framework. Besides, it also supports graphic-based Programming which makes the programming of this Nao robot easier. /3/

1.4 Background of Humanoid Nao Robot Team in Botnia

Since the Nao robot was chosen instead of Sony’s Aibo(puppy robot) to be the official platform of RoboCup in August 2007, the Nao robot with humanoid appearance have become more and more popular and worth of study. So in 2014, two Nao robots V5 with H25 body type was bought by VAMK for educational and research purpose. Therefore, the Botnia Nao robot team was set up and leaded by Dr. Liu Yang. Now, many applications have been completed successfully by the Botnia Nao robot team; e.g. the Nao robot can climb spiral stairs and play the Candy Crush game.

1.5 Features of Nao Robot V5

The height of Nao V5 is 57.4 and its width is 27.5 cm. And it weighs 5.4 kg. Besides, a special plastic material is utilized for the body of the robot. And it has a 21.6V 2.25 Ah battery that can be used up to 90 minutes for normal use and 60 minutes for active use. /3/,/4/,/5/

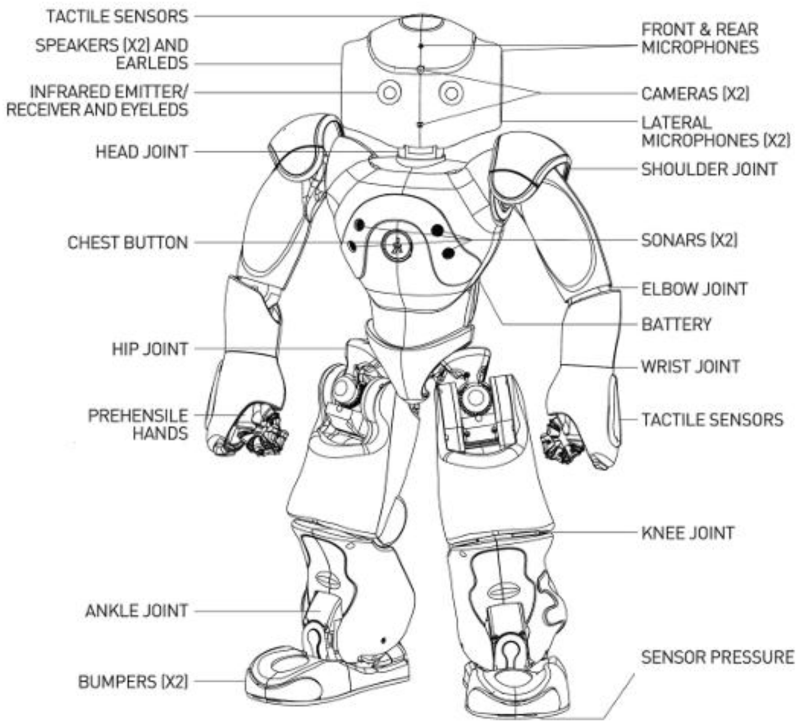


Figure 2. Nao H25 Features/6/

More importantly, as shown in Figure 2, this robot has 25 total joints and many kinds of sensors, such as infrared sensors and ultrasonic sensors (sonars). /3/,/7/

1.6 Software of Nao Robot

Nao supports Choregraphe, Naoqi and Monitor as development software.

1.6.1 Choregraphe

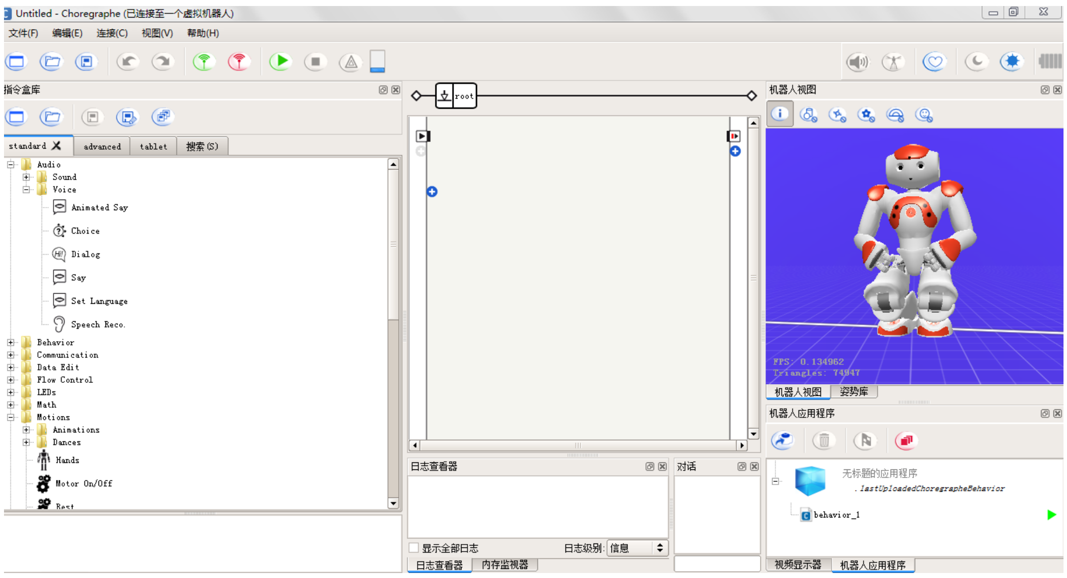
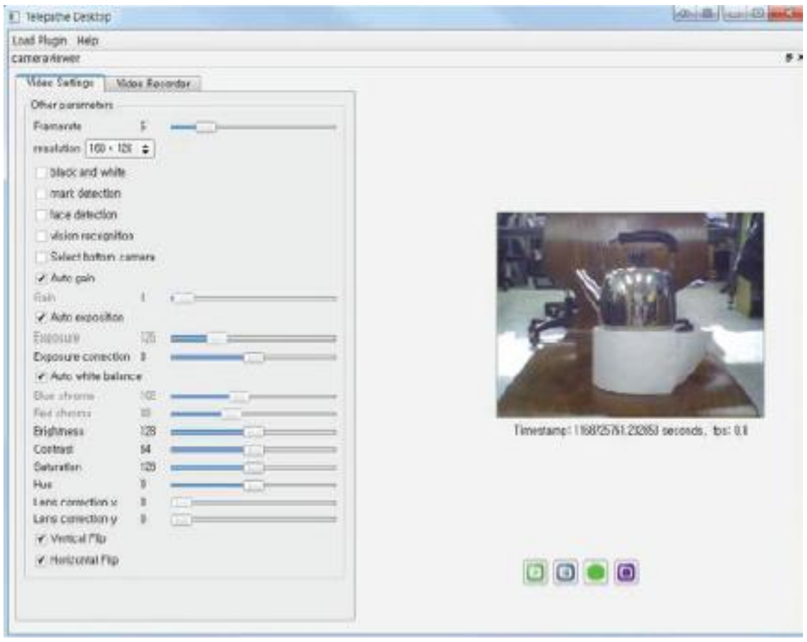


Figure 3. Choregraphe interface

As shown in Figure 3, Choregraphe is a cross-platform application that can program the behaviors of the Nao robot through graphics-based programming. This software is suitable for beginners, since it makes program visualized and easier.

1.6.2 Monitor



Other parameters

```
Framerate               5
resolution  160 × 120
[ ] black and white
[ ] mark detection
[ ] face detection
[ ] vision recognition
[ ] Select bottom camera
[✓] Auto gain
Gain                    1
[✓] Auto exposition
Exposure                125
Exposure correction     0
[✓] Auto white balance
Blue chroma             102
Red chroma              [illegible]
Brightness              128
Contrast                64
Saturation              128
Hue                     0
Lens correction x       0
Lens correction y       0
[✓] Vertical Flip
[✓] Horizontal Flip
```

Timestamp: 1168725761.232853 seconds, fps: 0.8

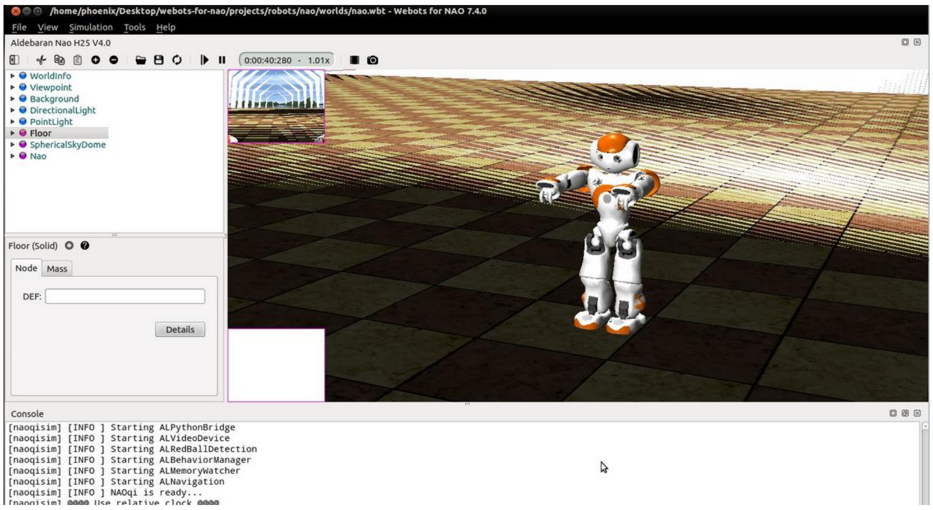
Figure 4. Monitor interface

Monitor is a program that displays real-time data about the cameras and memory of the Nao robot. (Figure 4) Furthermore, this program can also be used to save images or videos inside the robot and transmit to the user's PC. In addition, the value of the variables used by the Nao's system can be monitored with this program.

1.6.3 Naoqi

Naoqi is the programming framework for programming Nao. This specific frame-work contains parts such as parallel processing, resources management, synchroni-zation, and event processing mainly required for robotics. Naoqi is a SDK written in C++ and it has the functions such as calling python or C++. /3/

1.6.4 Webots



```
[naoqisim] [INFO ] Starting ALPythonBridge
[naoqisim] [INFO ] Starting ALVideoDevice
[naoqisim] [INFO ] Starting ALRedBallDetection
[naoqisim] [INFO ] Starting ALBehaviorManager
[naoqisim] [INFO ] Starting ALMemoryWatcher
[naoqisim] [INFO ] Starting ALNavigation
[naoqisim] [INFO ] NAOqi is ready...
[naoqisim] @@@@ Use relative clock @@@@
```

Figure 5. Webots interface/8/

Webots is a simulator developed by Cyberbotics company for modelling, programming and simulating many kinds of robots including the Nao robot. And it can be used to create virtual worlds for the Nao robots. /9/

1.7 Programming the Nao Robot

Fortunately, each Nao's software and software development tools are designed to have three different versions to make it suitable for Windows, Mac and Linux operating systems respectively. Therefore, the user can program in Windows, Mac or Linux operating systems environment.

First, Choregraph tends to be suitable for beginners who are not familiar with the programming. As shown in Figure 6, area 1 is the box library, and area 2 is the diagramming space. Area 3 is the menu screen. In the box library, there are all kinds of boxes written in Python listed by function. Then, the box in the box library can be dragged and dropped in the diagramming space. In addition, the input and output of the box can be connected as shown in Figure 6. Finally, by clicking the ”play” icon in the menu screen, the program runs, and the 3D Nao will move according to the program. And there is a video monitor embedded in this software, which can be used to check the view of the robot.

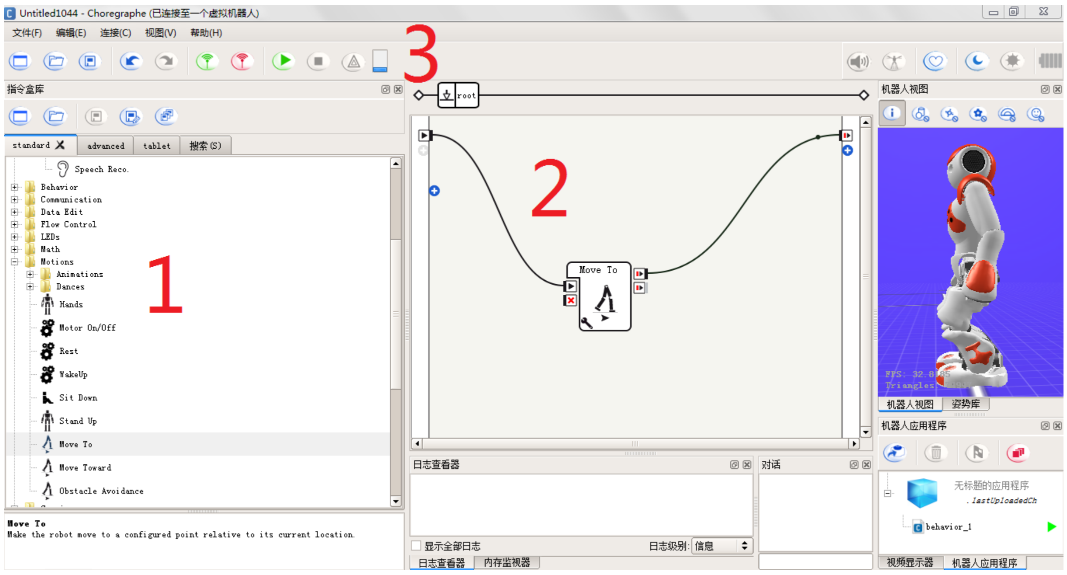


Figure 6. Program using Choregraphe

Furthermore, Python and C++ can also be used in programming Nao. Python is an object-oriented, interpreter-based language. The advantage of the Python language is that it is able to quickly check the content being tested. On the one hand, Python can be used in Choregraph to edit the existing box or creating a new box with the function which is not supported by the existing boxes. On the other hand, Python can also be used to directly control Nao's hardware through linking Naoqi or DCM. In addition, C++ can also be used to directly control Nao by linking Naoqi. In this project, Python was used to program the Nao robot. /3/

1.8 Communication Module

Nao can communicate with PCs either a wired connection using an Ethernet cable or wireless connection using WI-FI. But in order to use the wireless network, the router which supports the DHCP function needs to be set up first. Once the robot is connected to the network, the Nao's IP address is entered into the web browser, then the user can monitor the status of the Nao robot.

Chroregraph can connect to the real robot through the IP address and the port of the broker of the robot. Namely, after clicking the "connect" icon, Choregraph can connect to the robot.

Moreover, Python, C++ or other programming languages can also control the robot directly by using Naoqi, and it works as the user and Nao system communication. The Naoqi that executes on the robot is a broker. As shown in Figure 7, when Naoqi starts running, a preferences file called “autoload.ini” which defines which libraries it is supposed to load is loaded. In addition, each library contains modules which includes their methods./3/ /10/

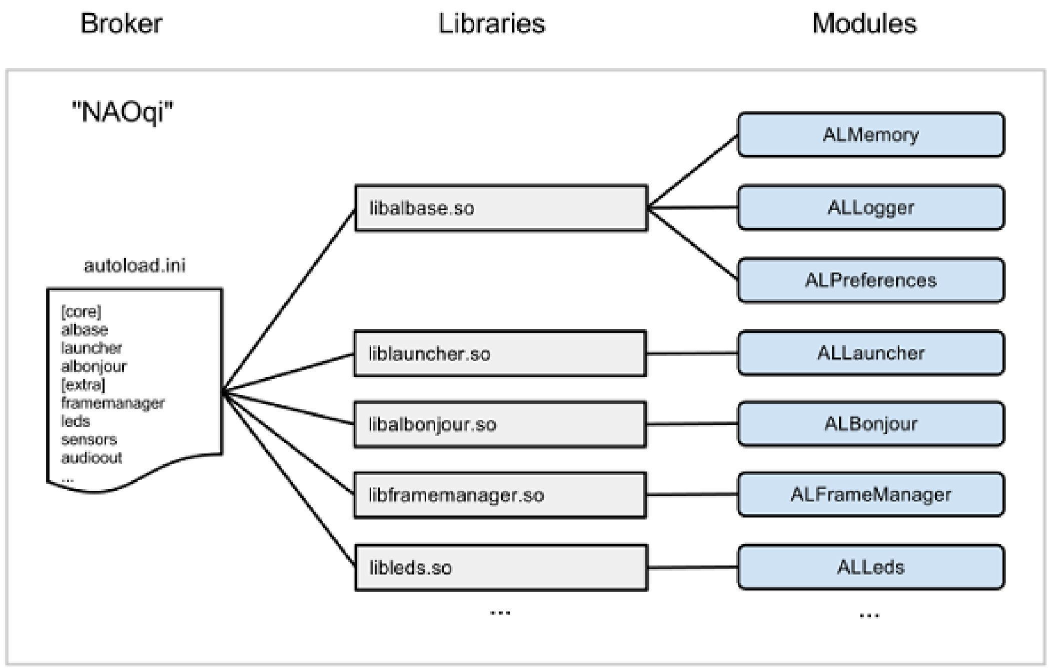


Figure 7. The way that Naoqi executes/10/

1.8.1 Broker

A broker is an object that has two main functions. First, it supports lookup services which is to find the corresponding modules and methods. Secondly, it provides network access to enable the methods of attached module to be called from the outside the process. Moreover, the broker will run transparently, so normally, the user do not need to think about the broker. /10/

1.8.2 Proxy

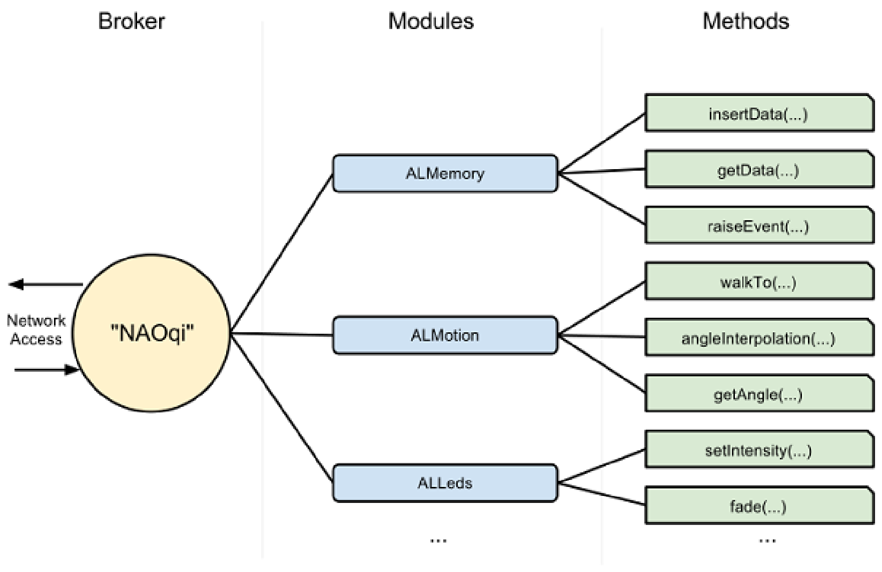


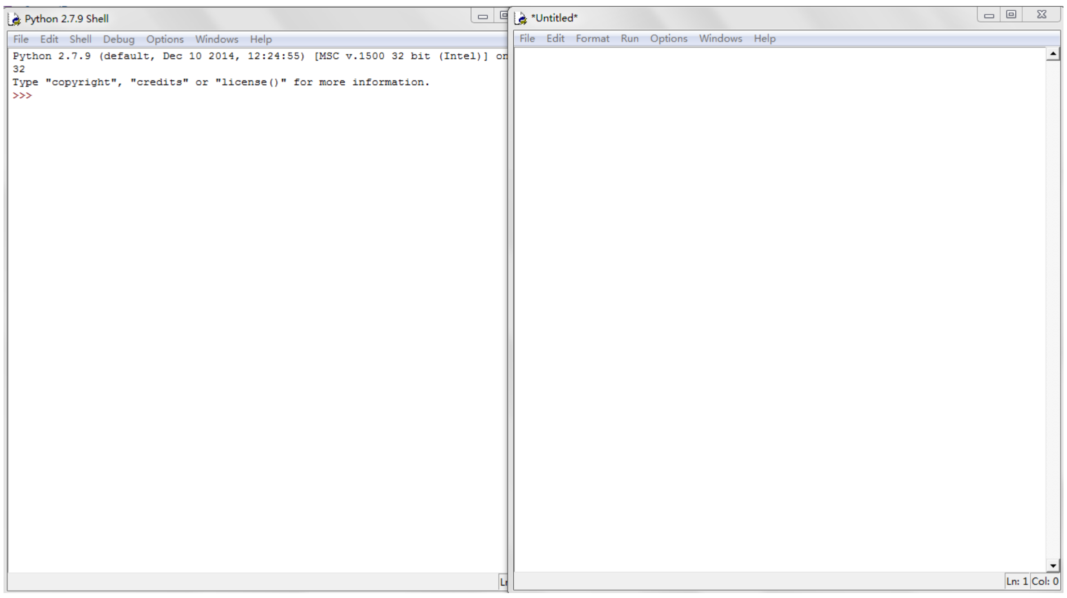
Figure 8. Broker, modules and methods/10/

As shown in Figure 8, a proxy is an object that behaves as the module it represents. For example, if a proxy is created and a module and the IP address and the port of the broker is specified, an object containing all the methods of that module will be obtained. In addition, by calling the method, the robot will behave as the program. /10/

1.9 Environment for Using Python

There is a considerable number of IDE which can be used to program Python, such as IDLE and eclipse. But in this project, Eclipse was used as the Python IDE.

IDLE


Python 2.7.9 Shell
File Edit Shell Debug Options Windows Help
Untitled
File Edit Format Run Options Windows Help
Ln: 1 Col: 0

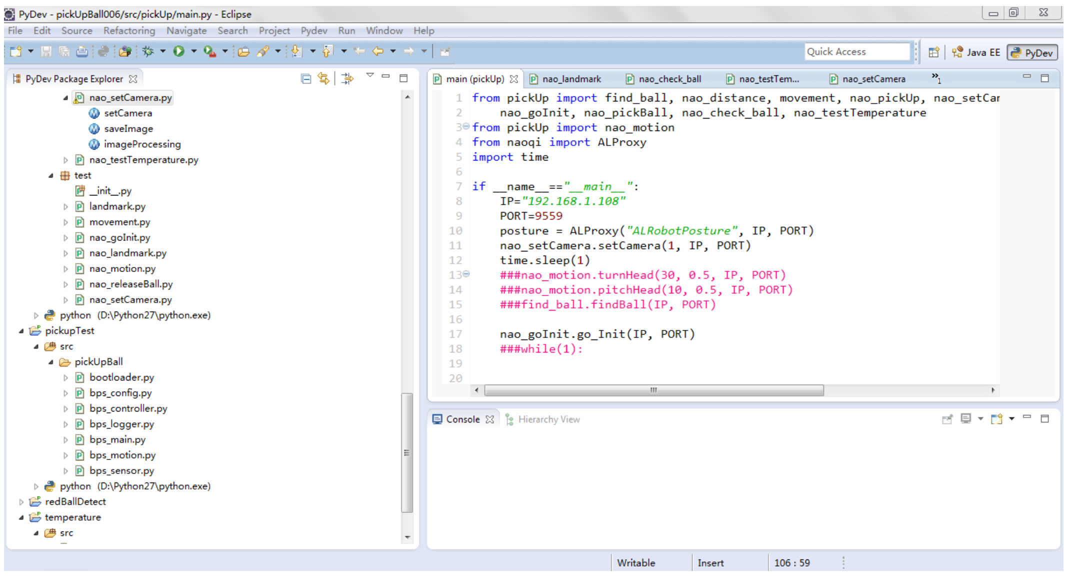
```
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)] on
32
Type "copyright", "credits" or "license()" for more information.
>>>
```

Figure 9. IDLE interface

IDLE is the Python IDE built with the tkinter GUI toolkit and it has two main window types, shell window and editor window as shown in Figure 9. /11/

Eclipse and PyDev

Eclipse is an IDE which can be used to develop applications in all kinds of languages, such as Java. In addition, PyDev is a plugin that enables Eclipse to work as a Python IDE. So in this project, Eclipse was chosen to program the Nao robot. The interface is shown in Figure 10. /12/



```python
from pickUp import find_ball, nao_distance, movement, nao_pickUp, nao_setCar
    nao_goInit, nao_pickBall, nao_check_ball, nao_testTemperature
from pickUp import nao_motion
from naoqi import ALProxy
import time

if __name__=="__main__":
    IP="192.168.1.108"
    PORT=9559
    posture = ALProxy("ALRobotPosture", IP, PORT)
    nao_setCamera.setCamera(1, IP, PORT)
    time.sleep(1)
    ###nao_motion.turnHead(30, 0.5, IP, PORT)
    ###nao_motion.pitchHead(10, 0.5, IP, PORT)
    ###find_ball.findBall(IP, PORT)

    nao_goInit.go_Init(IP, PORT)
    ###while(1):


```

Figure 10. Eclipse interface

1.10 Motivation

The field of robotics is becoming more and more popular around the world. More importantly, the humanoid Nao robot is worth of being researched. There are lots of fields about the Nao robot that are valuable for researching such as vision system of robot, robotic kinematics as well as animation design. So it is really a good opportunity for me to improve my programming skills and broaden my horizons.

Furthermore, many applications has been achieved by our Nao robot team at Botnia. For instance the Nao robot can even kick the ball. At that time, my supervisor suggested me with the topic: programming the Nao robot to pick up the ball and throw it into the box. In this project, vision system and animation design was involved. In my view, this application is quite valuable. In the future, the robot can recognize and pick up any stuff to help human beings to collect different things at home.

2 STRUCTURE OF THIS NAO APPLICATION

2.1 Flow Chart of the Whole Project

Figure 11 is the flow chart of the whole project. First, the top camera of the robot was used to track the red ball. After finding the ball, the robot would squat and find the precise location of the ball. Then the robot would pick up the ball and check whether the ball was in hand. And if the ball was not in hand, the robot would find the ball again. And if the ball was in hand, the robot would check the temperature of its joints. Then the bottom camera was used to find the Nao mark of the box. Finally, the robot would walk towards the box and throw the ball into the box and say “I succeed”.

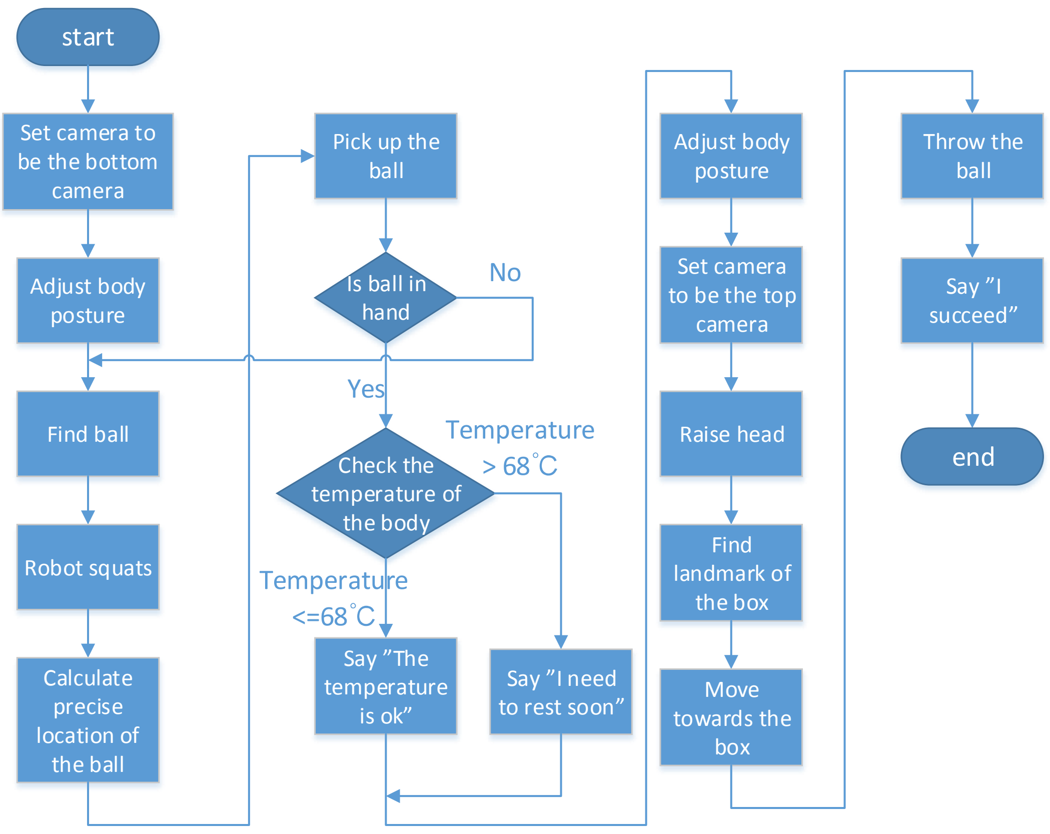


Figure 11. Flow chart of the whole project

2.2 Behaviors

In this part, some important behaviors of the robot in this project will be present.

2.2.1 Start animation

Two start animations were designed in this project. First, the stand-initial posture was used to initialize the posture of the robot after it woke up from the rest posture, since the stand-initial posture is a good pose for transitioning to the next movement. The posture is as shown in Figure 12.

Secondly, the go-initial posture was designed to adjust the posture of the robot before it starts walking or turning its body or doing the next movement. Since sometimes the robot cannot walk or do the next movement without reasons, after initializing the robot with this posture, the robot can move easily. The posture is as shown in Figure 13.

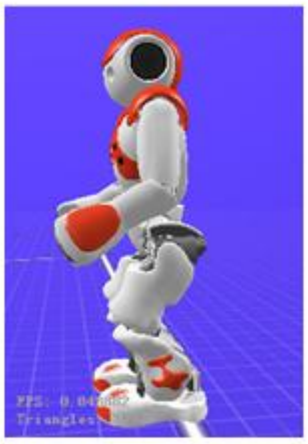


Figure 12. Stand-initial

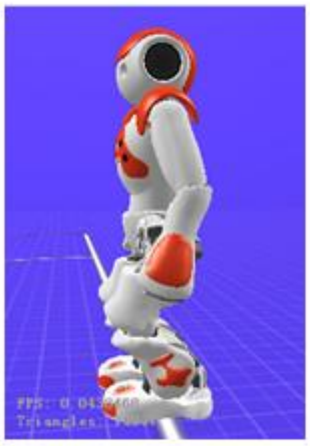


Figure 13. Go-initial

2.2.2 Finding the Ball

When Nao starts looking for the ball, it first will turn its head to search the position of the ball. Then the robot will turn its body to face the ball directly and then the robot will walk towards it.

2.2.3 Squatting

After the robot reaches the specific position, it will squat and turn its head to find the precise position of the ball. Since picking up this ball in this project needs an accurate position of the ball, before picking up the ball, the accurate positioning of the ball is started. The squat posture is as shown in Figure 14.

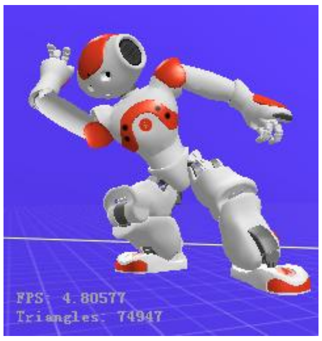


Figure 14. Robot squats

2.2.4 Picking up the Ball

After finding the precise location of the ball, the robot will use one hand to lift up the ball. The robot will stand up and check whether the ball is in hand. If the ball is not in hand, the robot will find and pick up the ball again.

2.2.5 Finding the Box with Holding the Ball in Hand

In this behavior, the robot will look for the Nao mark of the box and move towards the box. During that movement, it must hold and keep the ball in its hand.

2.2.6 Throwing the Ball into the Box

When the robot is in front of the box, it will raise its right arm and open its hand. Then the ball will drop into the box from the hand of the robot. Finally, the robot will say ”I succeed”.

3 VISION MODULE

In order to pick up the ball successfully, the precise location of the ball in the robot's view needs to be achieved. Although there is a method that can return the position of the ball, the value it returns is not precise enough to pick up the ball. So a vision module was created to return the coordinates of the center of the ball. In addition, by using the location of the ball, the robot is able to choose the best posture to increase the possibility of picking up the ball.

3.1 Hardware Part for Vision Module

In this section the definition of the ball and the specifications of robot's cameras is presented in details.

3.1.1 Definition of the Ball

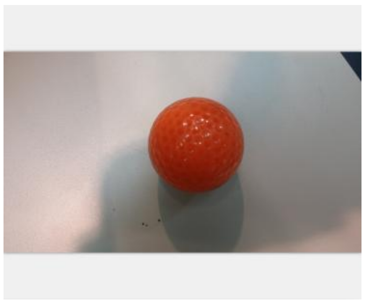


Figure 15. The red ball

The ball shown in Figure 15 was used in this project. It is obvious that the color of the ball is red and the surface of the ball is smooth. Besides, the ball is hard and the diameter of the ball is 4.5 centimeters.

3.1.2 Comparison between Picking up a Hard Ball and a Soft Ball

Figure 16. A red soft ball

Based on testing, picking up a hard ball is more complicated than picking up a soft ball. First, the surface of the hard ball in this case is quite smooth, so even if the robot picks up the ball there still is the possibility that the ball will drop from its hand if the ball is not in the center of the hand. In addition, the ball will move to somewhere else out of robot’s sight easily even with a light touch of the hand of the robot while the robot is picking up the ball. Therefore picking up a hard ball tolerates smaller errors in finding the location of the ball.

However, when it comes to picking up a soft ball, it seems to be easier. Since the shape of the soft ball can be changed, so it is easier to pick it up. Moreover, through increasing the strength of grasping the ball, the ball will not drop from its hand so easily, even if the ball is not in the center of its hand. Besides, the surface that the soft ball touched the ground is bigger than the hard ball, so the soft ball will not move easily with a light touch of robot’s hand when the robot is picking up the ball. Therefore picking up a soft ball accepts larger errors in searching for the location of the ball.

3.1.3 Technical Overview of Cameras

There are two cameras located in the forehead of the robot and those cameras support an up to 1280*960 resolution at 30 fps. The location of the top camera is nearly at the robot’s eyes level and the location of the bottom camera is at its mouth level. Figure 17 and Figure 18 shows the location of two cameras.

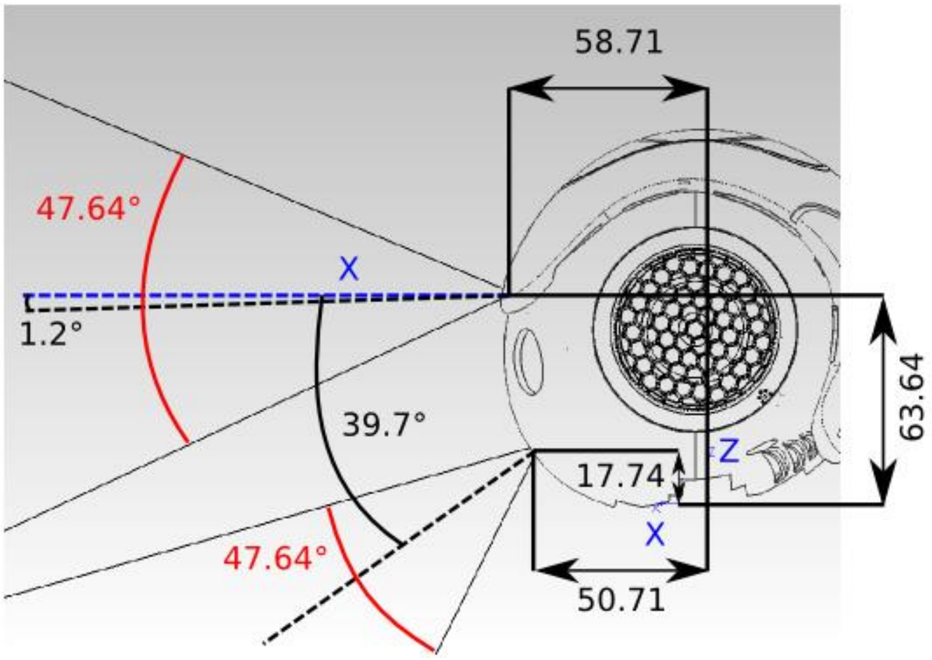


Figure 17. Side view of cameras/13/

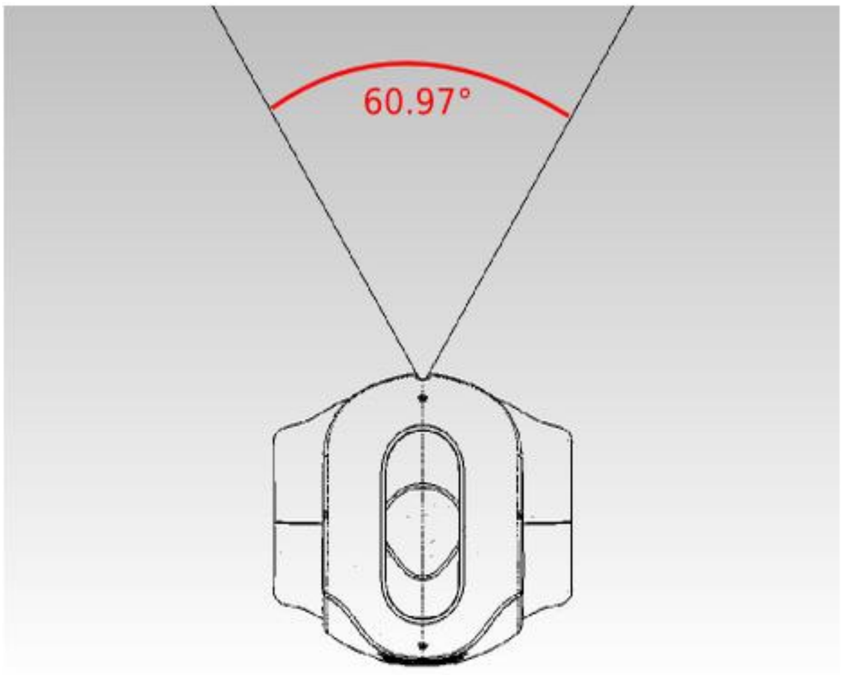


Figure 18. Top view of cameras/13/

As shown in Figure 17, if the head of the robot keeps in that position, the vertical range of the camera is 47.64˚ for each camera. Figure 19 shows the range of head pitch angle which is up to 68 ˚.

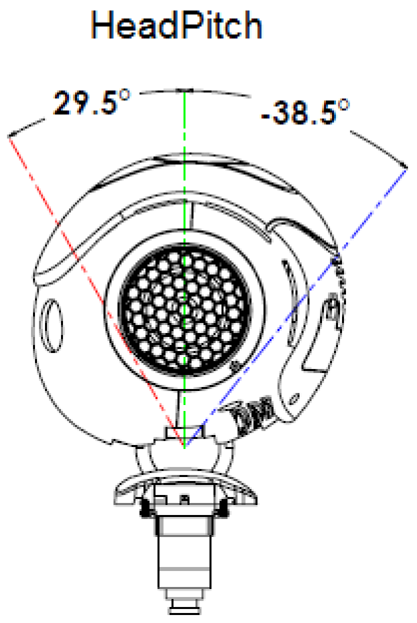


Figure 19. Head pitch angle from side view/14/

Figure 20 is the diagrammatic schematic of the vision range of those two cameras with the movement of the head. With the bottom camera, the red ball can be recognized when the distance between the ball and the feet is larger than 20 cm and smaller than 120 cm. In addition, with the top camera, as shown in Figure 20, the robot is supposed to look farther theoretically, but its view will easily be influenced by interferences such as lights and colorful objects on the wall. In this project, the bottom camera was used to search for the ball. But when it comes to looking for the Nao mark of the box, since the box is always very far from the position of the robot, the top camera was used to searching for the box.

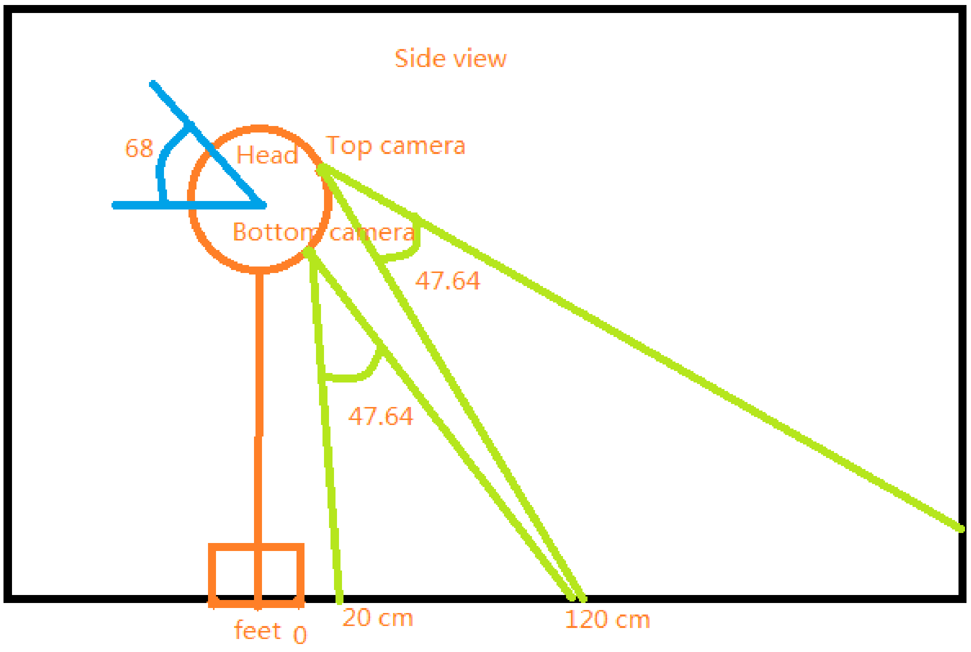
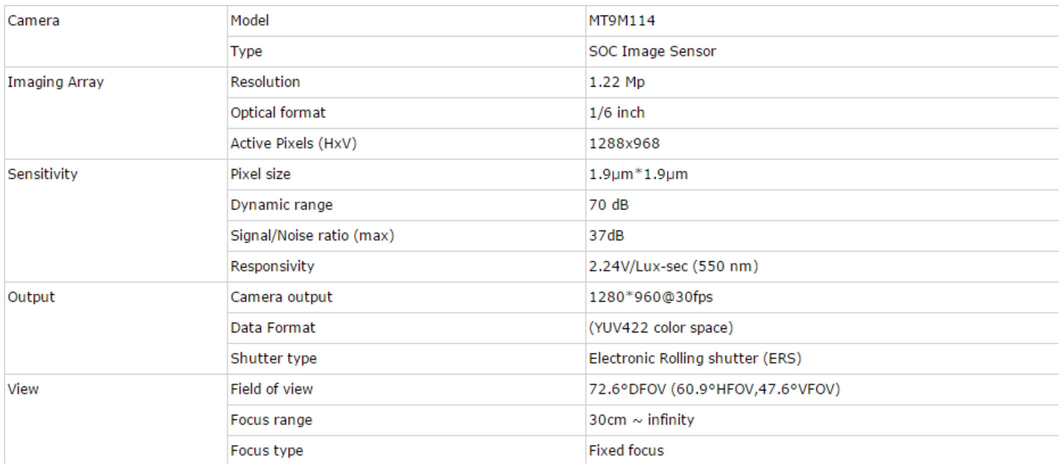


Figure 20. Distance that Nao can watch with its cameras

Table 1 is the data sheet of the cameras of the robot.

Table 1. Data sheet of cameras/13/



Camera	Model	MT9M114
	Type	SOC Image Sensor
Imaging Array	Resolution	1.22 Mp
	Optical format	1/6 inch
	Active Pixels (HxV)	1288x968
Sensitivity	Pixel size	1.9µm*1.9µm
	Dynamic range	70 dB
	Signal/Noise ratio (max)	37dB
	Responsivity	2.24V/Lux-sec (550 nm)
Output	Camera output	1280*960@30fps
	Data Format	(YUV422 color space)
	Shutter type	Electronic Rolling shutter (ERS)
View	Field of view	72.6°DFOV (60.9°HFOV,47.6°VFOV)
	Focus range	30cm ~ infinity
	Focus type	Fixed focus

3.2 Brief Introduction to OpenCV

OpenCV is an open source BSD-licensed library that contains a considerable number of computer vision algorithms. It has C++, C, Python and Java interfaces and it supports many operating systems such as Windows. In addition, OpenCV emphasizes on computational efficiency and focuses on real time applications. In this project, OpenCV and Python were used to process the image. /15/, /16/

3.3 Overview Structure of the Vision Module

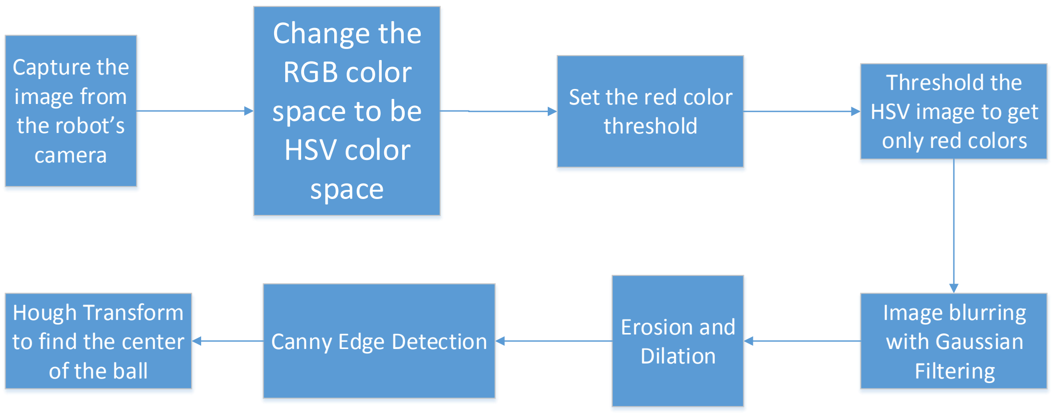

Capture the image from the robot's camera
Change the RGB color space to be HSV color space
Set the red color threshold
Threshold the HSV image to get only red colors
Image blurring with Gaussian Filtering
Erosion and Dilation
Canny Edge Detection
Hough Transform to find the center of the ball

Figure 21. Structure of the Vision module

Figure 21 shows the steps to find the location of the ball in the robot's view.

3.4 Image Acquisition

As shown in Figure 22, first, an object called "camProxy" of "ALProxy" was created and the module which is "ALVideoDevice" was specified. This module is in charge of providing images from the video source. In addition, the resolution of the image was set to be VGA which means 640*480 pixel. The resolution of VGA is enough in this project, if the resolution is higher, the efficiency of image processing will be decreased. In addition, the color space of the image was set to be RGB, since when creating the image by using "fromstring", this method only supports creating the RGB image. Then a vision module was subscribed to "ALVideoDevice", because this vision module is remote, an array containing all the image data would be obtained by using a method which is "getImageRemote". Finally, the image is saved as a PNG image on the local computer. The image obtained is as shown in Figure 23.

```
def saveImage(IP,PORT):
    camProxy = ALProxy("ALVideoDevice", IP, PORT)
    resolution = 2    # VGA
    colorSpace = 11   # RGB
    videoClient = camProxy.subscribe("python_client", resolution, colorSpace, 5)

    t0 = time.time()
# Get a camera image.
# image[6] contains the image data passed as an array of ASCII chars.
    naoImage = camProxy.getImageRemote(videoClient)
    t1 = time.time()
# Time the image transfer.
    print "acquisition delay ", t1 - t0
    camProxy.unsubscribe(videoClient)
# Now we work with the image returned and save it as a PNG  using ImageDraw
# package.
# Get the image size and pixel array.
    imageWidth = naoImage[0]
    imageHeight = naoImage[1]
    array = naoImage[6]

# Create a PIL Image from our pixel array.
    im = Image.fromstring("RGB", (imageWidth, imageHeight), array)

# Save the image.
    im.save("camImage.png", "PNG")

    #im.show()
```

Figure 22. Code for image acquisition

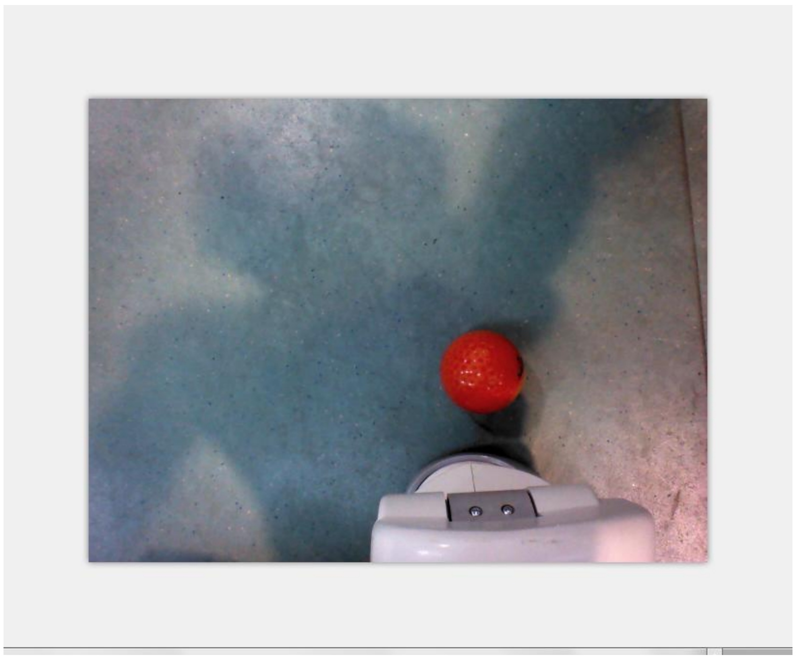


Figure 23. Original image

3.5 Capturing the Red ball in the Image

Since the image has already been obtained from the robot’s camera, capturing the red color is needed. And if the red color is captured, it is easier to find the contour of the red ball in the image.

3.5.1 RGB Color Space

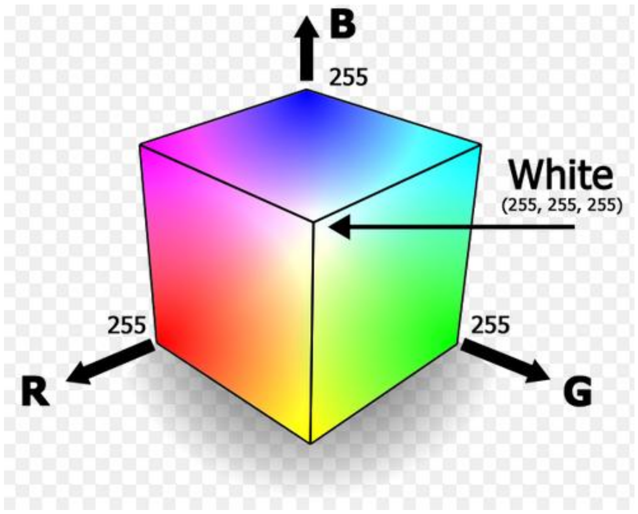


Figure 24. RGB color space/17/

As shown in Figure 24, the RGB color space is like a cube. RGB stands for red, green and blue respectively. All the other color are the combination of these three colors. For instance, if the color of a point in an 8 bit image is pure red, then the RGB value of this color is [255, 0, 0]. And if the color of a point in an 8 bit image is pure green, the RGB value of this color is [0, 255, 0]. But this color space is easily influenced by the interference such as sunshine and light. Since the same color in different intensity of light in the robot’s camera is different, this color space is not suitable for vision recognition.

3.5.2 HSV Color Space

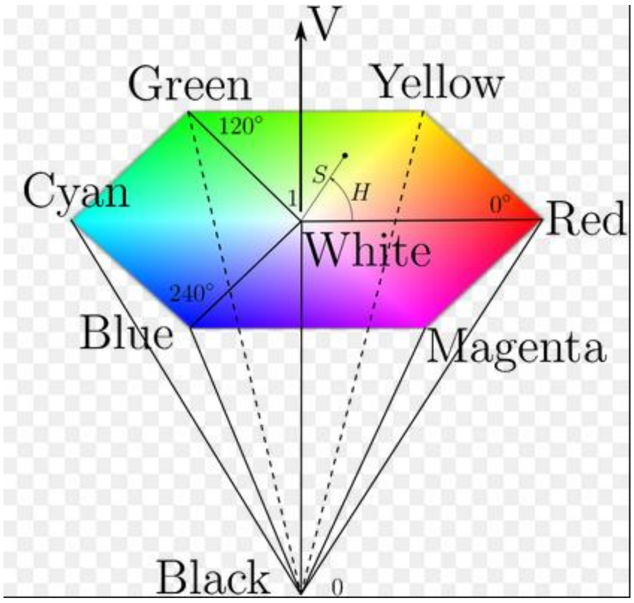


Figure 25. HSV color space/18/

Figure 25 shows the HSV color space, HSV stands for hue, saturation, value respectively. The range of hue is from 0˚ to 360 ˚ and Hue represents all the color range. When the color is red, the hue value of this color is 0 and if the color is green, the hue value of this color is 120. In addition, saturation stands for the saturation level of the color. For instance, if the color is pure red, the saturation value of this color is 1. Moreover the value stands for brightness. For instance when value equals 0, it stands for black color. More importantly, the value of HSV is the feature of the object, so it will not be influenced by the environment easily. Therefore, in this case, HSV image was used to capture the only red color.

3.5.3 Transferring the RGB Color Space to HSV Color Space

The HSV color space is suitable for vision recognition, but the image obtained is in the RGB color space. Therefore, the RGB color space needs to be converted to the HSV color space. There are formulas to convert the RGB value to HSV value.

$$V = \max(R, G, B) \tag{1}$$

$$S = \begin{cases} \frac{V-\min(R,G,B)}{V} & if(V \neq 0) \\ 0 & if(V = 0) \end{cases} \tag{2}$$

$$H = \begin{cases} \frac{60*(G-B)}{V-\min(R,G,B)} & if(V = R) \\ 120 + \frac{60*(G-B)}{V-\min(R,G,B)} & if(V = G) \\ 240 + \frac{60*(G-B)}{V-\min(R,G,B)} & if(V = B) \end{cases} \quad (3)$$

$$if\ (H < 0), H = H + 360 \quad (4)$$

In this case, $0 \leq H \leq 360,\ 0 \leq S \leq 1\ and\ 0 \leq V \leq 1$ (5)

But if it is an 8 bit picture, the value of HSV will be converted to be:

$$V = 255 * V \quad (6)$$

$$S = 255 * S \quad (7)$$

$$H = \frac{H}{2} \quad (8)$$

And in this case, the image obtained is an 8 bit RGB image. But in Opencv a pre-defined method can be used to convert the RGB image to be the HSV image directly. First, the library called “cv2” and “numpy” is imported, then all the methods of Opencv can be called. In addition the image obtained from the robot’s view is read by using “imread()” method. Finally, the method “cvtColor()” was used to convert the image to be the HSV image. The first parameter of this method is the input image and the second parameter of the method is the type of conversion which is cv2.COLOR_RGB2HSV. The HSV image is as shown in Figure 26.

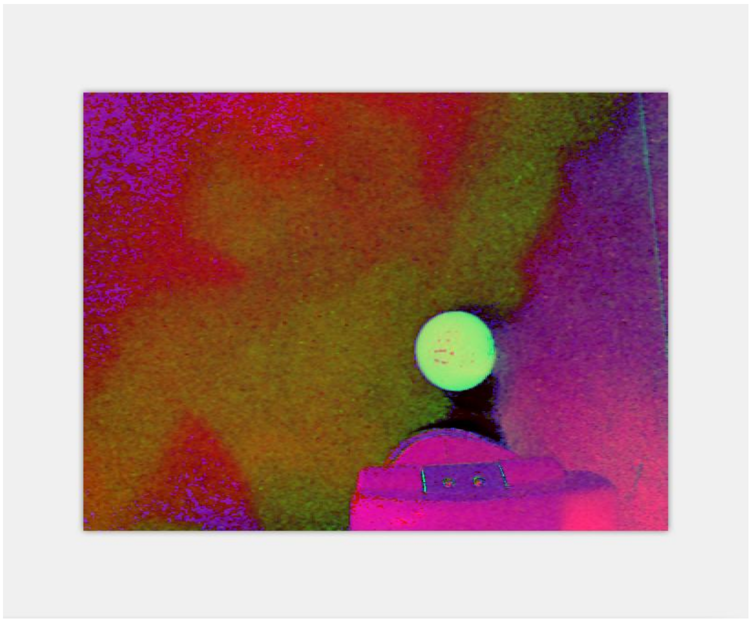


Figure 26. HSV image

3.5.4 Capturing the Red Ball in the Image

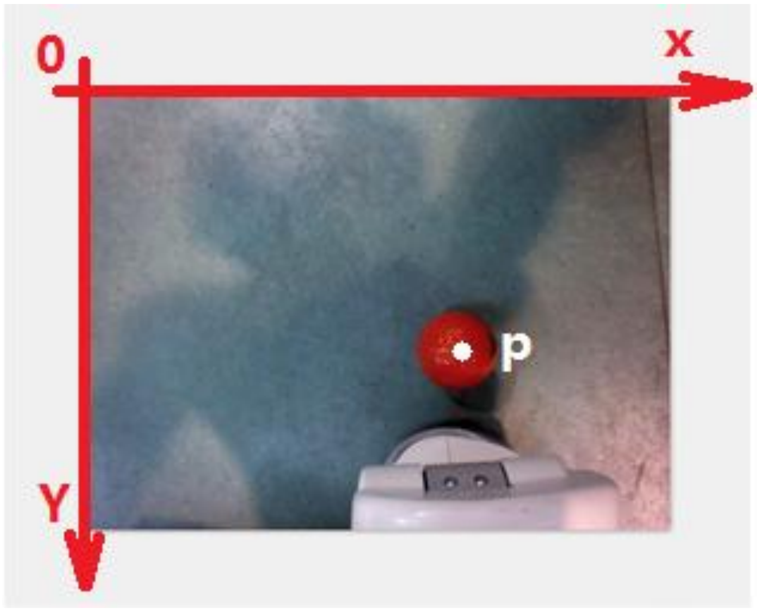


Figure 27. Taking one point from the red ball

In order to find the threshold value of the red color in HSV, first the RGB value of that color needs to be obtained. So one point called p was taken, as shown in Figure 27. The pixel coordinate of the point p is (465,229) and by using the pixel coordinate of the point p, the BGR value of this point which is can be obtained in OpenCV. In OpenCV, only the BGR value of the point can be obtained. In addition, BGR value means that the order of red value and blue value in the RGB value is inversed. Finally, the method cvtColor() was used to convert this BGR value to be HSV value. One piece of code about this can be found in Appendix 1.

According to the HSV value, the lower threshold value and the upper threshold value was set. Many different threshold values were tried to capture the red color, eventually the best threshold value which were [109,90,90] and [129,237,237] was found. Finally, the method “inRange(inputImage, lower_threshold, upper threshold)” was used to capture the only red color. The result is as shown in Figure 28. The red ball was captured successfully.

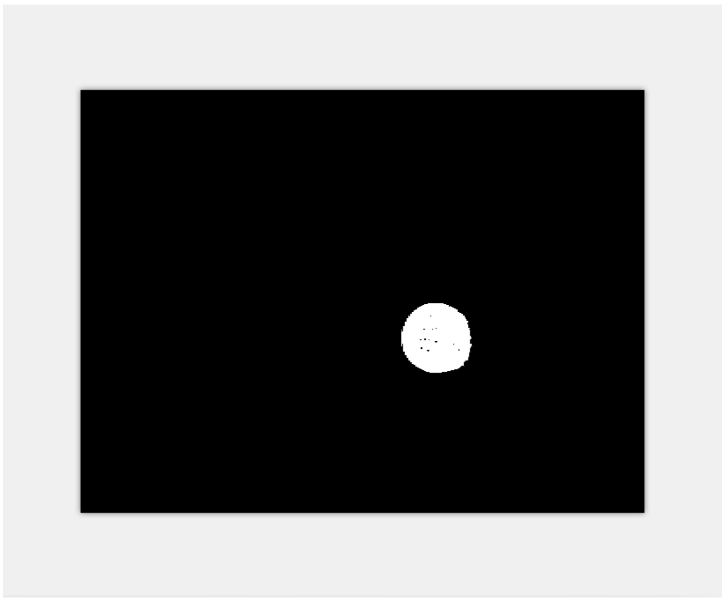


Figure 28. Capturing the red ball

3.6 Filter the Noise

First, the Gaussian noise needs to be remove. And in this case, it was done with the function, cv2.GaussianBlur() and a Gaussian kernel was used. In addition, the Gaussian kernel acted as the low pass filter, so it can remove the high frequency noise. The width and height of the kernel was set to be 5*5. The code is as follows:

```
blur = cv2.GaussianBlur(mask,(5,5),0)
```

The variable mask is the input image.

Then there were still black holes in the area of the ball, if the black holes were not connected, then it would cause a problem in finding the edge of the circle. So the function, dilate(), was used to connect the black holes. But that function can exaggerate the shape of the circle so the function, erode(), was used to restore the shape of the circle. The code is as follows:

```
kernel = np.ones((5,5),np.uint8)
dilation = cv2.dilate(blur,kernel,iterations = 3)
blur = cv2.erode(dilation,kernel,iterations = 3)
```

The result is as shown in Figure 29. There are not black holes in the circle in the image.

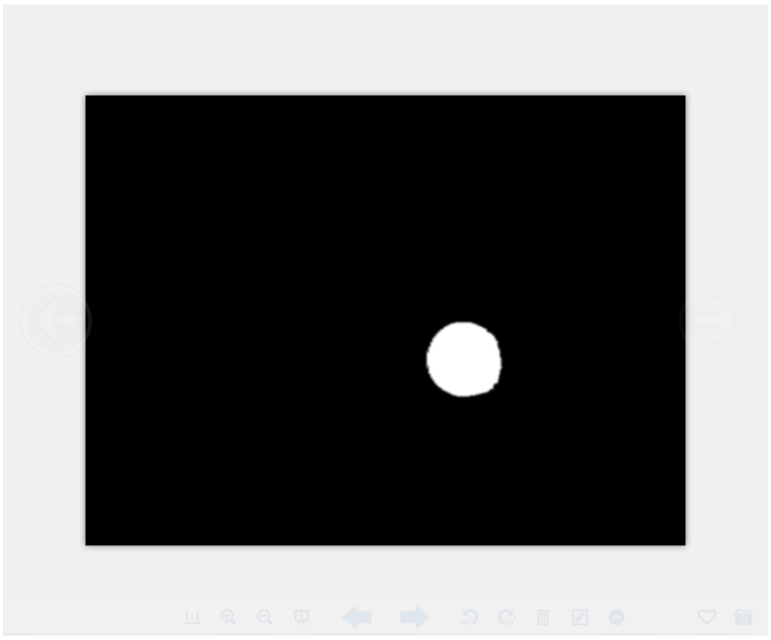


Figure 29. Image after filtering noise

3.7 Finding the Center of the Circle

3.7.1 Canny Edge Detection

First, the Canny edge detection was used to detect the edge of the circle, since it will facilitate finding a more accurate center of the circle. The Canny edge detection is a very popular edge detection algorithm. In addition, this algorithm is a multi-stage algorithm and the first stage of this algorithm is reducing the noise. A 5*5 Gaussian filter was used in that stage. Then the edge gradient and direction for each pixel in the image was calculated. Then the next stage was to remove any useless pixels which may not constitute the edge. Finally, the last stage was to decide the real edge of the image. In that stage, two threshold values which are minVal and maxVal were needed. Any edges with an intensity gradient more than the maxVal are definitely the edges and those with intensity gradient less than the minVal are sure to be non-edge. But those which are larger than the minVal and smaller than the maxVal are classified edges or non-edges based on their connectivity. If they are connected to "edge" pixel, they are considered to be edges, otherwise they are non-edges. As shown in the Figure 30, point A and B are edges and point C and D are non-edges. /19/

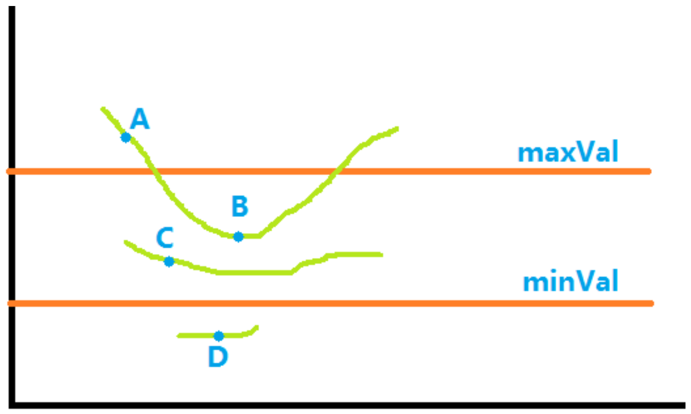


Figure 30. Canny edge detection

But in Opencv, all those stages are done by one method, cv2.Canny(). As shown in the code, the first argument is the input image and the second and the third arguments are minVal and maxVal respectively. The results is as shown Figure 31. It can be seen that the edges are detected perfectly.

```
edges = cv2.Canny(blur,50,300)
```

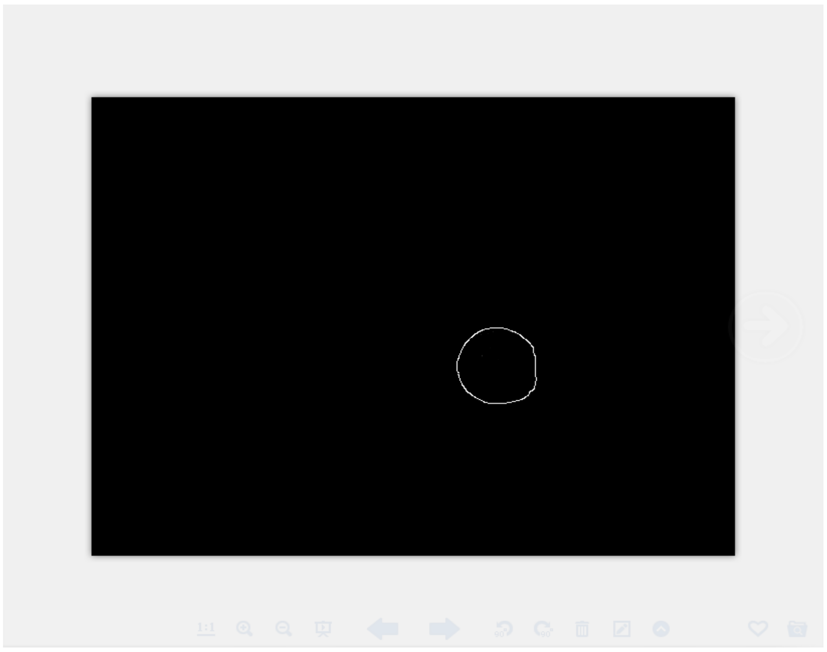


Figure 31. Edge of the circle

3.7.2 Hough Circle Transform

After detecting the edge of the circle, the Hough circle transform was used to detect the center of the circle. First, the theory of the Hough transform was introduced.

In polar coordinate, any circle and be expressed as:

$$x = x_0 + r\cos\theta \tag{9}$$

$$y = y_0 + r\sin\theta \tag{10}$$

x and y are the coordinate of any point at the circle, and x_0 and y_0 are the center of the circle. The r is the radius of the circle. In Hough circle transform, first, the center of the circle and the radius of the circle are assumed to be the given quantity. And with the angle θ changing from 0 to 360, the edge or the circle will be obtained. Therefore, inversely, if the coordinate of the each point on the circle and the radius of the circle are known, the coordinate of the center of the circle will be obtained with the angle θ changing from 0 to 360. And in this case, the edge of the circle was known, so the center of the circle can be obtained by using this method. But this algorithm requires too much calculation which will decrease the efficiency of the code. Therefore, in OpenCV, Hough Gradient Method is used to detect the center of the circle, which makes the algorithm easier.

Therefore, in OpenCV, all the algorithms were done by the method, cv2.HoughCircles(). And the code is as follows:

```
circles=cv2.HoughCircles(edges,cv2.cv.CV_HOUGH_GRADIENT,1,25,
        param1=55,param2=25,minRadius=10,maxRadius=0)
```

The first argument of this method is the input image and the second argument is the method to be used to launch the Hough circle transform. In addition, the third argument is ratio of the image resolution to the accumulator resolution .For example, if its value is 1, the accumulator has the same resolution as the input image. Then the forth argument is the minimum distance between the centers of the detected circles. Since there might be many circles in an image, so if this argument is too large, many circles will be missed. The fifth argument is the higher threshold in the Canny method and the sixth argument is the accumulator threshold for the circle centers at the detection stage. If it is too small, the more false circles may be detected. Eventually, the last two arguments are the minimum radius value of the circle and the maximum radius value of the circle. Since the only limitation of this circle is that the Hough circle transform cannot find the precise radius of the circle, there is a need to specify the estimated radius of the circle in this method. But the Hough circle transform can find the center of the circle precisely even it is not a perfect circle because of shadow of the ball. It will return the x and y coordinates of the circle in pixel. The result is as shown in Figure 32. In this case, the coordinate of the center found is [407,282].

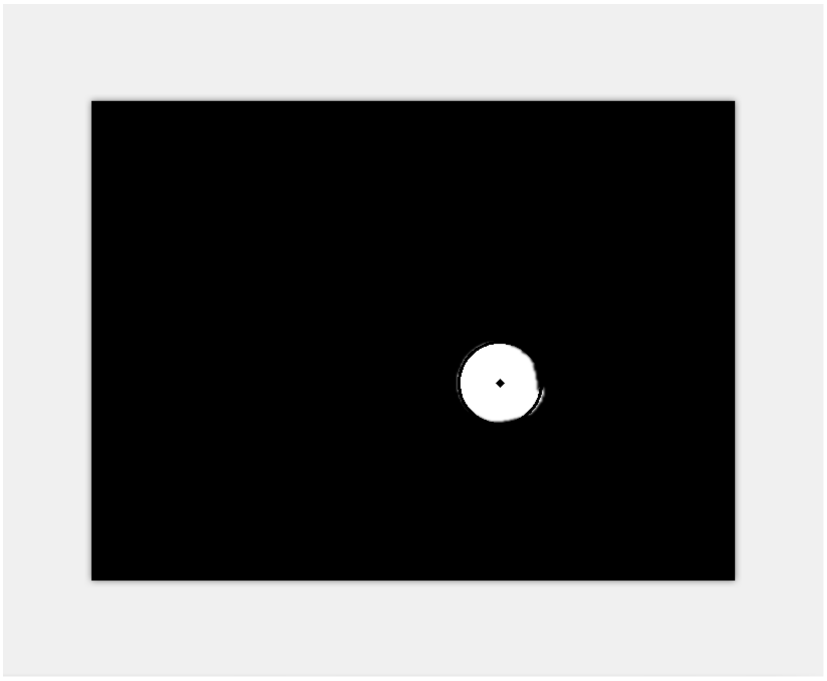


Figure 32. Center of the circle

4 STRATEGY DESIGN

This chapter will introduce finding the ball strategy, picking up the ball strategy and finding the landmark strategy.

4.1 Calculate the real distance of the ball

In this case, the module, ALRedBallTracker, was used to make Nao track the red ball. The main goal of this module is to build a connection between the red ball and motion in order to make Nao keep the red ball in view in the middle of the camera. When tracking the ball, the stiffness of the head is set to be one. Then the head will move with the movement of the red ball. In addition, the method getPosition() was used to return the [x,y,z] position in FRAME_TORSO which is the relative [x,y,z] position based on the position of robot. But this method was done assuming that an average red ball size is 6 cm. So the [x,y,z] position is not accurate. In this case, only the x value was used to calculate the real distance between the ball and the robot’s feet in x axel direction. Figure 33 shows x,y and z position in FRAME_TORSO. /20/

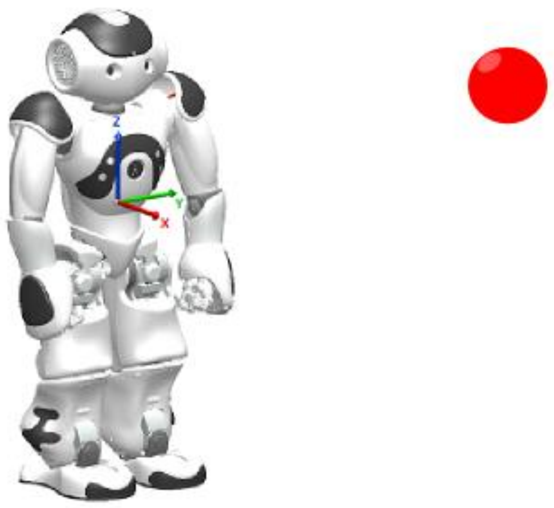


Figure 33. x, y and z position in FRAME_TORSO/20/

So the robot was kept in the go-initial posture and the ball in the center of the robot’s view. There is a ruler in the x axel direction in the center line of the robot’s feet. The relative position of ball and the robot is as shown in Figure 34.

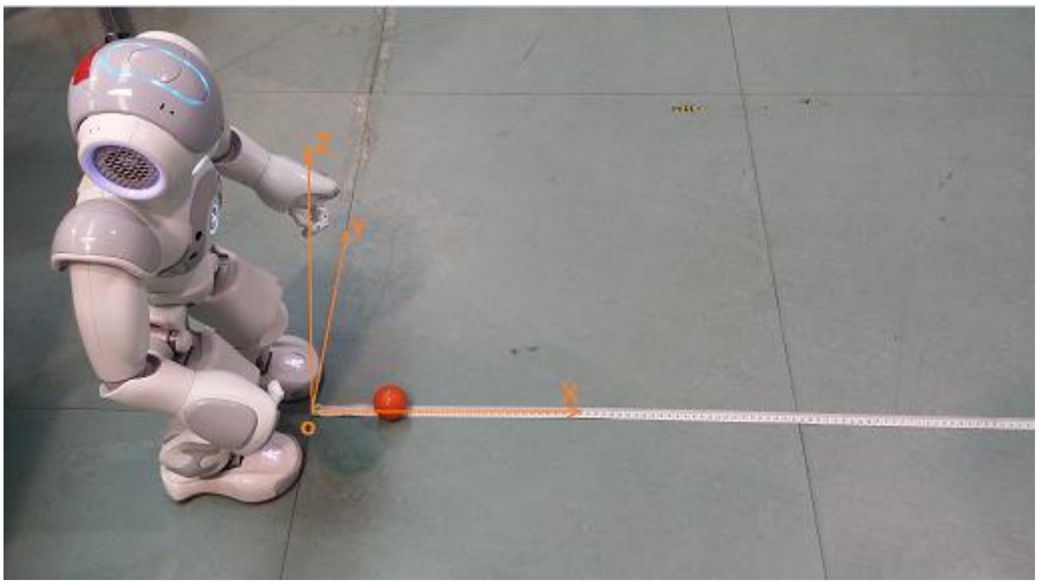


Figure 34. Find the real distance strategy/20/

The relationship between the real distance and the ball was found. First, the ball was placed in the point o shown in Figure 34**,** where the distance between the ball and the robot’s feet is almost 0. Then the code was ran. The robot started tracking the ball and got the x value of the ball which is 0.163 in this case. Then the ball was placed in the point where the distance between the ball and the robot’s feet in x axel was 10 centimeters. The robot started tracking the ball without any body movement. The x value of the ball which is 0.355 was obtained. Then the distance was increased in every 10 centimeters. But when the distance was larger than 50 centimeters, the robot could not keep the ball in the center of its view. Therefore the x value was not accurate and those x values were discarded. So in a real situation, if the robot finds that the distance between the ball and its feet is larger than 50 centimeters, it will firstly move 40 centimeters in x axel direction and then it will track the ball again to find the precise location of the ball in x axel direction.

Six pairs of data was found. A piecewise linear function was used to express that relationship between the real distance and the robot’s feet. The 6 pairs of data are shown in Table 2 and the piecewise linear function is as shown in Figure 35.

Table 2. Real distance and X value

X value (x)	Real distance in meter(y)
0.163	0
0.355	0.1
0.609	0.2
0.829	0.3
1.092	0.4
1.585	0.5

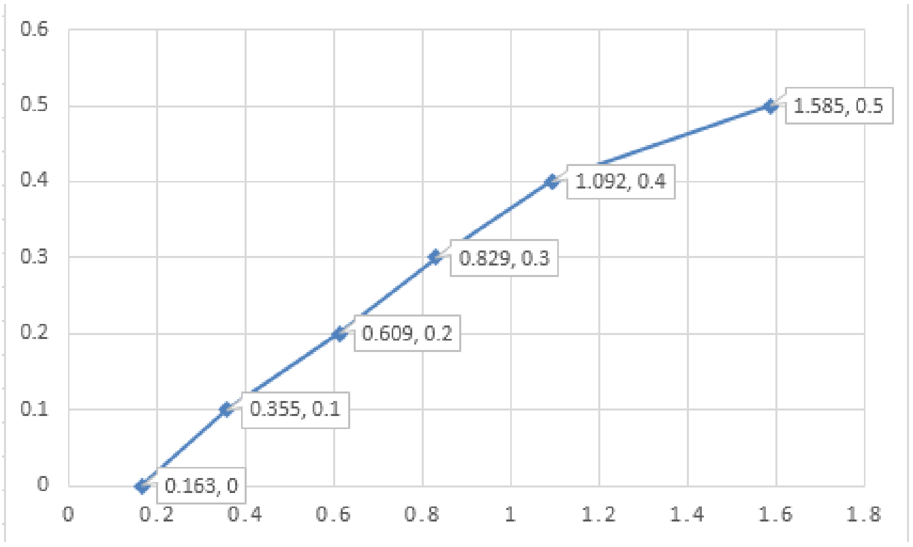


Figure 35. Piecewise function for real distance and the x value.

Then the expression of each piecewise function needed to be found. First, the expression for a linear function is as follows:

$$y = kx + b \tag{11}$$

There are two points on that line which are (x_1, y_1) and (x_2, y_2). So the coordinates of both two points can be substituted into that function. Then the expressions obtained is as follows:

$$\begin{cases} y_1 = kx_1 + b \\ y_2 = kx_2 + b \end{cases} \tag{12}$$

Then the expression for k and b can be found which is:

$$k = \frac{y_2 - y_1}{x_2 - x_1} \tag{13}$$

$$b = \frac{y_1 x_2 - y_2 x_1}{x_2 - x_1} \tag{14}$$

The x value obtained by the robot was set to be x and the real distance was set to be y. So by substituting those six pairs of data into the expression k and b the piecewise function was found. And it is as follows:

$$\begin{cases} y = 0.521x - 0.0849 & (0.163 < x \leq 0.355) \\ y = 0.394x - 0.0398 & (0.355 < x \leq 0.609) \\ y = 0.455x - 0.0768 & (0.609 < x \leq 0.829) \\ y = 0.380x - 0.0152 & (0.829 < x \leq 1.092) \\ y = 0.203x + 0.178 & (1.092 < x \leq 1.585) \\ 0.4 & (x \geq 1.585) \end{cases} \tag{15}$$

Therefore, while the x value was obtained, the real distance could be calculated by using this piecewise function. Based on the test this real distance was found to be accurate in this case.

4.2 Tracking the Ball Strategy

Sometimes, the ball is not in the center line of the robot's feet in the x axel direction, so only finding the real distance between the ball and the robot's feet is not enough. Tracking the ball strategy in this case is that when the robot finds the ball, it will turn its body to let it face the ball directly and then find the real distance in the x axel direction.

First, the position of the ball is assumed to be in the Ball position 1 in Figure 36. And the left foot and right foot position of the robot are assumed to be in Left foot center1 and right foot center 1 as in Figure 36. In this situation, the robot is facing the ball directly. But if the position of the ball changes to be in the Ball position 2 as shown in Figure 36 and if the position of the robot keeps that place, the head of the robot will turn right. The angle the head turns is angle b as shown in Figure 36. And if the robot wants to face the ball directly, the body of the robot needs to turn right and the angle is b. Since B_1O is perpendicular to L_1R_1 and B_2O is perpendicular to L_2R_2, angle a is equal to angle b. So every time when the robot finds the ball and the head will turn in an angle, and the body of the robot will turn in the same angle as the head. Then the robot is facing the ball directly. Then the robot can move only in the x axel direction until it reaches the specific point.

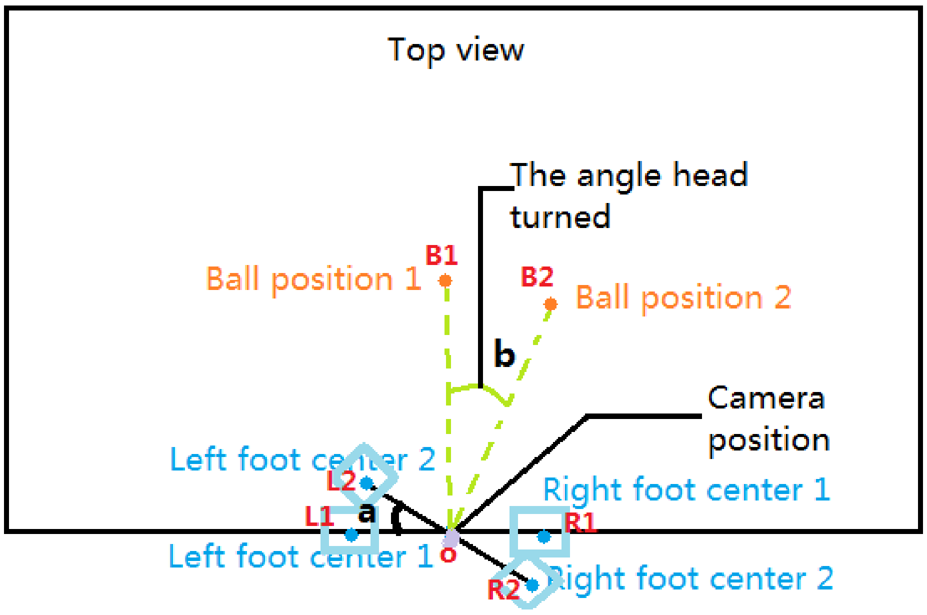


Figure 36. Facing the ball directly

Since the posture of picking up the ball is almost fixed and the length of the robot's arm is also fixed, so there is a relative position between the robot and the ball, where the robot can pick up the ball successfully. This is shown in Figure 37. When the left foot and right foot are in the position of L1 and R1, and the position of the ball is at the point p. Then the robot can pick up the ball. Based on the measurement, the length of the line OP is 5.5cm. The relative position of the robot and the ball is kept in that way, and let the robot to track the ball with its head. Then the angle that the head turned is 33.5˚, the angle α in Figure 37.

Furthermore, in this case, according to my tracking the ball strategy, the robot will always face the ball directly. So in this case, the original position of the robot in Figure 37 is on the line LR. Besides, in order to pick up the ball successfully, the robot must turn its body to move to the position where the line L1R1 lies. As mentioned above, the angle that the body need to turn equals to the value of angle αwhich is 33.5°. Since $\alpha + \beta = 90°$, so $\beta = 56.5°$. Therefore:

$$CP = \frac{OP}{\sin \beta} = 6.59 \tag{16}$$

So, the strategy is that after finding the ball, the distance of the robot and the ball is kept to be 6.59 cm. Then the robot will turn left and the angle that it turn is 33.5°. Then the robot will reach a specific place where the robot is able to pick up the ball.

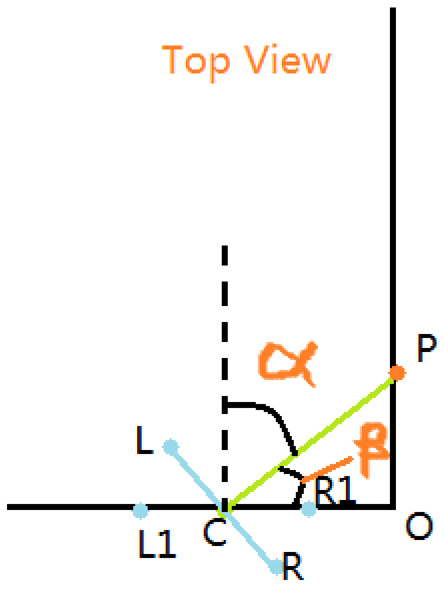


Figure 37. The specific position to pick up the ball

But in a real situation, the distance that the robot moves is not precise and sometimes the robot will move the distance where it is longer than the value set for the robot. So the robot will track the ball twice. In the first round, the robot will reach the point where the distance to the ball is 9.5 cm, since in the first round if the distance is set to be too small, the robot will always kick the ball. But in the second round, the robot will reach the point where the distance to the ball is 6.59 cm. Then the robot will turn its body to reach the position where the robot is able to pick up the ball. Figure 38 shows the flow chart for tracking the ball.

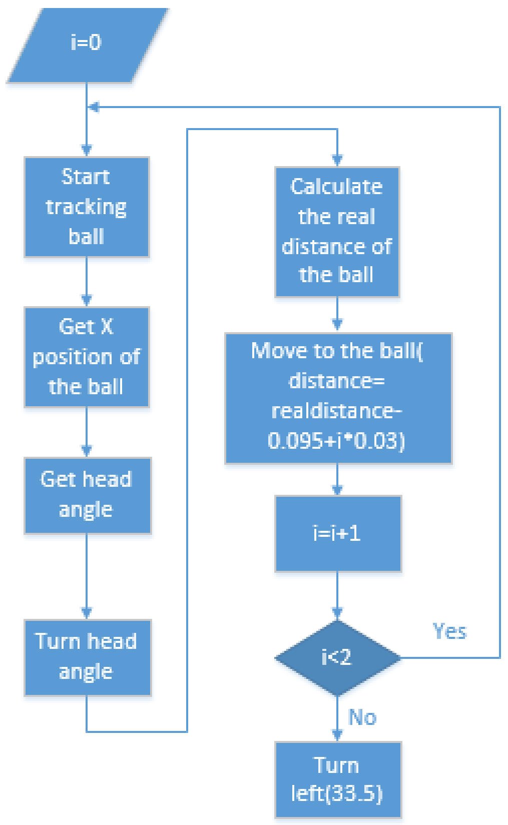

i=0
Start tracking ball
Get X position of the ball
Get head angle
Turn head angle
Calculate the real distance of the ball
Move to the ball(distance= realdistance-0.095+i*0.03)
i=i+1
i<2
Yes
No
Turn left(33.5)

Figure 38. Tracking ball flow chart

4.3 Picking up the Ball Strategy

Although the robot reaches the specific point where it is possible for the robot to pick up the ball, the rate that the robot can pick up the ball successfully is still very low. So in order to increase the rate of picking up the ball, a picking up the ball strategy was designed.

Three different picking up the ball animation were designed initially to pick up the ball in a different area. The area that the robot can pick up the ball for each animation was found as shown in Figure 39 based on many times testing. In Figure 39, those points with three different color is the place where the red ball appears. And the area that the red points appears is the range that the robot can pick up the ball using animation 1. The area that the green points appears is the range that the robot can pick up the ball using animation 2. And the area that the purple points appears is the range that the robot can pick up the ball using animation 3. Therefore, the range that the robot can pick up the ball for each animation is like a parallelogram. The threshold is one side of that parallelogram which is the green line in Figure 39.

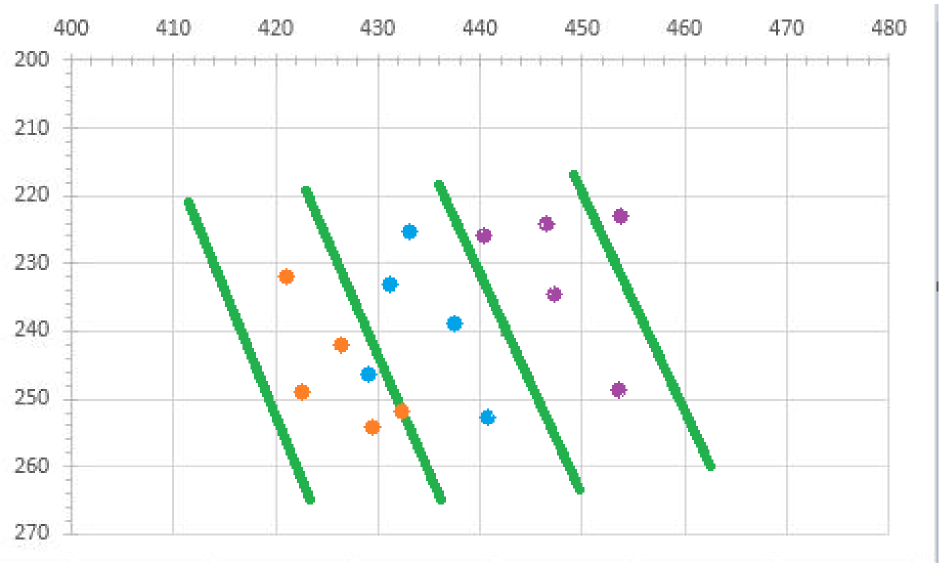


Figure 39. Threshold for each animation

Then the area that the ball will appear in the robot’s view with high possibility was tested. The area is as shown in Figure 40. These red points are the center of the ball. The coordinate of center of the ball in pixel was already known, and the robot tracks the ball for many times and finds many pairs of coordinates of the center of the ball. Then according to the coordinates, the area that the ball will appear with high frequency as shown in Figure 40.

Based on testing, the area that the ball appears is quite small in the robot’s view. Therefore, two animation are enough to increase the rate of picking up ball to be 80%. Then there is a need to set the threshold to decide to use which animation. One side of that parallelogram is the threshold as shown in Figure 41. Then green line passing through the point P1 and P2 is the threshold line. Since using a linear function to be the threshold will increase the calculation of the program, the x coordinate of the point which is the intersection of the threshold line and the x axel was set to be the threshold. The area that is on the left of the threshold line as shown in Figure 41 is the range that the robot can pick up the ball in using animation 1. Therefore, the right area of the threshold line is for animation 2.

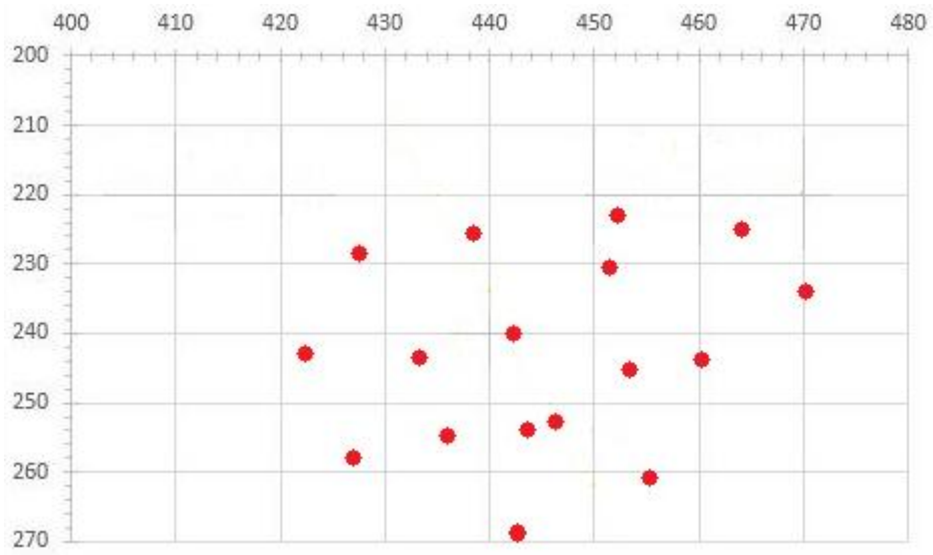


Figure 40. The area that the ball appears

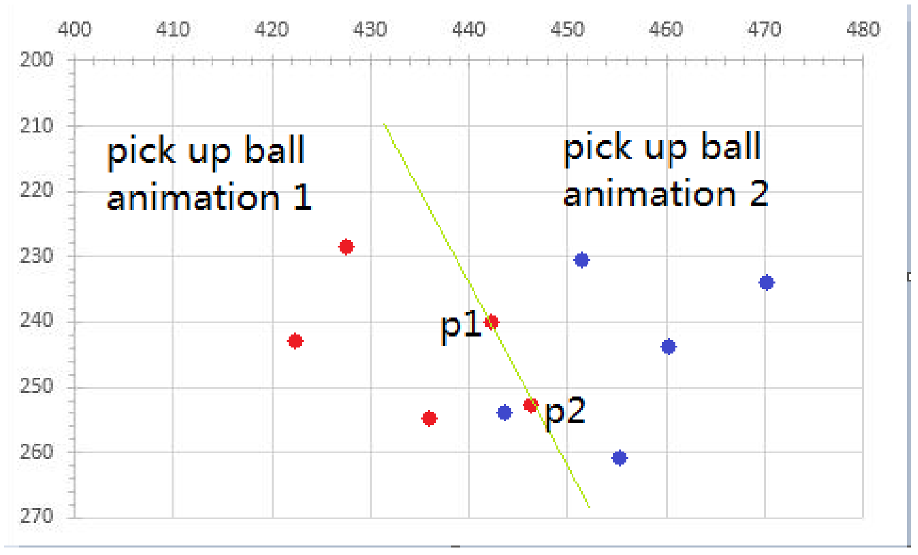


Figure 41. Threshold for deciding which animation to use

The coordinates of the point p1 and p2 are [442,240] and [446,252]. Therefore the linear function of the threshold line is:

$$y = 3x - 1086 \tag{17}$$

And the inverse function of this linear function is:

$$x = \frac{1}{3}y + 362 \tag{18}$$

The x coordinate of the intersection point is the vertical intercept of this inverse linear function. And it is 362. Therefore this is the threshold.

In a real situation, after the robot has found the coordinate of the center of the ball in pixel, through that point a parallel line of the threshold line can be found. The x coordinate of the intersection of that line and the x axel can be calculated. In this case, the coordinate of the center of the ball in pixel was assumed to be [a, b]. Since the line through that point is parallel to the threshold line, the slope of the inverse linear function of that line is the same as the slope of the inverse linear function of the threshold line and it is $\frac{1}{3}$. So, the inverse function of that line can be expressed as:

$$x = \frac{1}{3}y + n \tag{19}$$

n is the x coordinate of that intersection. Since that point is on this line, therefore:

$$a = \frac{1}{3}b + n \tag{20}$$

So:

$$n = a - \frac{1}{3}b \tag{21}$$

Therefore, if the value of $a - \frac{1}{3}b$ is larger than 362, the animation 2 will be chosen to pick up the ball. Otherwise, animation 1 will be used to pick up the ball.

4.4 Finding the Location of the Box

4.4.1 Definition of the Box and Nao Mark

In this case, the box used is as shown in Figure 42. The depth of the box is 25 cm. There is a Nao mark on the box and it is used to find the location of the box. See Figure 43.

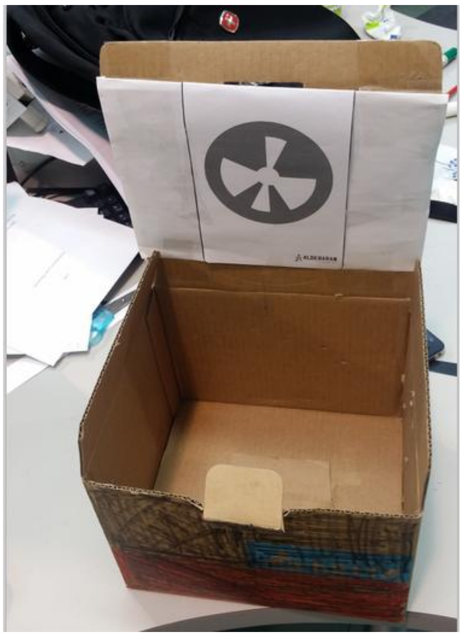


Figure 42. The box

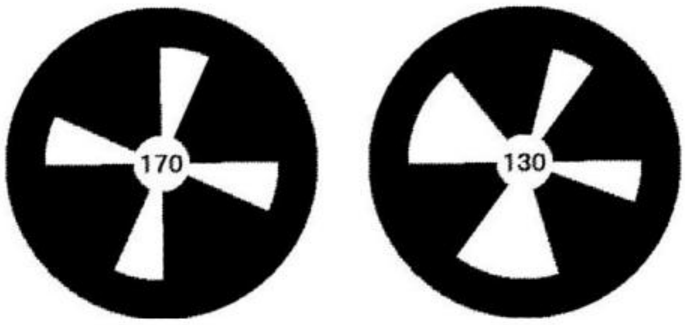


Figure 43. Nao marks/21/

As shown each Nao mark consists of black circles with white triangle fans centered at the circle's center. The particular location of the different triangle fans is used to distinguish different Nao marks. Each Nao mark has its own ID number. And in this case, the Nao mark used was the number 114 Nao mark, as shown in Figure 44. /21/

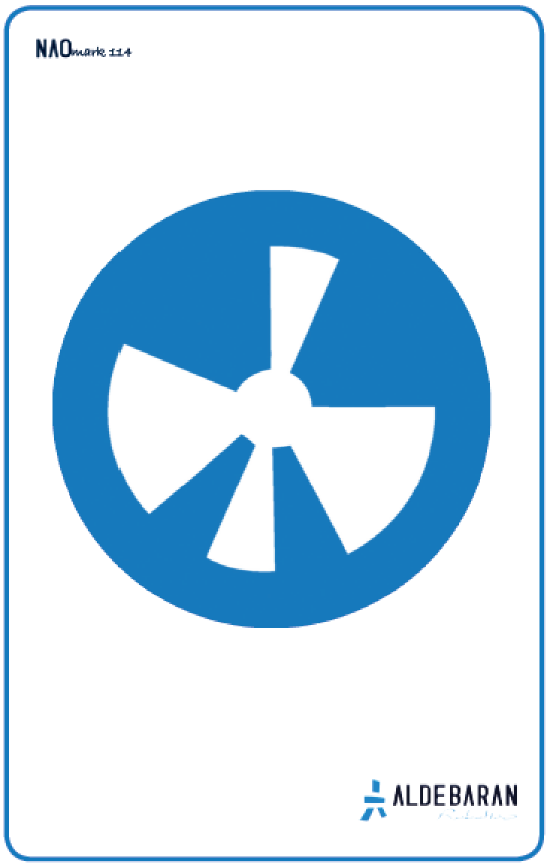


Figure 44. Number 114 Nao mark/22/

4.4.2 Finding Nao Mark Strategy

In this case, the module, ALLandMarkDetection, was used to recognize the Nao mark and get the size of the Nao mark and the module, ALMemory, was used to read and output the information of the Nao mark which was obtained from the AL-LandMarkDetection module.

First, a proxy to the module, ALLandMarkDetection, was created. By subscribing to the ALLandMarkDetection proxy, the module will write in ALMemory. Then the robot starts finding the Nao mark. By using the method, getData(), of the module ALMemory, size X and the angle of the Nao mark will be obtained. Based on testing, the angle of the Nao mark divides 2, and it is the angle that the robot needs to turn its body. After the robot has turned its body, the robot will face the Nao mark directly. Then the robot only needs to walk to a distance in the x axel direction to reach the place of the box.

The size X is the size of the Nao mark in robot's camera. The relationship between the size of the Nao mark and real distance to the robot can be found.

The relationship was tested as shown in Figure 45. First, the posture of the robot was set to be the go-initial posture. There was a ruler on the center line of the robot's feet in the x axel direction. The Nao mark was placed at the point where the distance to the robot is 30 cm since if the distance is less than 30cm, the robot cannot recognize the Nao mark. Then the robot was programmed to find the size x of the box. Then those value was recorded. Then the distance was increase in every 10 centimeters. Then the corresponding size X was obtained. Until the distance reached 150 centimeters, the robot could recognize the Nao mark. Table 3 shows the data obtained.

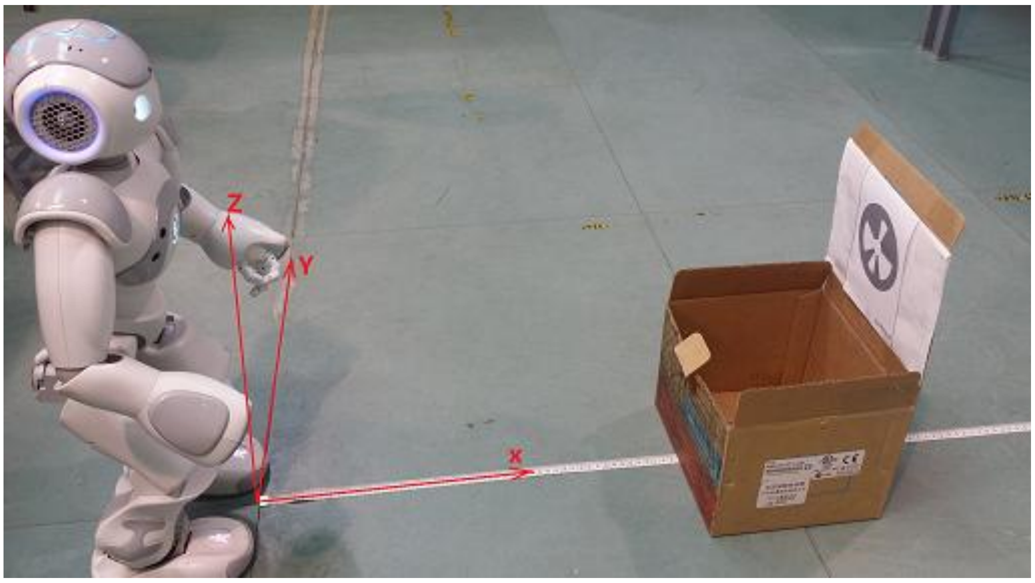


Figure 45. Measuring the Nao mark's distance (side view)

Table 3. Size X and the distance

Size X	Distance (meter)
0.228	0.3
0.188	0.4
0.158	0.5
0.138	0.6

0.121	0.7
0.108	0.8
0.098	0.9
0.091	1.0
0.085	1.1
0.078	1.2
0.073	1.3
0.070	1.4
0.065	1.5

Using these data in Matlab, the relationship of the real distance and size X was plotted. The result is as shown in Figure 46, where the x axel is the size X and the y axel is the real distance.

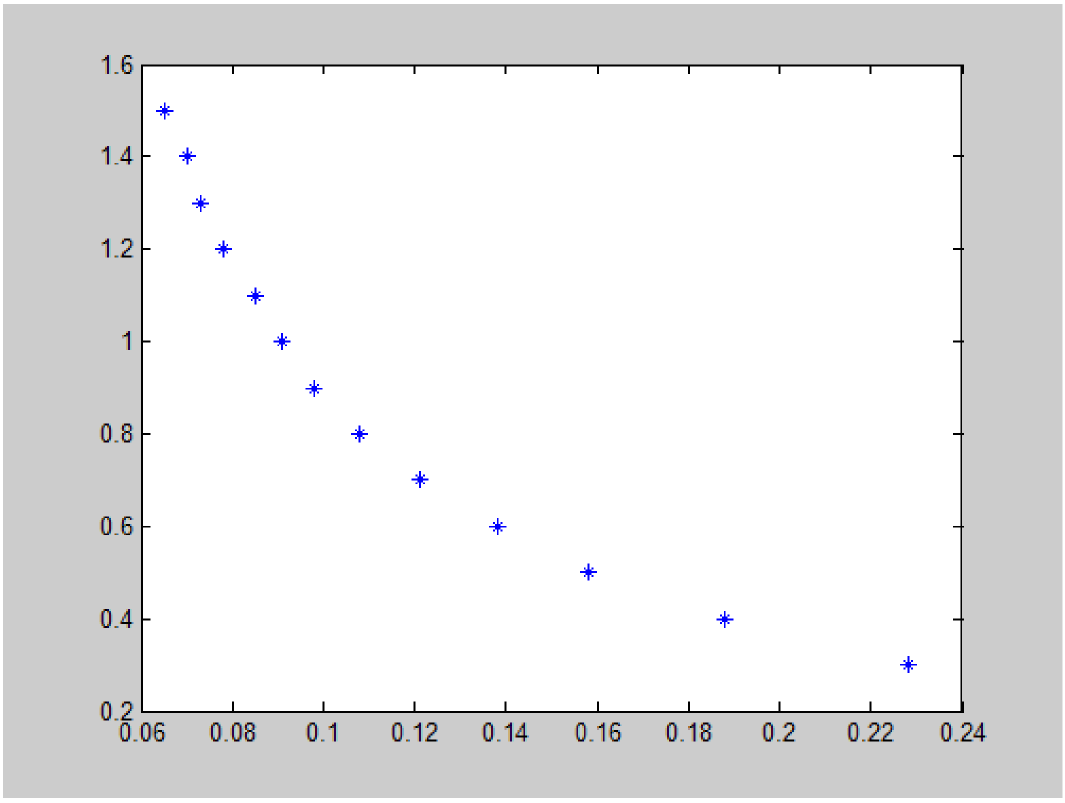


Figure 46. Relationship of the real distance and size X

As shown in Figure 46, first, if those points are connected, the curve in Figure 46 tends to be similar as the curve of the exponential function. But since the relationship of the real distance and size X is not known, if only exponential functions are used, the curve of that expression will not fit the curve perfectly. Furthermore, according to the Taylor's theorem, any functions can be expressed by the polynomial functions known as Taylor polynomial, therefore even the exponential function can be expressed by the polynomial functions. So in general, if the expression of the curve is not known, using the polynomial functions is the best solution to reduce errors. And in Matlab, the polyfit function can be used to find the best polynomial functions for any curve. And the degree of the polynomial functions can be changed to find the best expression for any curve. And in this case, based on testing, when the degree of that function is three, it can expressed the curve perfectly. In addition, it can make the codes efficient instead of involving too much calculation if the exponential functions are used. And the expression of that cubic polynomial functions is:

$$y = -485.5931x^3 + 266.2636x^2 - 50.8341x + 3.7969 \tag{22}$$

The curves of the both original relationship and the fit function was plotted in the same figure. The result is as shown in Figure 47. The blue crosses are the original data obtained and the green curve is the curve of the fit function. As it can be seen, the fit function is perfect. The code for Matlab can be found in Appendix 2.

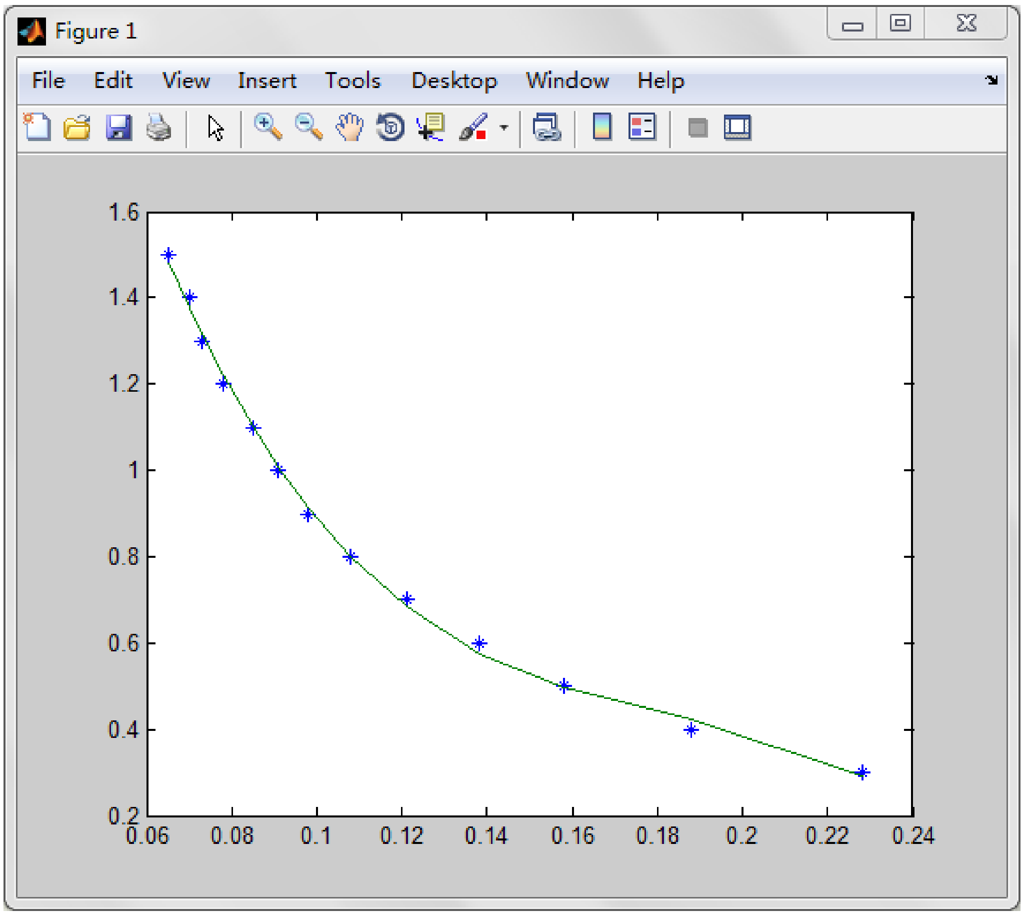


Figure 47. The fit function

In a real situation, after finding the size X of the Nao mark, the value of the size X will be substitute to the fit function. Then the real distance was calculated. Then the real distance that minuses 35 centimeters is the distance that the robot need to walk towards. Since the depth of the box is 25 centimeters, and in case of the robot kicking the box, the value was set to be 35 centimeters. Finally after the robot reaches the box, the robot will throwing the ball.

5 IMPLEMENTATION DETAILS

In this chapter, the environment configuration, animation design and some important functions as well as the trouble shooting are introduced.

5.1 Set the Environment

5.1.1 Setting the Environment for Eclipse and Python

Eclipse is an IDE which contains a basic workspace and an extensible plug-in system for customizing the environment. In addition, PyDev is a plugin that provides features such as code completion and code analysis. So with the combination of Eclipse and PyDev, it is easier to program and debug.

First, eclipse was opened, and in the menu screen, the help icon was clicked, a "Install New Software" was chosen. The view of "Install New Software" is as shown in Figure 48.

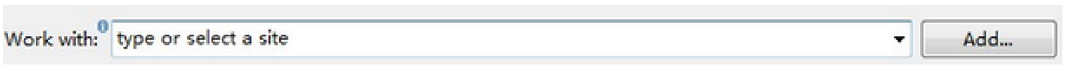
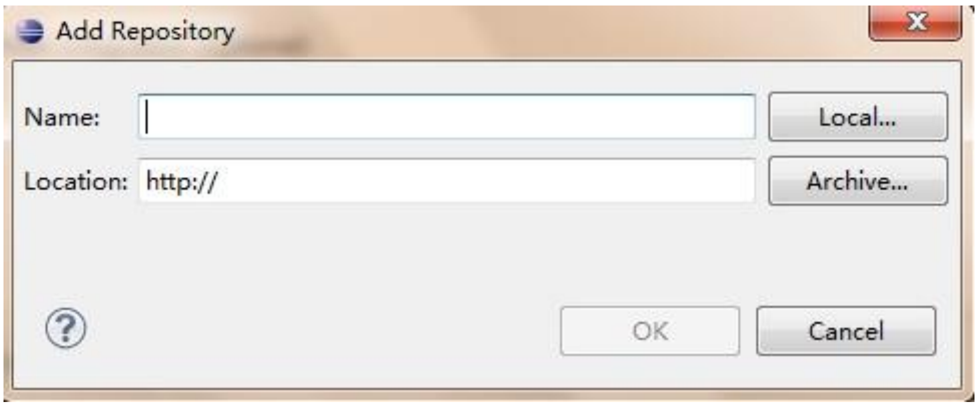


Figure 48. Install new software

By clicking the Add button, a view shown in Figure 49 appeared in the Name part, PyDev was input. And input the link for downloading the PyDev in the location part. Then ok was clicked. Finally the PyDev was installed automatically.


Add Repository

Name:

Location: http://

Figure 49. Add a new software

After installing the PyDev successfully, there was a need to configure the interpreter. Therefore, in the menu screen, Windows > Preferences>PyDev>Interpreter-Python was clicked. And the view was as shown in Figure 50. Then the Auto Config button was clicked. Then the interpreter was configured successfully.

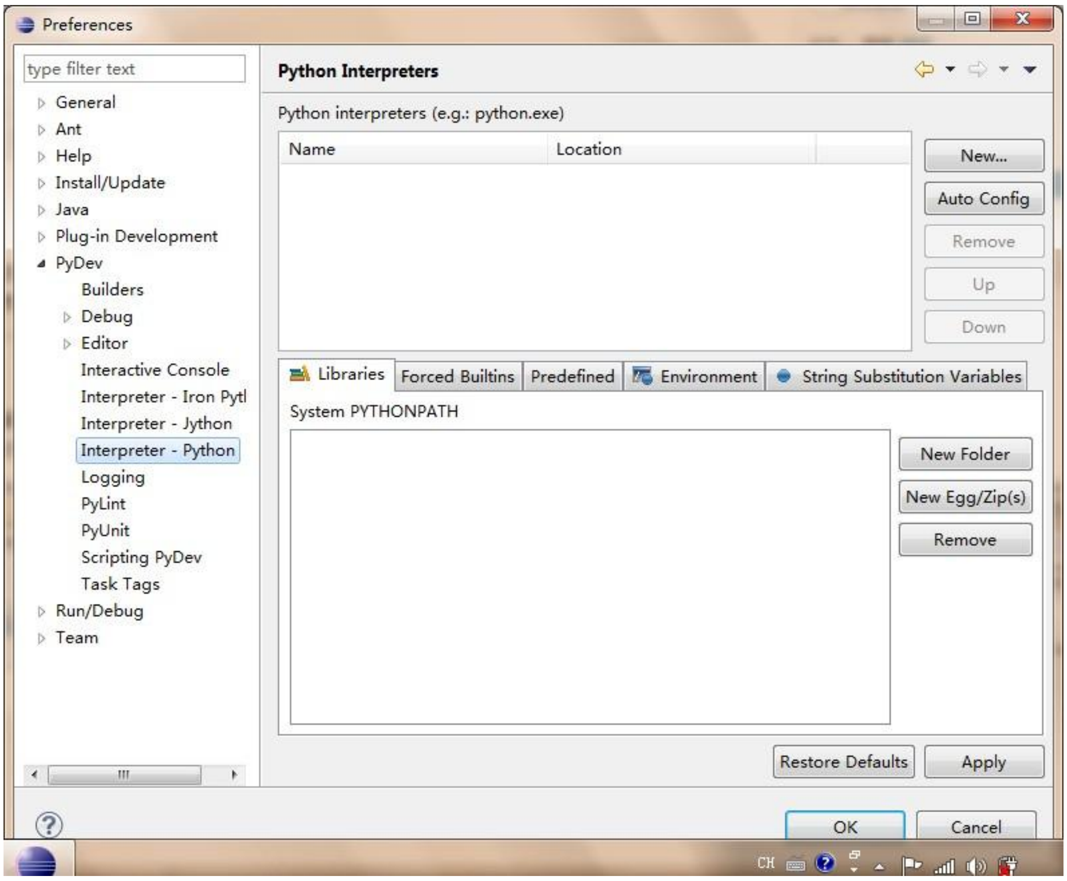


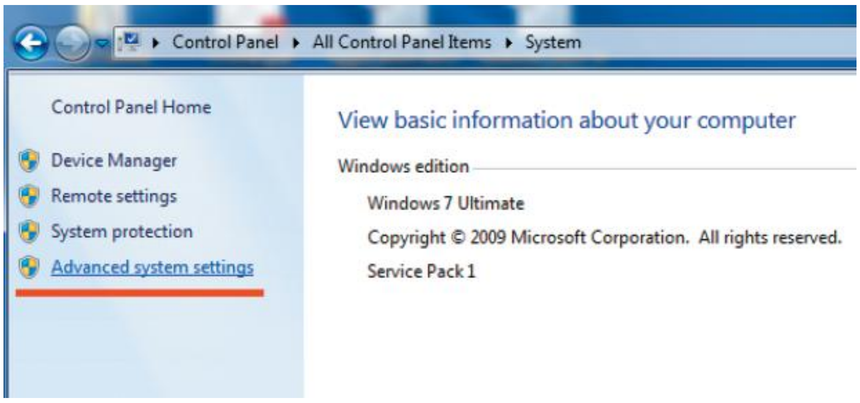
Figure 50. Configure the interpreter

5.1.2 Setting the Environment for OpenCV

First, the Python 2.7.3 was downloaded and installed. And since OpenCV only supports python 2.x, the version of the Python must be 2.x. Then OpenCV was downloaded and installed. Then the control panel was opened and the All Control Panel Items was clicked and then click the System. And the view was as shown in Figure 51. And the Advanced system settings on the left side of the System properties window was clicked. Then, the environment variable was clicked from the System Properties window. And the environment variable view was as shown in Figure 52. Then the path was clicked, and the paths of OpenCV and Python were saved into the Variable value. The paths were separated by using a semicolon. /3/

Then NumPy was downloaded and the version of it must be numpy-1.6.2-win32-superpack-python2.7 since only this version supports Python 2.7. Then SciPy was downloaded. The version of it must be scipy-0.11.0-win32-superpack-python2.7. And both of them were plugin which will be used in OpenCV.

Finally, the file “cv2.pyd” was found in the directory of OpenCV and this file was copied to the directory of Python. The path of that directory was “.\Program Files\Python27\Lib\site-packages”. Then the environment for OpenCV was built.



View basic information about your computer

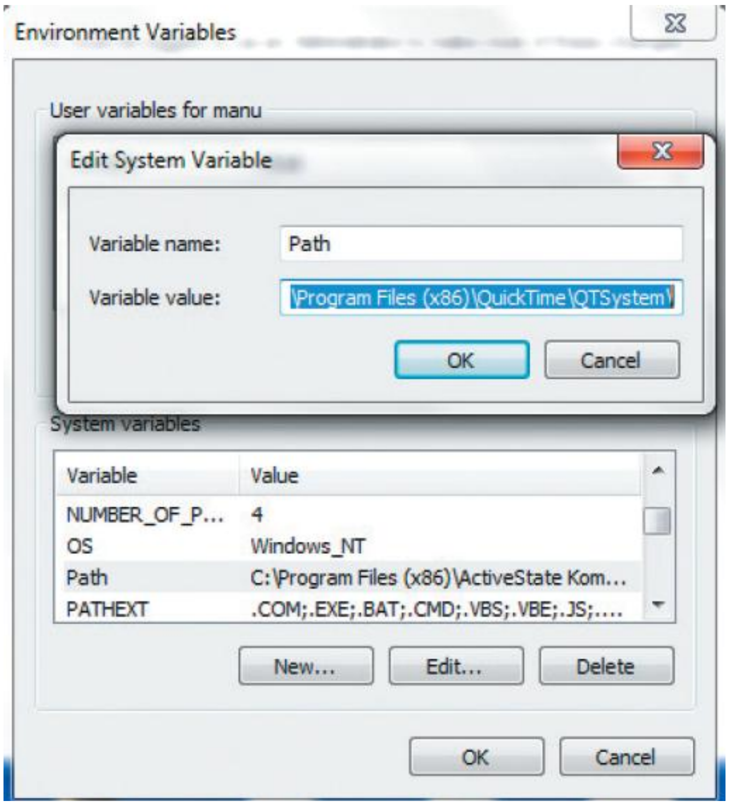
Windows edition

Windows 7 Ultimate

Copyright © 2009 Microsoft Corporation. All rights reserved.

Service Pack 1

Figure 51. Control panel



Variable name: Path

Variable value: \Program Files (x86)\QuickTime\QTSystem\

Variable	Value
NUMBER_OF_P...	4
OS	Windows_NT
Path	C:\Program Files (x86)\ActiveState Kom...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....

Figure 52. Set the environment variable

5.2 Animation Design

For the animation design, Choregraph was used to design those movement and the box timeline was used to create any animation. By double clicking the timeline, each animation could be created in each key frame. The timeline panel is as shown in Figure 53. The function of each part pf the timeline panel can be found in Table 4.

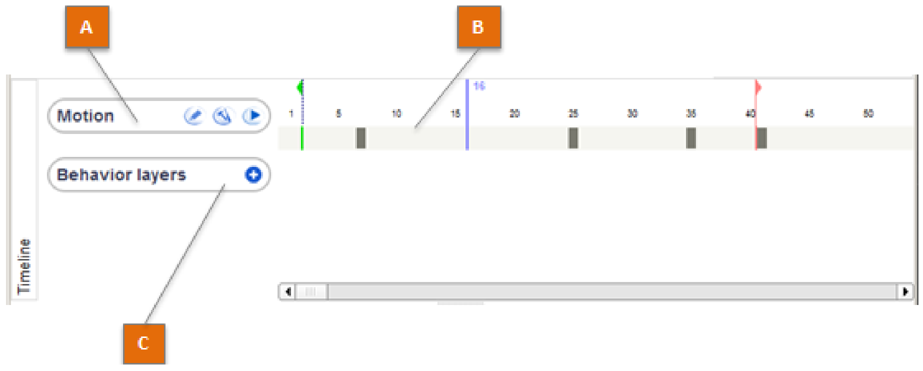


Figure 53. Timeline panel/23/

Table 4. Functions of the timeline

Part	Name	Description
A	Motion	It can be used to define the Motion key frames Timeline editor button can be used to edit the motion of each joints in more details Timeline properties can be used to define the value of frame per second. And in this case, it is set to be 10. Since in one second, the robot will move 10 frames, which will make the movement stable. Besides it can also be used to set the mode of resources acquisition. And normally this mode is set to be passive mode. Play motion will play the motion layer of the timeline
B	Time ruler	The posture of each key frame can be created in the time ruler by click the one frame on the time ruler. And when creating the movement by using creating the several key frames instead of

		creating all the frames of the movement, the robot will supplement the frames between each two key frames automatically.
C	Behavior layers	It can be used to define behavior layers to be executed in parallel with motion key frame/23/. ⊕ Add button can be used to add one or more behavior layers.

Furthermore, in this project, those designed animations are Nao movements that occur over time, and in Timeline each frame represents the posture of the Nao robot. Therefore by defining the a posture manually by opening the stiffness of each joints and storing the position of each joint of the robot in some key frames on the Time ruler, the robot will do the posture in each frame then the animation is achieved. And since the robot can automatically calculate the missing frames between those key frames, defining the posture of the robot in each frame is not needed, so only defining the postures of the robot on the key frames are enough.

5.2.1 Nao Squat Animation

For this Nao squat animation, there are total of six key frames. First, the robot was initialized as the go-initial posture. Then the robot will stretch its left leg. Then the robot will bent its right leg and also raise its right arm. The arm needs to be raised, since when the robot squats, the arm will block the sight of the robot’s camera, so the robot cannot find the ball. Then the head of the robot will turn right and look down onto the floor to find the precise location of the ball. The series of the images from Figure 54 to Figure 59 show every single key frame of this animation:

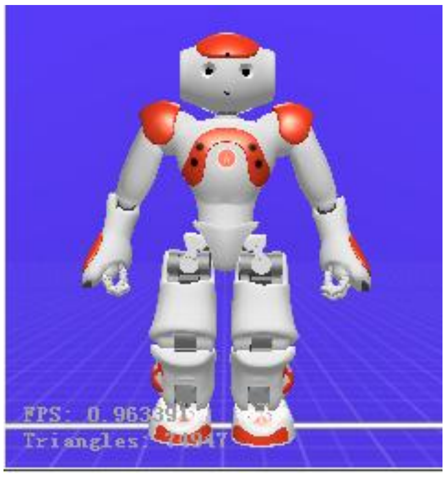


Figure 54. Key frame 10

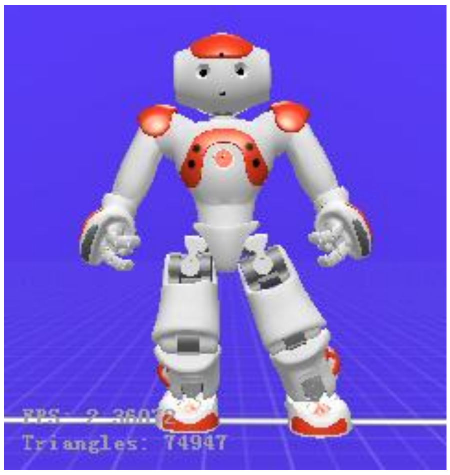


Figure 55. Key frame 30

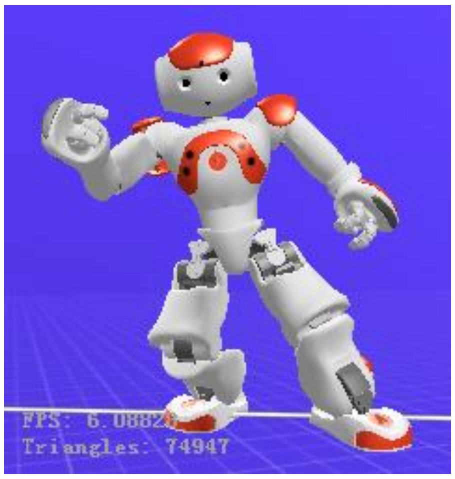


Figure 56. Key frame 55

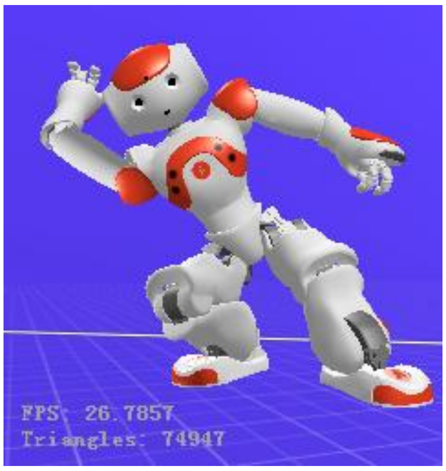


Figure 57. Key frame 87

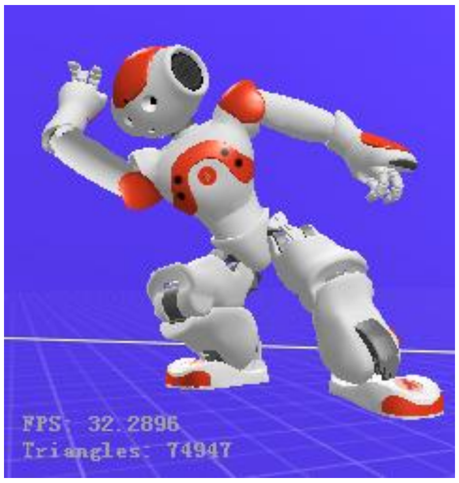


Figure 58. Key frame 105

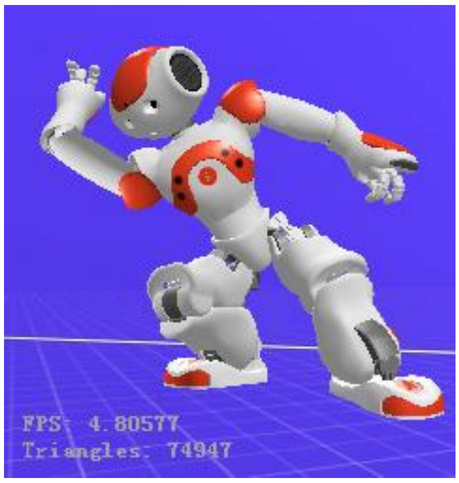


Figure 59. Key frame 120

5.2.2 Picking up the Ball Animation

There are two kinds of pick up ball animations. Animation 1 is used when the location of ball is on the left of the threshold line. Animation 2 is used when the location of the ball is on the right of the threshold line.

First, the robot is in the posture of squat, then the robot will lower its arm. Then the robot will open its hand and will spin its right wrist in counterclockwise direction until it touches the ball. In addition, the wrist of the robot will be spun, since it will increase the range that the robot can pick up the ball for each animation. Then the robot will close its hand and grasp the ball in its hand. Then the robot will stand up.

There are total seven key frames for each animation. The difference between the animation 1 and the animation 2 is that when the robot lowers its arm, the position of the right hand is deviation to the left for animation 1, but the position of the right hand is deviation to the right for animation 2. The series of the images from Figure 60 to Figure 66 shows every key frame of picking up the ball animation 1:

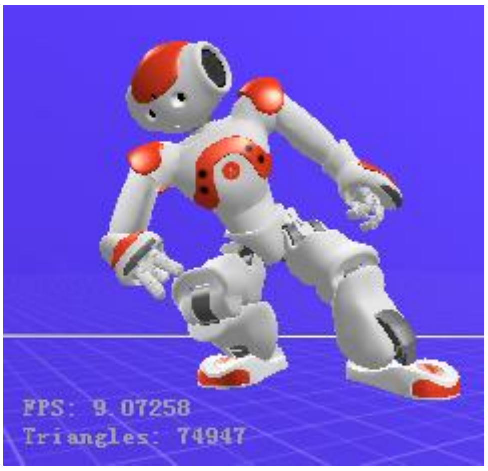


Figure 60. Key frame 30

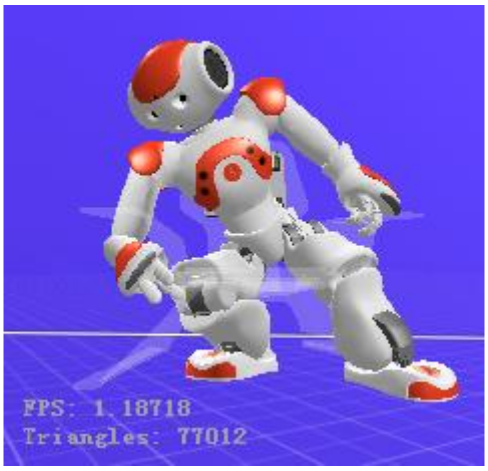


Figure 61. Key frame 40

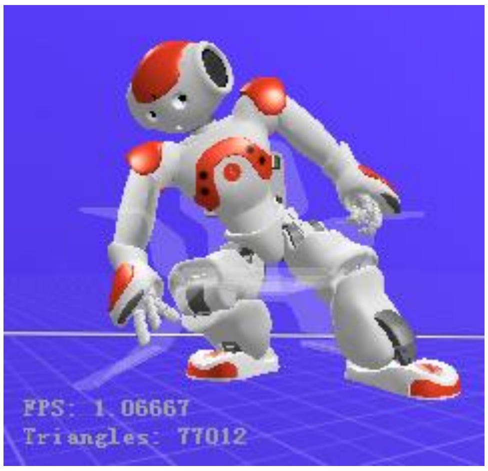


Figure 62. Key frame 50

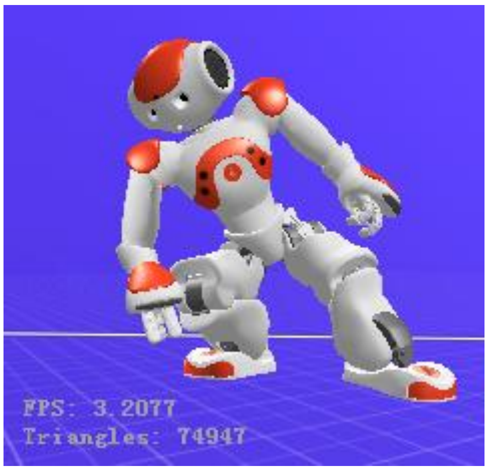


Figure 63. Key frame 70

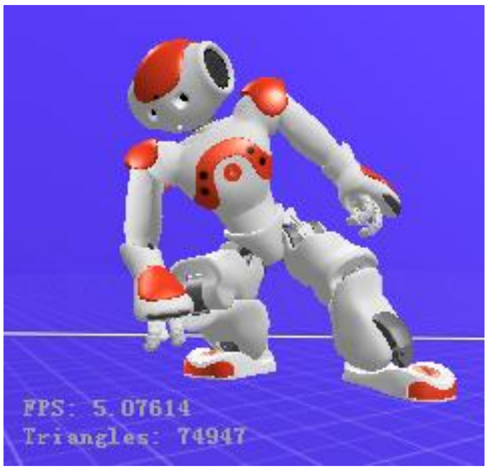


Figure 64. Key frame 90

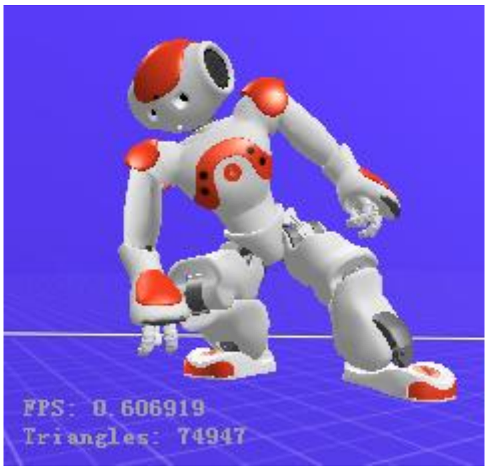


Figure 65. Key frame 110

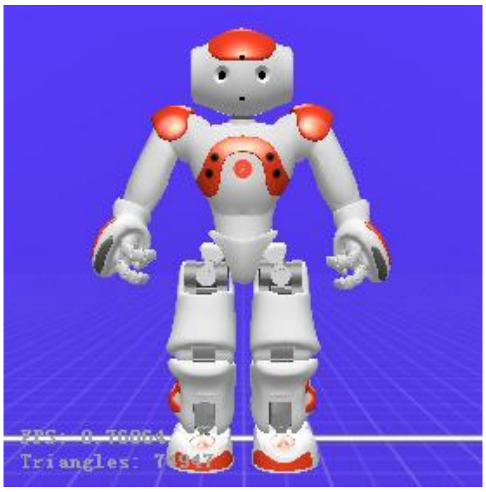


Figure 66. Key frame 175

The series of the images from Figure 67 to Figure 73 show every single key frame of picking up the ball animation 2:

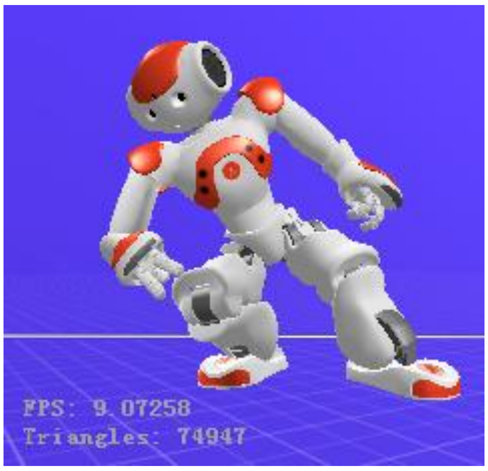


Figure 67. Key frame 30 (2)

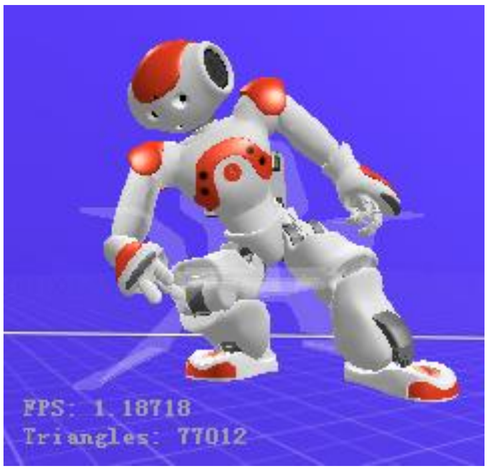


Figure 68. Key frame 40 (2)

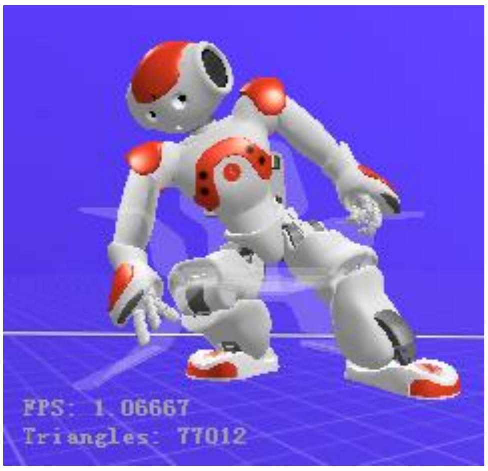


Figure 69. Key frame 50 (2)

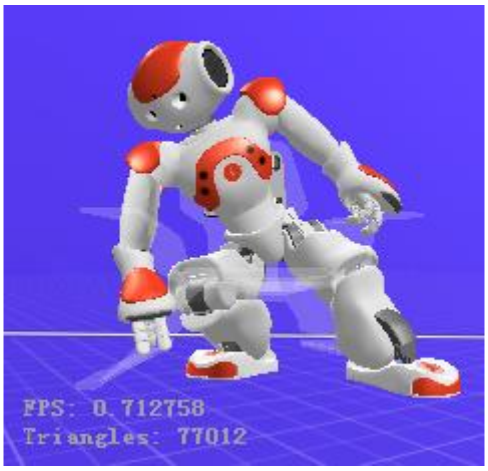


Figure 70. Key frame 70 (2)

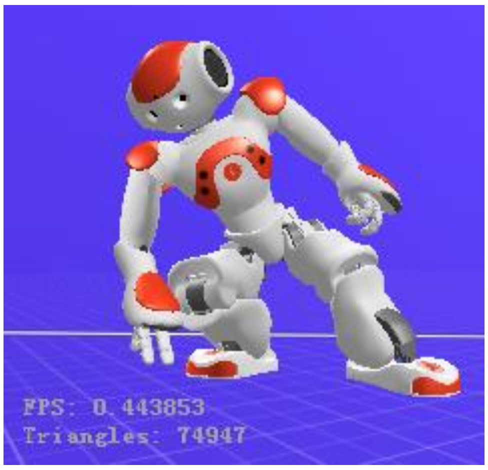


Figure 71. Key frame 90 (2)

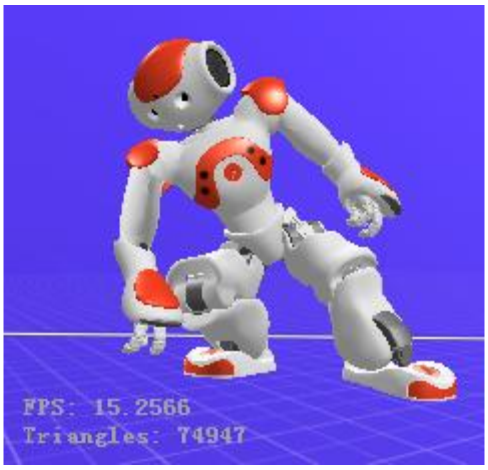


Figure 72. Key frame 110 (2)

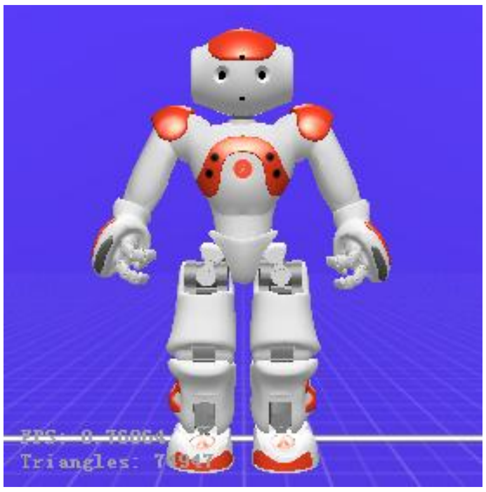


Figure 73. Key frame 175 (2)

5.2.3 Checking the Ball in Hand Animation

For this animation, there are two key frames. First, the robot raises its right arm and puts the hand in the position where the ball is in sight of the robot’s camera and the robot looks down to its hand. Then the robot will lower its right arm and stay in the posture of go-initial. Figure 74 and Figure 75 show every key frame of checking whether the ball is in its hand.

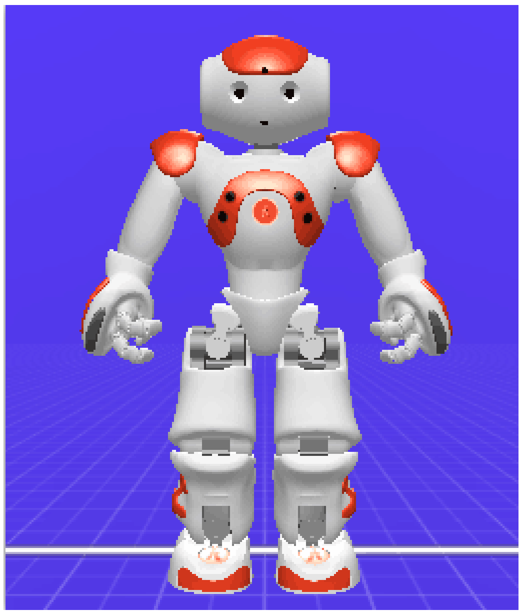


Figure 74. Key frame 60

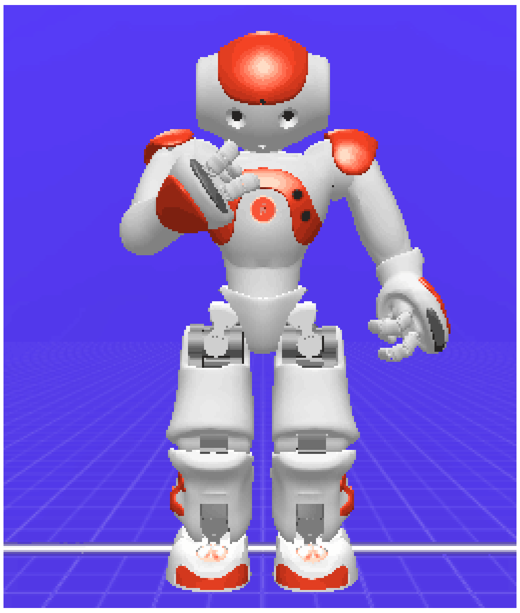


Figure 75. Key frame 80

5.2.4 Throw the ball animation

First, the robot is in the posture of go-initial, then the robot will raise its right arm and move its arm until the right hand is in front of its chest. Then the robot will spin its right wrist until the palm of its right hand faces the ground. Then the robot will open its hand, therefore the ball will drop from its hand. Finally, the robot will move to the initial posture.

In this case, there are total of nine key frames of this animation. The series of the images from Figure 76 to Figure 84 show every key frame of throwing the ball animation:

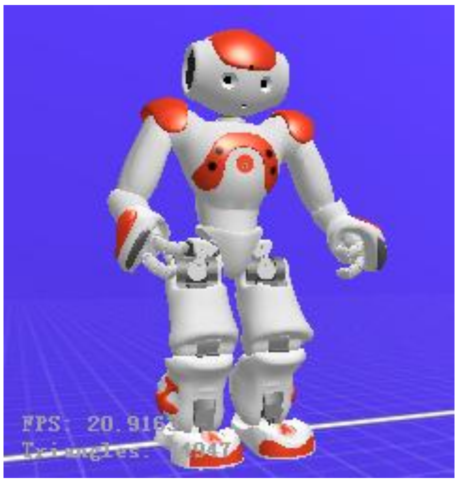


Figure 76. Key frame 10

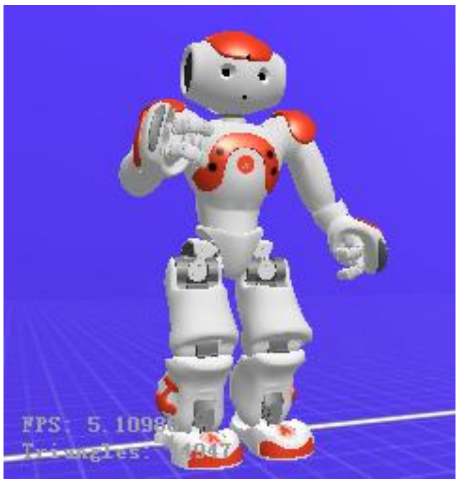


Figure 77. Key frame 20

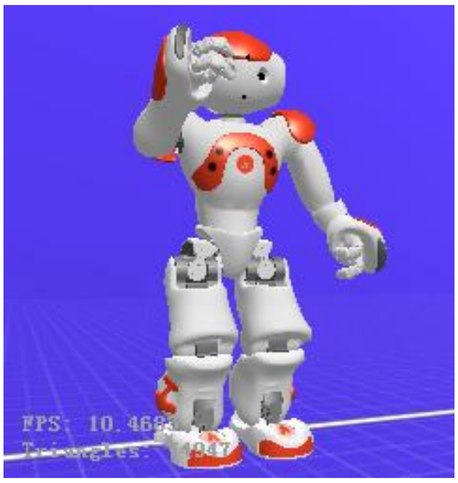


Figure 78. Key frame 30

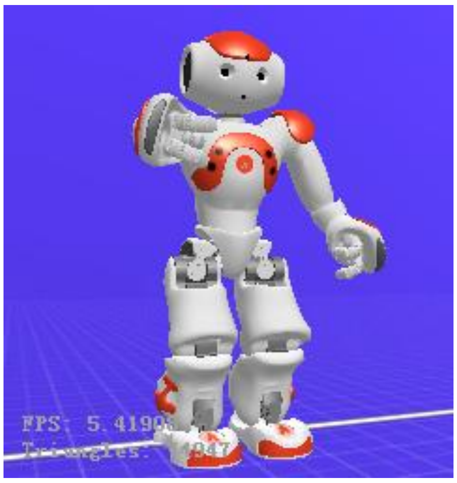


Figure 79. Key frame 45

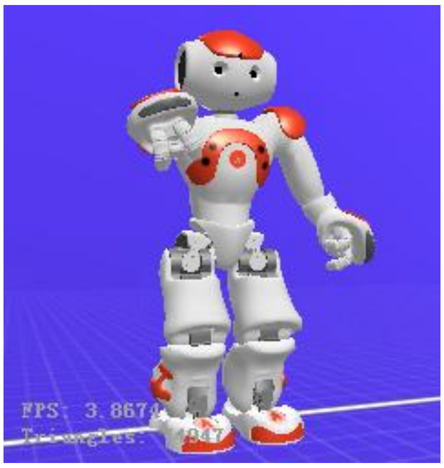


Figure 80. Key frame 60

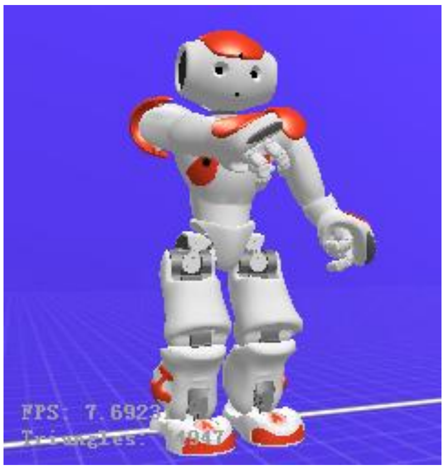


Figure 81. Key frame 75

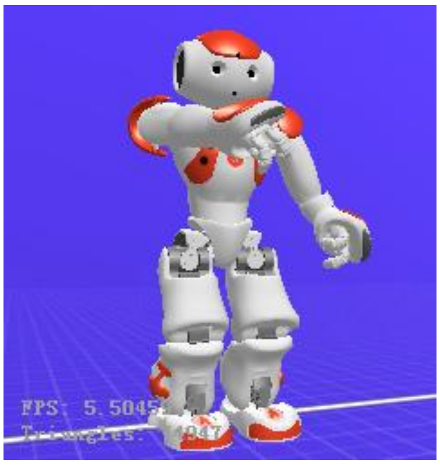


Figure 82. Key frame 90

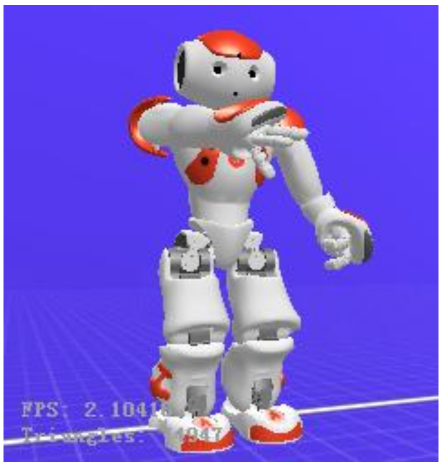


Figure 83. Key frame 100

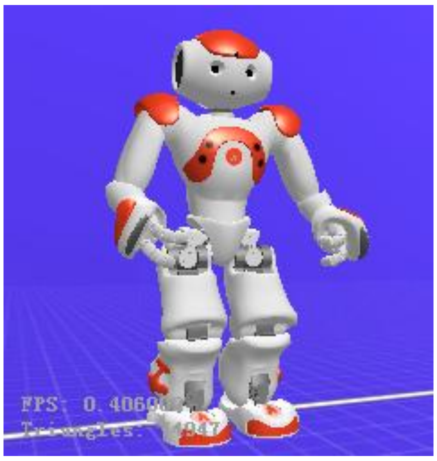


Figure 84. Key frame 120

5.3 Getting the Angles of the Robot's Joints

5.3.1 Checking Whether the Ball Is in Hand

For checking whether the ball is in hand, the module, ALMotion, was used and the method, getAngles(), was used to get the angle of the hand. This module provides methods which facilitate the movement of the robot. That method can be used to get angles of each joints of the robot.

In addition, when the robot picks up the ball, the radian of the angle of its right hand is always larger than 0.3. Otherwise it is smaller than 0.3. Therefore 0.3 is the threshold to decide whether the ball is in the robot's hand. The following is the code of checking whether the ball is in hand.

```
def isBallInHand(IP,PORT):
        ###
    # Summary: This will indicate us if the ball is being
       grabbed by the hand
    # Return: return if ball is in hand or not
    ###

        isIn = False
        motion = ALProxy("ALMotion",IP,PORT)

        # if angles of right hand are >= 0.3 then ball is
in hand
        if(motion.getAngles("RHand",True)[0] >= 0.3):
            isIn = True
        else:
            isIn = False
        print motion.getAngles("RHand",True)[0]
        # return if ball is in hand or not
        return isIn
```

5.3.2 Getting the Angle that Robot's Head Turns

For obtaining the angle that robot's head turns, the module, ALMotion, and the method, getAngle(), were used. In addition, the proxy of ALMotion was created. The first argument of the method is the name of the robot's joint. Therefore, only by specifying the name of the joints, then the method will return its angle. Here is the code for getting the angle of the head turns

```
def getHeadAngles(IP, PORT):
    motion = ALProxy("ALMotion", IP, PORT)
    headDegree = motion.getAngles("HeadYaw", True)[0]
    return headDegree
```

5.4 Development of the Process of Walking

Although there is already a method, moveTo(), of the module, ALMotion, that can be used to let the robot walk. But in real situation, this method does not work. Since sometimes the robot cannot walk to the defined distance, a new method was created to let the robot walk continuously until it reaches the defined distance.

First, the proxy of ALMotion was created. Then the threshold value for the position in x and y axel direction was set to be 1 cm. It means that if the value that the distance which the robot needs to walk subtracts the real distance that the robot moves is higher than that threshold value, the robot will walk continuously. The threshold value for the angle is 0.03 rad. It also means that if the value that the angle which the robot needs to turn subtracts the real angle that robot turns is higher than that threshold value, the robot will turn continuously.

Then by using the method, getRobotPosition(), it will get the position of the robot in FRAME_TORSO. Then by combining the distance and the angle that robot need to move with the original position of the robot, the robot will calculate the final position of the robot in FRAME_TORSO after it reaches the destination. But when the robot stops in the process of walking towards the destination, the robot will check its position. It will calculate the difference value of the real position and the final position. If it is higher than the threshold, the robot will walk the rest of the distance. Otherwise, it will print “move success”. The robot will do the check three times. If the robot still cannot walk to the destination after checking three times, then the robot will not walk anymore. Then the robot needs to rest for a while and run the program again.

After creating this method, the robot can almost always move to the destination. The code can be found in Appendix 3.

5.5 Adjustment When Tracking the Ball

Since in a real situation, the distance that robot moves is not accurate enough and the robot does not turn its body based on the center of its feet strictly, the distance that the robot needs to keep to the ball was modified to be 5.5 cm. And after the robot moves to the specific position before it squats, it will turn its body. The angle is 33.5 degree as mentioned in Chapter 4. Then the robot will move in the y axel direction and the distance is 2.5 cm. Then the robot will move forward and the distance is 1.5 cm. Finally the robot moves to the point where the robot is able to pick up the ball.

5.6 Checking the Temperature of the Robot's Joints

When it comes to the method to check the temperature of the robot's joints, the module, ALMemory, and the method, getData(), was used to get the temperature value of the robot. In this case, only the joints in both of the legs will get hot easily. So the temperature of both joints must be checked and a proxy of the module is created. Here is the code:

```
def checkRightLegTemperature(IP,PORT):
    memory = ALProxy("ALMemory",IP,PORT)    tem-
    perature=memory.getData("Device/SubDevice-
    List/RAnklePitch/Temperature/Sensor/Value")
    return temperature

def checkLeftLegTemperature(IP,PORT):
    memory = ALProxy("ALMemory",IP,PORT)
    temperature=memory.getData("Device/SubDevice-
    List/LAnklePitch/Temperature/Sensor/Value")
    return temperature
```

6 OVERVIEW OF FUTURE RESEARCH

6.1 Recognizing the Box instead of the Nao Mark

In this project, Nao mark was used to find the location of the box. But in real situation, there is not a Nao mark on the surface of the box. So it is better that the robot can recognize the box itself and find the location of the box.

Here are some suggestions about the way to implement the recognition of the box. First, the top camera will be used to find the box, since the range that the top camera can see is farther. The box is always in the place where it is far away from the robot. Then an image of the box will be captured from the box. The color space of the image is transferred from RGB to HSV. The brown color threshold should be found and the shape of the box be captured from the HSV image. Then the contour and the size of the box can be found. Finally, the relationship of the size of the box and the real distance should be found. Then the piecewise linear function can be used to calculate the real distance of the box to the robot. Finally, the robot can recognize the box and walk towards the box.

6.2 Checking Whether the Ball is in the Box

In this case, sometimes the robot did not throw the ball into the box and it was dropped the outside of the box. But the robot would still say ”I succeed”. Therefore, after the robot throws the ball, it should check whether the ball is in the box. If the ball is not in the box, the robot will track the ball and pick up the ball again. Then the robot will walk to the box and throw the ball again.

One of solution is that after the robot has thrown the ball, the robot will look down to the box. Then an image can be captured from the robot’s camera and be cut. Only the area which is the bottom of the box is left. It is as shown in Figure 85. Then by setting the threshold of the red color, the robot can detect the ball. If it is in this area, the ball was found. Then the robot will say “I succeed”. Otherwise, the robot will track the ball and throw the ball again until it the ball goes into the box.

Figure 85. Bottom of the box

6.3 Tracking the ball in Real Time

Since the long term goal of the Botnia Nao robot team is to let the robots join the Robot Cup game, the way that the robot tracks the ball need to be improved. Here are some suggestions:

First, there is no need to measure the distance between the ball and the robot, since in real situation there is no time for calculating the distance but just let the robot walk towards the ball in the direct way. Therefore, the robot should track the ball dynamically. Since the resolution of the image used is 640*480, the coordinate of the center of the robot’s view is [320,240]. The angle of the joint, HeadPitch should always be adjusted. And the robot turn its body at the same time to keep the ball at the position whose coordinate is [320,240]. Finally based on the threshold value of the angle of the HeadPitch, the robot will stop in front of the ball and kick the ball.

The vision module can be created by OpenCV and C++ and all the movement need to be created by C++. In this case, using C++ is more efficient than using Python.

7 SUMMARY

This thesis introduces the behavior design of a humanoid Nao robot: a case of picking up the ball and throwing the ball into the box.

In the vision system stage, the main algorithm is based on filtering noise and Hough circle transform, and by utilizing that algorithm, the accuracy of detecting the location of the center of the ball is improved. In the strategy stage, probability theory is used properly to design the strategy of picking up the ball, which leads to improve the rate of Nao picking up the ball successfully and the rate is almost 85%. Besides, appropriate mathematical models also contributes to calculating the distance to the ball and designing the strategy of tracking the ball dynamically. In addition, the polynomial functions are also utilized to calculate the distance to the box. And key frames in timeline are used to achieve the animation design. Therefore, the key points of this thesis is algorithm in vision system and appropriate mathematical models as well as animation design.

In the beginning, this project is quite challenging. But by separating this huge project into several small modules, this project was completed successfully. And in the process of completing a project, starting from completing some simple parts will simplify the project and provide you confidence. And there is no denying that much knowledge are learnt in the process of completing this project. Now, the methods about how to track a round thing in any color and build mathematical models are clear. Besides, the skills to program the Nao robot are also improved.

When it comes to the suggestions to the students who are interested in the Nao robot, choosing an interesting topic is very important. Before starting programming with Python or C++, Choregraph can be used to get familiar with the environment of programming Nao. Each box in the Choregraph is like the demo code, which may inspire you in the process of doing the project. Through doing project about Nao, the student can improve himself a lot.

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

8 REFERENCES

/1/Aldebaran official website. Accessed 15. April.2015.https://www.aldeba-ran.com/sites/aldebaran/files/images/S%C3%A9lection%2012.jpg

/2/Nao Robot Introduction. Accessed17.April.2015. https://www.aldeba-ran.com/en/humanoid-robot/nao-robot

/3/ Seo KiSung, 2011, Using Nao: Introduction to Interactive Humanoid Robots, Aldebaran robotics & NT research, INC,1,10-24

/4/ Aldebarab Documentation - H25 – Construction. Accessed 20.April.2015. http://doc.aldebaran.com/2-1/family/nao_h25/dimensions_h25.html

/5/ Aldebaran Documentation - NAO Battery. Accessed 23.April.2015. http://doc.aldebaran.com/2-1/family/robots/battery_robot.html

/6/ Aldebaran Documentation - NAO H25. Accessed 25.April.2015. http://doc.aldebaran.com/2-1/family/nao_h25/index_h25.html#nao-h25

/7/ Sonar of the Nao robot. Accessed 26.April.2015. http://doc.aldebaran.com/2-1/family/robots/sonar_robot.html#robot-sonar

/8/Cyberrobotics. Accessed 27.April.2015. https://www.cyberbotics.com/forums_attachments/3226/webott.jpg

/9/Webots overview. Accessed 30.April.2015. http://www.cyberbotics.com/overview

/10/NAOqi Framework. Accessed 1.May.2015 http://doc.aldebaran.com/1-14/dev/naoqi/index.html?highlight=naoqi%20frame-work

/11/ Python IDE - IDLE. Accessed 2.May.2015 https://docs.python.org/2/li-brary/idle.html

/12/ PyDev Python IDE for Eclipse. Accessed 3.May.2015 http://marketplace.eclipse.org/content/pydev-python-ide-eclipse

/13/ Aldebaran Documentation- NAO Video Camera. Accessed 5.May.2015 http://doc.aldebaran.com/2-1/family/robots/video_robot.html#robot-video

/14/ Aldebaran Documentation: Joints. Accessed 5.May.2015 http://doc.aldebaran.com/2-1/family/robots/joints_robot.html

/15/ Opencv Introduction. Accessed 6.May.2015. http://docs.opencv.org/mod-ules/core/doc/intro.html#

/16/Opencv. Accessed 7.May.2015.
http://opencv.org/

/17/ The Blog of Dr Moron. November 9, 2013. Accessed 8.May.2015
http://drmoron.org/is-black-a-color/

/18/ Imgarcade - The HSV Color. Accessed 8.May.2015
http://imgarcade.com/1/hsi-color-model/

/19/ Canny Edge Detection. Accessed 9.May.2015.
http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_imgproc
/py_canny/py_canny.html#canny

/20/NAO Software Documentation: NAOqi. Accessed 9.May.2015
http://doc.aldebaran.com/1-14/naoqi/trackers/index.html#naoqi-trackers

/21/ NAO Software Documentation. Accessed 9.May.2015. http://doc.aldeba-ran.com/1-14/naoqi/vision/allandmarkdetection.html

/22/Aldebaran- Nao Mark. Accessed 10.May.2015
http://doc.aldebaran.com/2-1/_downloads/NAOmark.pdf

/23/NAO Software Documentation: Timeline. Accessed 10.May.2015.
http://doc.aldebaran.com/1-14/software/choregraphe/panels/timeline_panel.html

APPENDIX 1.

FIND THE RED COLOR THRESHOLD

```
import cv2

import numpy as np

 red=np.uint8([[[227,62,56]]])

hsv_red=cv2.cvtColor(red,cv2.COLOR_BGR2HSV)

print hsv_red
```

APPENDIX 2.

MATLAB COMMAND FOR FINDING THE RELATIONSHIP OF REAL DISTANCE AND SIZE X

```
x=[0.228,0.188,0.158,0.138,0.121,0.108,0.098,0.091,0.085,0.078,0.073,0.070,0.0
65]

y=[0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0,1.1,1.2,1.3,1.4,1.5]

P = polyfit(x,y,3)

y2=-485.5931*x.^3+266.2636*x.^2-50.8341*x+3.7969

plot(x,y,'*',x,y2,'-')
```

APPENDIX 3.

MOVETO METHOD

```
def moveTo(x, y, degree, IP, PORT):


    motion=ALProxy("ALMotion", IP, PORT)
    positionErrorThresholdPos = 0.01
    positionErrorThresholdAng = 0.03


    # The command position estimation will be set to the sensor position
    # when the robot starts moving, so we use sensors first and commands later.
    initPosition = almath.Pose2D(motion.getRobotPosition(True))
    targetDistance = almath.Pose2D(x,y,degree * almath.PI / 180)
    expectedEndPosition = initPosition * targetDistance
    enableArms = True
    motion.setMoveArmsEnabled(enableArms, enableArms)
    #motion.moveTo(x, y, degree * almath.PI / 180)
    motion.moveTo(x, y, degree)
    # The move is finished so output
    realEndPosition = almath.Pose2D(motion.getRobotPosition(False))
    positionError = realEndPosition.diff(expectedEndPosition)
    positionError.theta = almath.modulo2PI(positionError.theta)


    index=0
    while (abs(positionError.x) > positionErrorThresholdPos or abs(positionEr-
            ror.y) > positionErrorThresholdPos or abs(positionError.theta) >
            positionErrorThresholdAng):
```

```
    errorX=positionError.x
    errorY=positionError.y
    errorDegree=positionError.theta

    motion.moveTo(errorX, errorY, errorDegree * almath.PI / 180)
    realEndPosition = almath.Pose2D(motion.getRobotPosition(False))
    positionError = realEndPosition.diff(expectedEndPosition)
    positionError.theta = almath.modulo2PI(positionError.theta)
    print "go again"
    print positionError.toVector()
    time.sleep(2)
    index+=1
    if(index==3):
        break



if (abs(positionError.x) < positionErrorThresholdPos
    and abs(positionError.y) < positionErrorThresholdPos
    and abs(positionError.theta) < positionErrorThresholdAng):
    print "move success!"
else:
    print positionError.toVector()

motion.moveToward(0.0, 0.0, 0.0)
```
.