



Minxue Xia

BEHAVIOR DESIGN OF NAO ROBOT PLAYING BLACKJACK

Information Technology

2015

FOREWORD

The program presented in the thesis has been developed from June 2015 to October 2015.

In the first place, I would like to take this opportunity to express my deep gratitude to my supervisor, Dr. Yang Liu, who has seen me through my journey in the department of *Information Technology*, at *Vaasan Ammattikorkeakoulu, Vaasa University of Applied Sciences*. Without his assistance and instructions, I would have hardly completed my studies in this research. He has also given plenty of useful advice and guidance in almost all my specialized courses, which at last embarked the program in the field of *Nao* robot.

In the second place, special thanks to all the lectures, including Dr. Ghodrat Moghadampour, Dr. Chao Gao, Dr. Smail Menani, Mr. Jani Ahvonen, Mr. Santiago Chavez Vega, Mr. Antti Virtanen, Mr. Jukka Matila and all the colleagues who have taught and helped me throughout my studies in the university.

In the third place, I would like to thank my family, especially my mother, Zeng Qiong, for all the support and understanding given during my study.

Xia Minxue

Vaasa, 9th of October, 2015

ABSTRACT

Author	Minxue Xia
Title	Behavior Design of Nao Robot Playing Blackjack
Year	2015
Language	English
Pages	70 + Appendices
Name of Supervisor	Yang Liu

The principal objective of this thesis was to explore the possibility of applying *Nao* robot in the gambling or gaming industry. In this thesis, *Nao* played *Blackjack* game as a player and announced the result of each round. The thesis work was designed and implemented by applying the combination of computer vision and behaviour design of a *Nao* robot.

Python as well as *OpenCV* were used in Windows platform throughout the whole project. As for the hardware, the robot is in the newest 5th generation and the *NAoqi* is in the 2.1 version for programming. Moreover, software *Choregraphe* was regarded as an alternative platform connecting *Python* with the *Nao* robot.

The application implementation method is typically software engineering approaches including planning, developing, debugging and testing.

Finally, the goals of the application were overall achieved in both precision and timing.

CONTENT

ABSTRACT	3
LIST OF FIGURES AND TABLES	6
LIST OF APPENDICES	9
1. INTRODUCTION	10
2. OVERALL STRUCTURE.....	15
3. COMMUNICATION MODULE.....	17
3.1 Software Architecture	17
3.1.1 Python	17
3.1.2 OpenCV	19
3.1.3 NumPy&SciPy	21
3.1.4 Choregraphe.....	22
3.2 Monitor	23
3.3 Webots	23
3.4 Naoqi.....	24
3.4.1Naoqi Framework	24
3.4.2 Naoqi Process.....	25
4. VISION MODULE	28
4.1 NAO Vision Introduction	29
4.1.1 Camera Specifications	29
4.1.2 Vision Range.....	31
4.2 Image Acquisition.....	33
4.2.1 Original Images.....	33
4.2.2 Head Adjustments.....	34
4.2.3 Image From <i>Choregraphe</i>	35
4.3 Pre-Processing.....	37
4.3.1 Change Color-spaces	37
4.3.2 Smoothing Images	38
4.3.3 Image Thresholding	39
4.4 Feature Extraction	40
4.4.1 Draw Contours	40
4.4.2 Find Edges	41
4.4.3 Perspective Transform	43
4.5 Recognizing Cards.....	45
4.6 Comparing Differences.....	47
5. NAO BEHAVIORS	50
5.1 Joints Introduction	50
5.1.1 Head Joints.....	50
5.1.2 Arm Joints.....	52
5.2 Nao Behaviors.....	54

5.2.1 Animation	54
5.2.2 Get Angles	57
5.2.3 Set Angles	58
5.2.4 Arm Move.....	58
5.3 Results Announcement	61
6. BLACKJACK ALGORITHMS.....	63
7. IMPROVEMENTS.....	64
8. CONCLUSION.....	66
REFERENCES	68

LIST OF FIGURES AND TABLES

Figure 1. Decreasing Profits of Casino in USA	p. 10
Figure 2. Croupier Robot in Exhibition	p. 11
Figure 3. Relations between Key Components	p. 12
Figure 4. Nao Robot	p. 13
Figure 5. <i>Nao V5</i> Specifications	p. 14
Figure 6. Nao Operating System Structure	p. 15
Figure 7. Thesis Structure	p. 16
Figure 8. <i>Python</i> Installation Page	p. 17
Figure 9. <i>Python Shell IDLE</i> Window	p. 18
Figure 10. <i>PyDev</i> Initial Window	p. 19
Figure 11. <i>OpenCv</i> Packages	p. 20
Figure 12. <i>OpenCV</i> Installation Path	p. 20
Figure 13. Test on Python Successfully	p. 21
Figure 14. Test on Python Unsuccessfully	p. 21
Figure 15. <i>Choregraphe</i> Interface	p. 22
Figure 16. Monitor Window	p. 23
Figure 17. Webots Interface	p. 24
Figure 18. Methods in <i>Naoqi</i>	p. 25

Figure 19. Relations between Broker, modules and methods	p. 26
Figure 20. relations between broker, libraries and modules	p. 27
Figure 21. Vision Module Structure	p. 29
Figure 22. Location of <i>Nao</i> Cameras	p. 30
Figure 23. Camera Specifications	p. 31
Figure 24. Vision Range	p. 32
Figure 25. Horizontal Range	p. 32
Figure 26. Initial Position	p. 33
Figure 27. Initial View of <i>Nao</i>	p. 34
Figure 28. <i>Nao</i> Vision Range	p. 34
Figure 29. <i>Nao</i> 's View After Adjustments	p. 35
Figure 30. Python Code	p. 36
Figure 31. <i>Nao</i> Connection through <i>Python</i>	p. 37
Figure 32. Gray Image	p. 38
Figure 33. Blurring Image	p. 39
Figure 34. Threshold Image	p. 40
Figure 35. Contours of Cards	p. 41
Figure 36. Find Areas of Cards	p. 42
Figure 37. Edges of Cards	p. 43

Figure 38. Perspective Transform	p. 44
Figure 39. Separate Each Card	p. 45
Figure 40. Four types of Cards	p. 46
Figure 41. Representation of 52 Cards	p. 46
Figure 42. Relations Between Card Numbers and File Names	p. 47
Figure 43. Difference between Heart 2 and Heart 5	p. 48
Figure 44. The Similarity of Comparing Heart 2 with Itself	p. 48
Figure 45. Original Cards	p. 49
Figure 46. Python Results of Recognizing Cards	p. 49
Figure 47. Nao Head Joints	p. 51
Figure 48. Head Joints Movement	p. 52
Figure 49. The Specifications of Right Arm	p. 53
Figure 50. Animation Logo on <i>Choregraphe</i>	p. 55
Figure 51. First Part of Animation	p. 56
Figure 52. Second Part of Animation	p. 57
Figure 53. Code for Nao Arm	p. 61
Figure 54. Python Code for Saying	p. 62
Figure 55. Artificial Neuron Network	p. 64
Figure 56. Neuron Nodes	p. 65

LIST OF ABBREVIATIONS

3D	Three Dimension
V5	Version 5
CV	Computer Vision
OpenCV	Open Source Computer Vision Library
FSR	Force Sensors Resistors
DCM	Device Control Manager
IDLE	Integrated Development Environment
BSD	Berkeley Structure Distribution
PC	Personal Computer
OS	Operating System
VGA	Video Graphics Array
RGB	Combinational Color of Red, Green, Blue
BJ	BlackJack
ANNs	Artificial Neural Network

1. INTRODUCTION

The worldwide casino industry is faltering nowadays. While an increasing number of them are commercial licensed, profits are sinking in a gradually saturating market. New casinos are popping up but there are not enough matching profits. Figure 1 illustrates the American gambling revenue in 2007 and 2013. Currently, casinos worldwide are desperate to cut costs in order to save sinking profits.

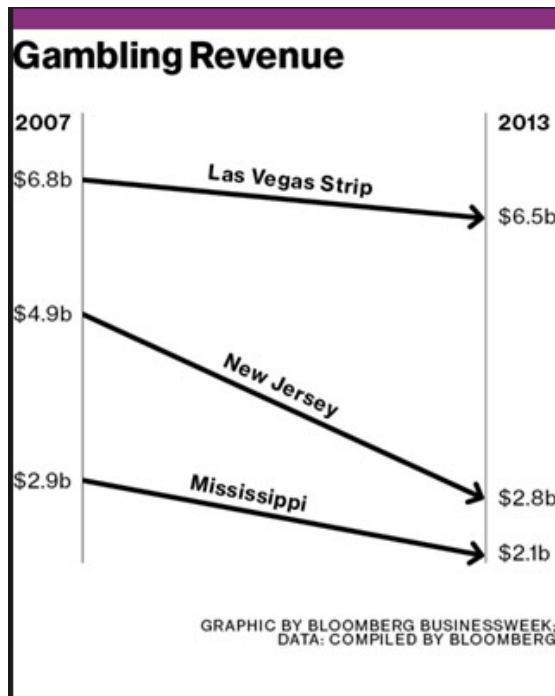


Figure 1. Decreasing Profits of Casino in USA

Therefore, a robot croupier may help cut costs in the medium-term. When a human being is hired, it always comes with high wages, break times, paid vacations as well as health benefits while robots only require a purchase cost once and maintenance fees. Cheaper and more sophisticated robots will definitely be welcomed in the not-so-distant future. Figure 2 shows a robot in a technologic exhibition.

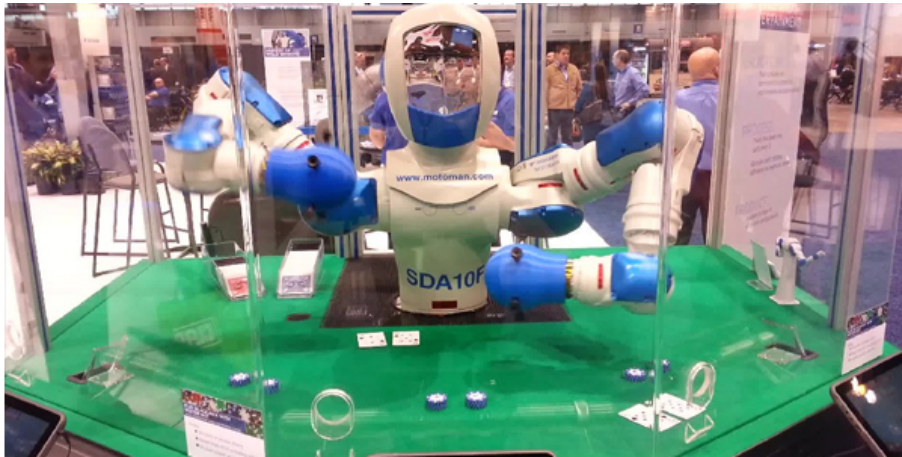


Figure 2. Croupier Robot in Exhibition

The thesis introduces the development in computer vision in detail for the Nao Robot Team. The initial aim of this application is to raise efficiency and reduce labor costs in the gambling industry. Nevertheless, this technology can be further expanded to be applied in other relevant fields also.

1.1 Robotics

Robotics is a new field that emerged in recent decades, combining knowledge of engineering with computer science in order to deal with the design, construction, operation, and application of robots, as well as computer systems for their control, sensory feedback and information processing. Also it is substantially different from those of the original technologies used before.

Nowadays, higher level of mobility and dexterity are required in order to work in a variety of ranges, access multiple places, handle different problems, and perform flexible tasks. Therefore, a change to the structure of robots has merged, from original mechanical design to assembling human structure.

Concretely, a *robot* is a programmable, multifunctional mechanical agent designed to sense and exhibit intelligent behaviors based on various levels of technological sophistication, ranging from a simple material handling device to a humanoid. It is

widely used in various industries today. The key components of a *robot* consist of power, sensors, actuators, controller, user interface as well as a manipulator. The relations between those components are presented in Figure 3.

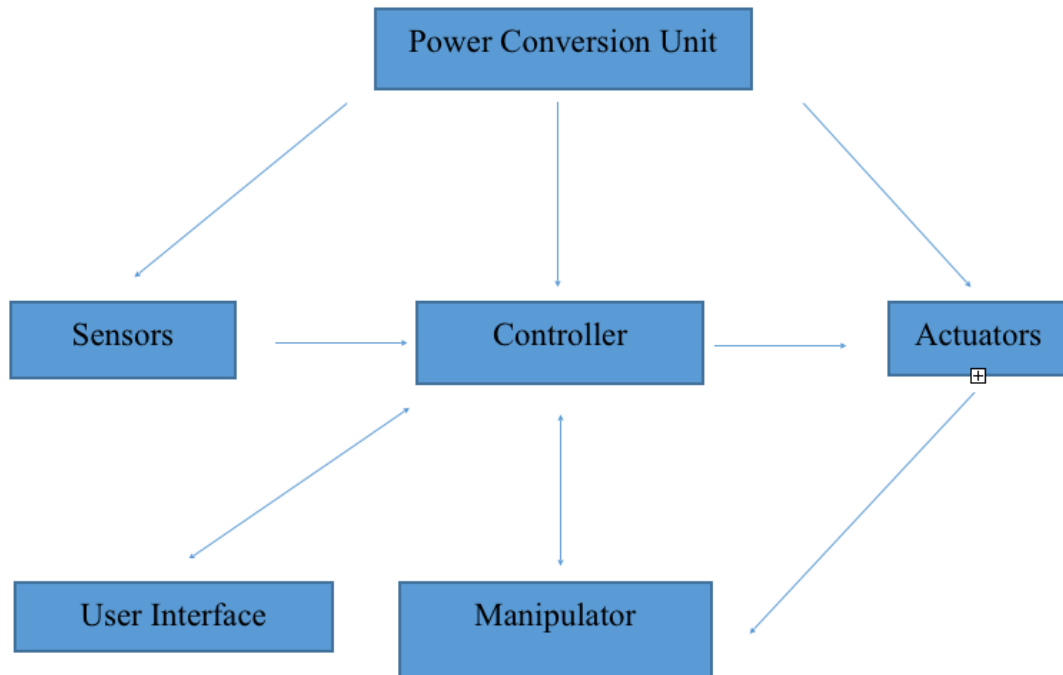


Figure 3. Relations between Key Components

1.2 Computer Vision

Computer vision is to make computers understand and interpret images and video like human beings do. The human vision system has definitely no problem perceiving the three-dimensional structure of the surrounding world. Researchers in this field have developed reliable techniques for computer vision system to make computers see just like people do, for example, naming objects, identifying people, inferring 3D geometry of things, tracking a person against a complex background, or understanding relations, emotions, actions and intentions. However, despite all these advancements, having a computer interpret an image still remains at the same level with a two-year old child. This is because vision is an inverse problem and we need to model the visual world

with its rich complexity first, and then provide solutions to the computer.

1.3 Nao Robot Overview

Nao Robot is an autonomous, programmable humanoid robot developed by *Aldebaran Robotics*, a French robotics company headquartered in Paris. He is 58-cm tall and born in 2006, is able to move, recognize, hear and even talk to human beings. *Nao robot* is a platform of *Two-Legged Standard League*.

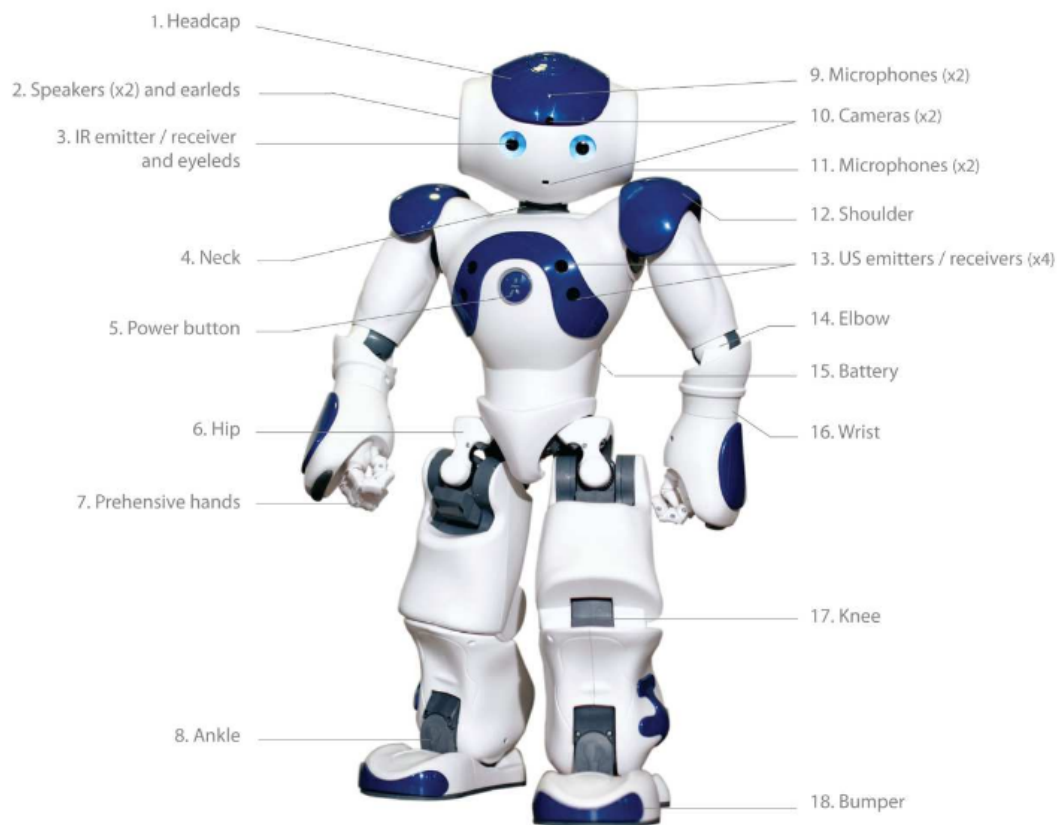


Figure 4. *Nao Robot*

Below Figure 5 shows the specifications of *Nao V5 Evolution*:

Nao V5 Evolution	
Height	58 cm (23 in)
Weight	4.3 kilograms (9.5 lb)
Power supply	lithium battery providing 48.6
Autonomy	90 minutes (active use)
Degrees of freedom	25
CPU	Intel Atom @ 1.6 GHz
Built-in OS	NAOqi 2.0 (Linux-based)
Compatible OS	Windows, Mac OS, Linux
Programming languages	C++, Python, Java, MATLAB, Urbi, C, .Net
Sensors	Two HD cameras, four microphones, sonar rangefinder, two infrared emitters and receivers, inertial board, nine tactile sensors, eight pressure sensors
Connectivity	Ethernet, Wi-Fi

Figure 5. *Nao V5* Specifications

Nao is equipped with many sensors in order to receive information from its surroundings:

Ultrasound Providing space distance in the range of 1 meter and below 30 degrees of the robot chest.

Cameras There are two cameras acting as *Nao*'s eyes. One is located in the forehead and the other is in the lip level.

Bumper Situated in front of each foot, bumpers act like an alarm if *Nao* touched obstacles.

Force Sensors *Nao* has 8 *Force Sensors Resistors (FSR)* located in its two feet respectively. It is usually related to the distance of movements.

Inertial Sensors Always checking whether *Nao* is in a stable position.

The major operating system is *NaoQi* designed as a distributed system with three parts of *NaoQi Operating System*, *NaoQi Library* and *Device Control Manager (DCM)*.

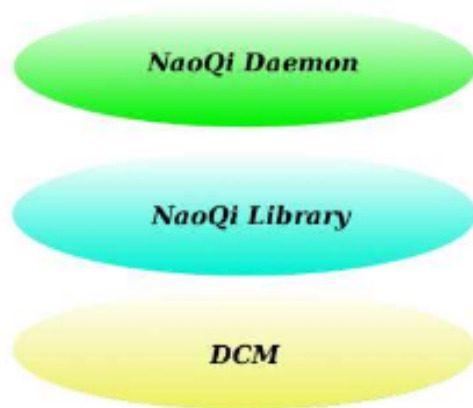


Figure 6. Nao Operating System Structure

2. OVERALL STRUCTURE

This whole program mainly consists of five modules connecting the Nao robot with PC (Figure 7). Computer vision module as well as motion module happens on Nao robot taking care of information collection. PC controller and algorithms decide the strategy of the game while communication module relates these two parts together. To be specific, the five modules are divided into 8 chapters to be explained in the thesis work.

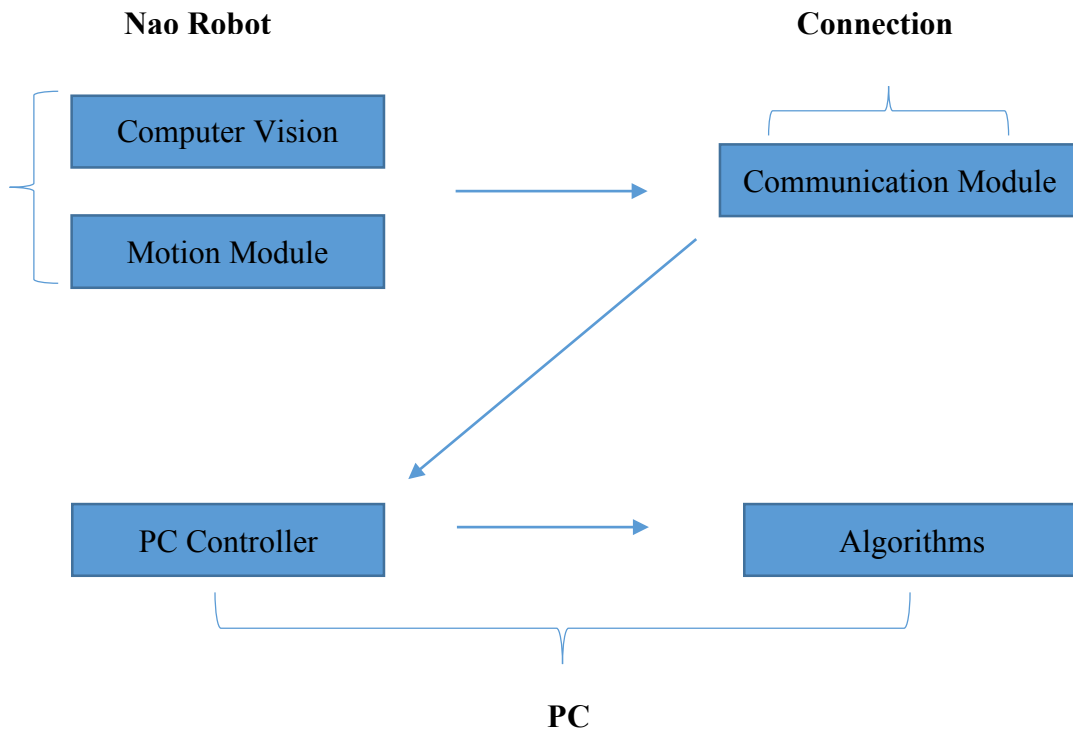


Figure 7. Thesis Structure

3. COMMUNICATION MODULE

3.1 Software Architecture

3.1.1 Python

Due to its simplified syntax and design philosophy, Python makes concepts expressed simpler and more clearly than any other programming languages such as C++ or Java.

- **Installation**

Go to the webpage of Python <https://www.python.org/> . And choose the suitable package to download (Figure 8).



Figure 8. *Python* Installation Page

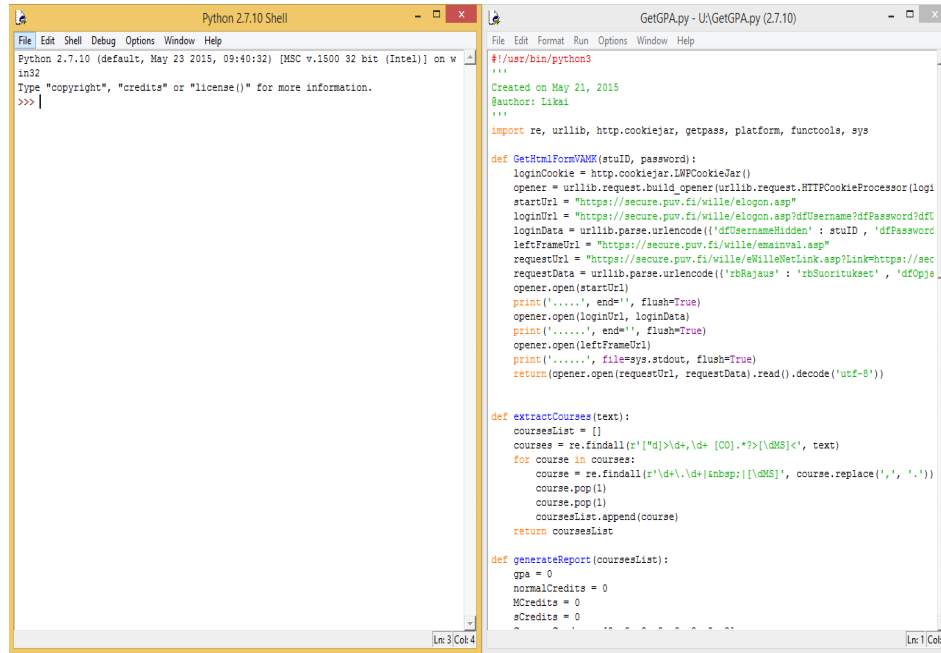
- **Python Interface**

There are numerous interfaces that can build and execute Python, for instance, IDLE and Eclipse. In this project, Python IDLE was used to launch different Python tasks.

➤ **IDLE**

IDLE (Integrated Development Environment) is an integrated development environment for *Python*, which has been bundled with the default

implementation of the language. It is packaged as an optional part of the *Python* packaging with many *Linux* distributions. (Figure 9)



```
Python 2.7.10 Shell
Python 2.7.10 (default, May 23 2015, 09:40:32) [MSC v.1500 32 bit (Intel)] on w
in32
Type "copyright", "credits" or "license()" for more information.
>>> |

GetGPA.py - U:\GetGPA.py (2.7.10)
#!/usr/bin/python3
...
Created on May 21, 2015
@author: Likai
...
import re, urllib, http.cookiejar, getpass, platform, functools, sys

def GetHtmlFormUNK(stuID, password):
    loginCookie = http.cookiejar.CookieJar()
    opener = urllib.request.build_opener(urllib.request.HTTPCookieProcessor(logi
startUrl = "https://secure.puv.fi/wille/elogon.asp"
loginUrl = "https://secure.puv.fi/wille/elogon.asp?dfUsername=dfPassword=dfC
loginData = urllib.parse.urlencode({'dfUsernameHidden': stuID, 'dfPasswor
leftFrameUrl = "https://secure.puv.fi/wille/eWilleNetLink.asp"
requestUrl = "https://secure.puv.fi/wille/eWilleNetLink.asp?Link=https://sec
requestData = urllib.parse.urlencode({'rbRajaus': 'rbSuoritukset', 'dfOpje
opener.open(startUrl)
print('.....', end='', flush=True)
opener.open(loginUrl, loginData)
print('.....', end='', flush=True)
opener.open(leftFrameUrl)
print('.....', file=sys.stdout, flush=True)
return(opener.open(requestUrl, requestData).read().decode('utf-8'))

def extractCourses(text):
    coursesList = []
    courses = re.findall('[<td>de, <td> [CO].*>[\dMS]<', text)
    for course in courses:
        course = re.findall('[<td>[\dMS]', course.replace(' ', '.'))
        course.pop(1)
        course.pop(1)
        coursesList.append(course)
    return coursesList

def generateReport(coursesList):
    gpa = 0
    normalCredits = 0
    MCredits = 0
    SCredits = 0
    ...
```

Figure 9. *Python Shell IDLE* Window

➤ ***Eclipse***

Eclipse is an integrated development environment for various programming languages. Written mostly in *Java*, *Eclipse* can be used to develop applications in *C*, *C++*, *Python*, *PHP* and so on, based on its means of plug-ins. *PyDev* is a *Python IDE* for *Eclipse* which may be used in *Pyhon*, *Jython* and *IronPython* development (Figure 10).

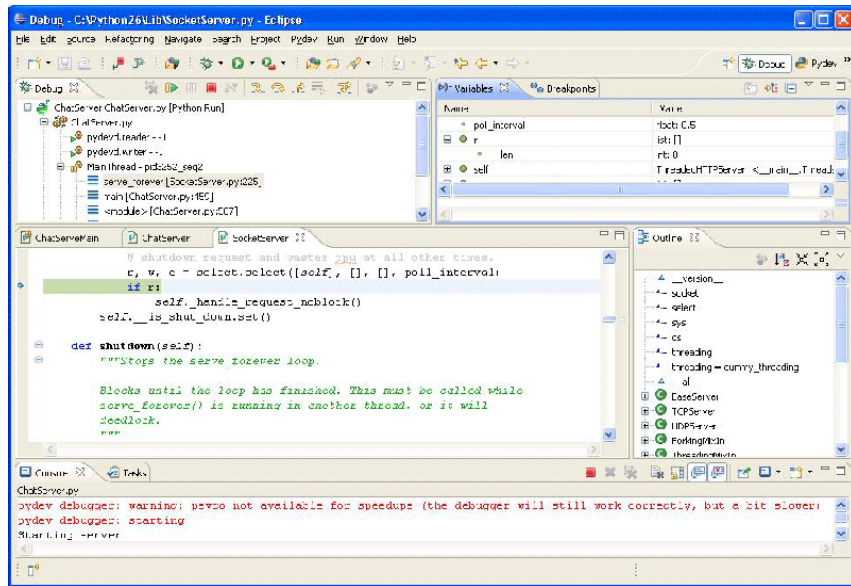


Figure 10. PyDev Initial Window

3.1.2 OpenCV

OpenCV is a programming library mainly designed for computational efficiency and aimed at real-time computer vision applications. It is under a *BSD* license and supports *C*, *C++*, *Python* as well as *Java* languages.

- **Installation**

- Find the suitable package for the python vision you installed from the Sourceforge of OpenCV. Remember to choose the corresponding Python installation package!

[\(http://sourceforge.net/projects/opencvlibrary/files/opencv-win/\)](http://sourceforge.net/projects/opencvlibrary/files/opencv-win/)

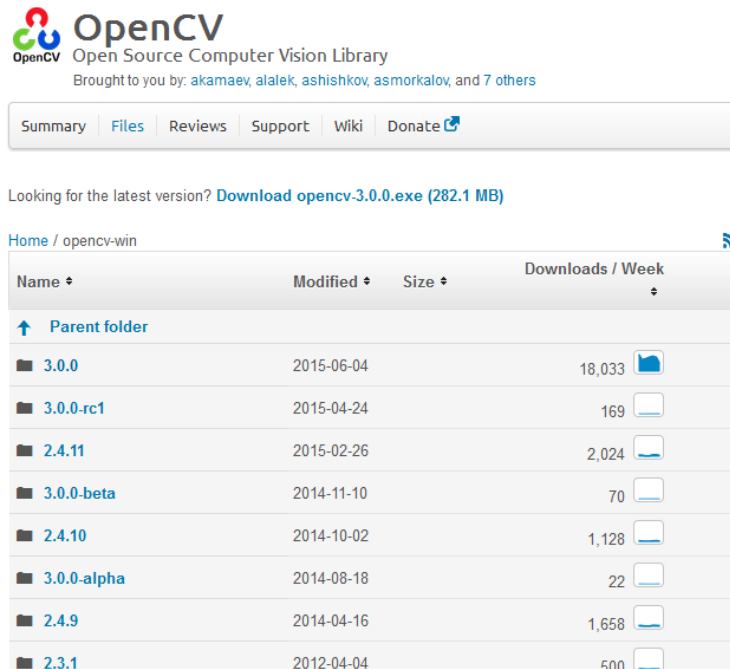


Figure 11. *OpenCv* Packages

- Unpack the self-extracting archive.
- Check the installation at the chosen path coordinates with your Python installation path. Add cv2.pyd file to the following directory:

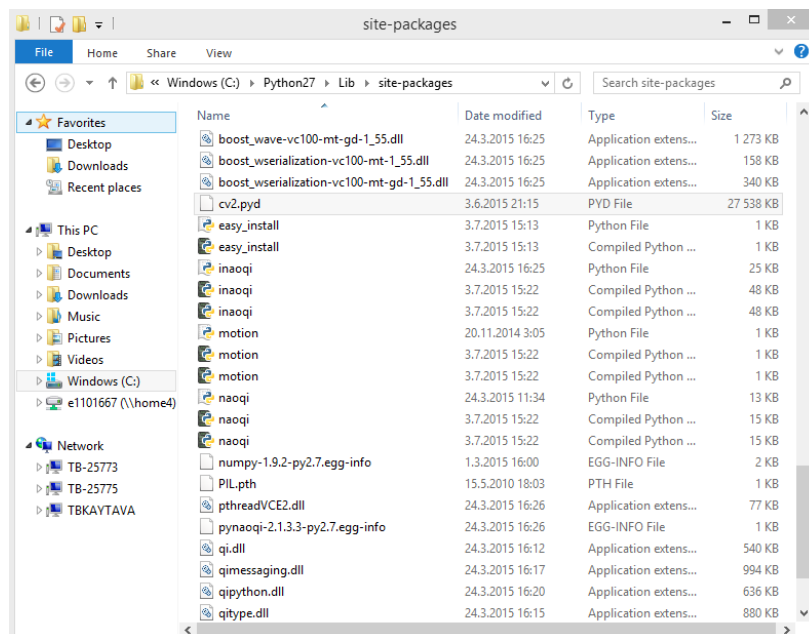


Figure 12. *OpenCv* Installation Path

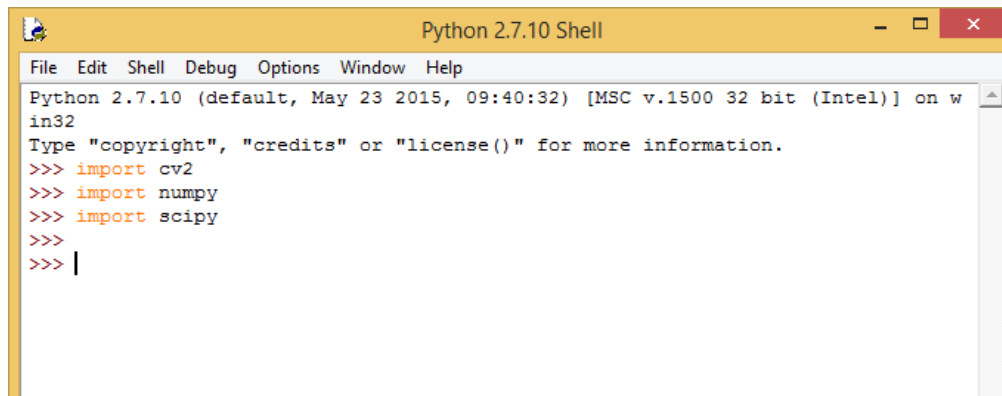
3.1.3 NumPy&SciPy

NumPy and *SciPy* are open-source add-on modules to *Python* that provide common mathematical and numerical routines in pre-compiled, fast functions.

- **Installation**

- Download a suitable package for Python version (Python 2.7 in my case) and unpack the self-extracting archive.
- Test on Python

Enter following lines shown in Figure 13 on Python 2.7.10 Shell:



```
Python 2.7.10 Shell
File Edit Shell Debug Options Window Help
Python 2.7.10 (default, May 23 2015, 09:40:32) [MSC v.1500 32 bit (Intel)] on w
in32
Type "copyright", "credits" or "license()" for more information.
>>> import cv2
>>> import numpy
>>> import scipy
>>>
>>> |
```

Figure 13. Test on Python Successfully

If installed unsuccessfully, Python will show the following lines like Figure 14 shown below:

```
>>>
>>> import PIL

Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    import PIL
ImportError: No module named PIL
>>>
>>>
```

Figure 14. Test on Python Unsuccessfully

3.1.4 Choregraphe

Choregraphe is a multi-platform desktop application which allows you to create animations, test them and monitor Nao through programming or just simply through the various behavior boxes.

- **Installation**

- Make sure you have logged on as “administrator” or a user with “administrator” privileges to download and install the software.
- Download the recent release from the *Aldebaran Community Website* <https://community.aldebaran.com/> .
- Choose “Resources”, “Software”.
- Then follow the steps on the page, first create an account and sign in.
- Next, find the suitable package for yourself and install. In addition, remember to install “Bonjour” as well in the installation process of *Choregraphe*.

- **Interface**

Figure 15 shows the initial *Choregraphe* interface.

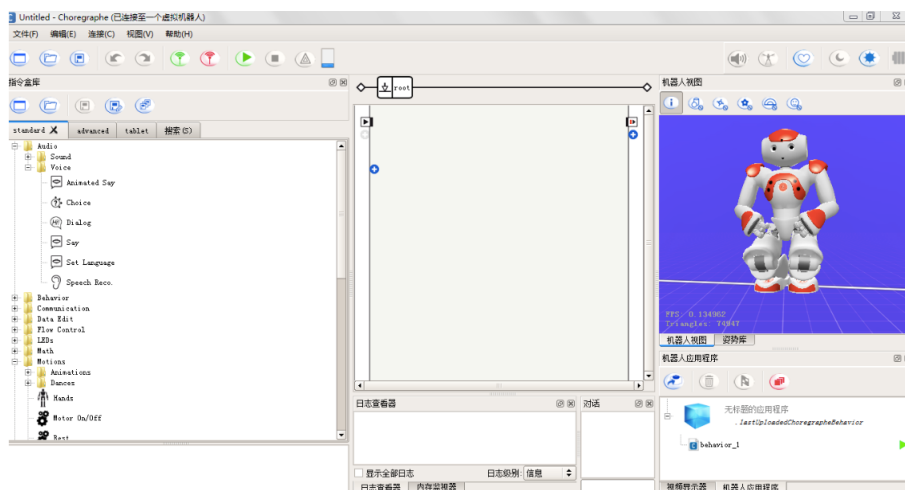


Figure 15. Choregraphe Interface

3.2 Monitor

Monitor is a desktop application which gives feedbacks of what Nao is seeing and feeling to users. With the Camera module, data from the camera will be received. With the Memory module, data from the robot's sensors will be accessed in an ergonomic approach. Moreover, it also gives the possibility to test vision algorithms on recorded excerpts.

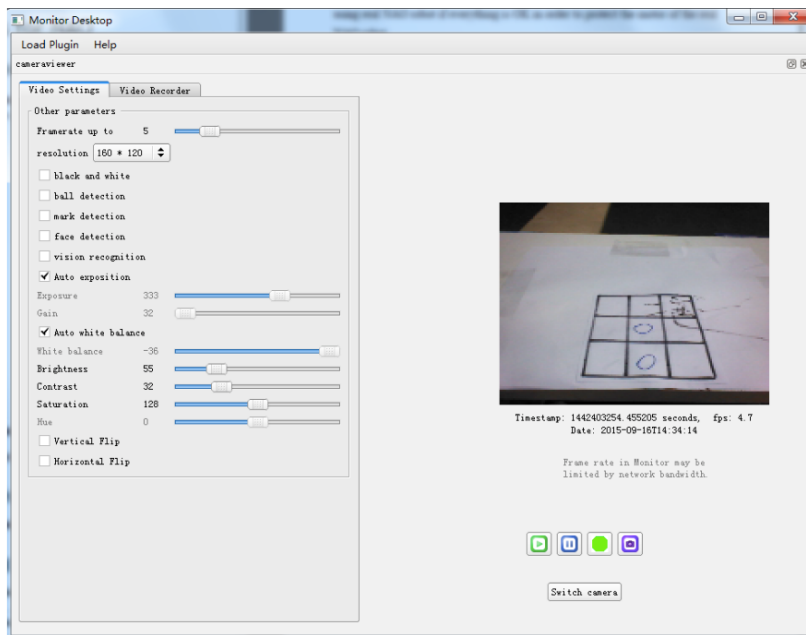


Figure 16. Monitor Window

3.3 Webots

The simulator Webots allows users test the algorithms in a virtual world governed by real physics. It is a perfect software together Monitor as well as Choregraphe with to test behaviors before playing on the real Nao robot.

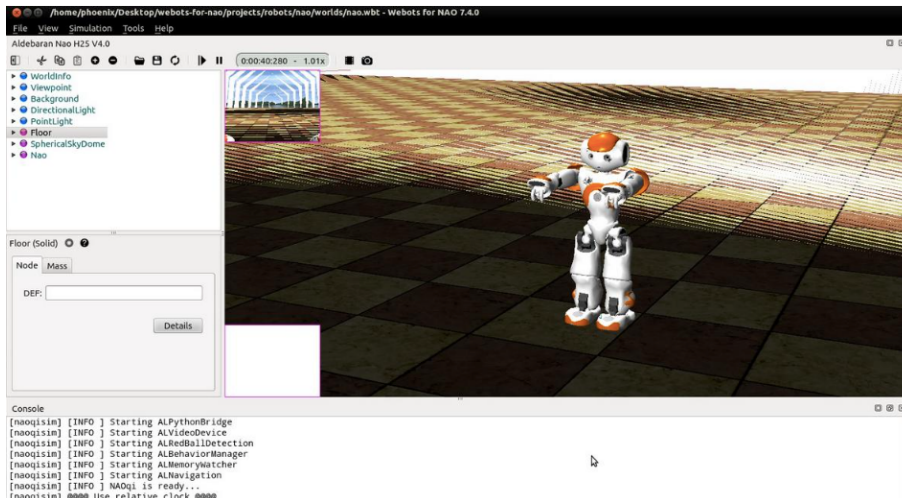


Figure 17. Webots Interface

3.4 Naoqi

3.4.1 Naoqi Framework

Naoqi is the main software to control the Nao robot. The framework provides the basic needs of Nao, including parallelism, resources, synchronization as well as events. It allows homogeneous communication between various modules, such as motion, audio and video. Meanwhile, the main characteristics of Naoqi framework are cross platform and cross language.

- **Cross Platform**

Three platform support Naoqi framework: Linux, Windows as well as MacOS. It is possible to develop by the following two languages:

- Using Python

Code can be run easily both on personal computers or directly on a robot.

- Using C++

In the first place, code need to be compiled for the targeted operating system

since it is a compiled language. It is essential to install a cross-compile tool in order to run C++ on the robot operating system Naoqi OS.

- **Cross Language**

Both Python and C++ can be used to develop software, moreover, the methods in Naoqi framework are almost the same.

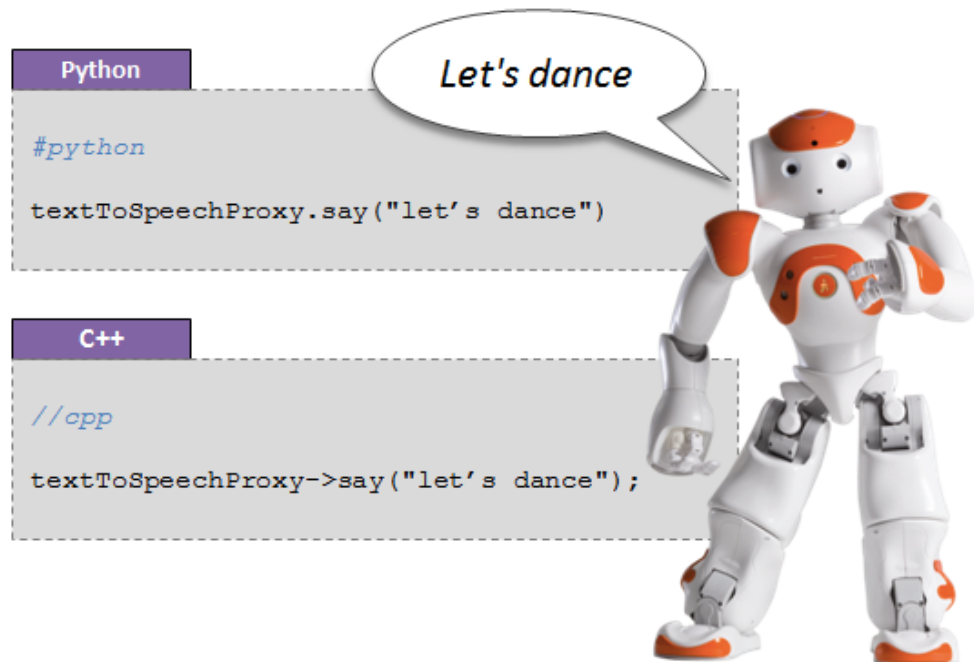


Figure 18. Methods in *Naoqi*

3.4.2 Naoqi Process

- **Broker**

A broker is an object which provides:

- **Directory services**

Any method that has been advertised can be found by any module in the network. Loading modules constitutes a group of methods related to

modules, meanwhile, modules attached to a broker.

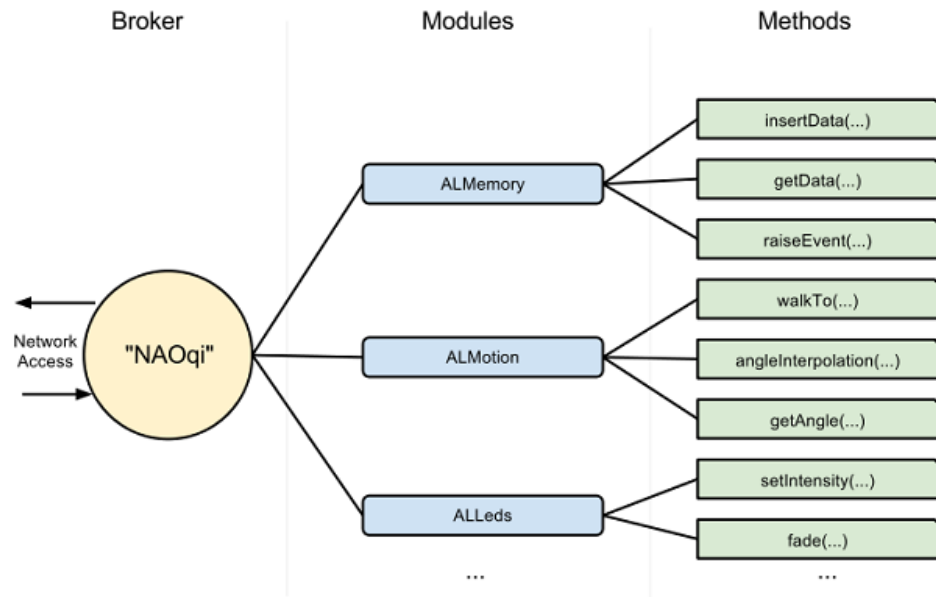


Figure 19. Relations between Broker, modules and methods

➤ **Network access**

Allowing the methods of attached modules to be called from outside the process. Once it starts, it loads a preferences file called *autoload.ini* that defines which libraries it should load. Figure 20 shows the relations between broker, libraries and modules.

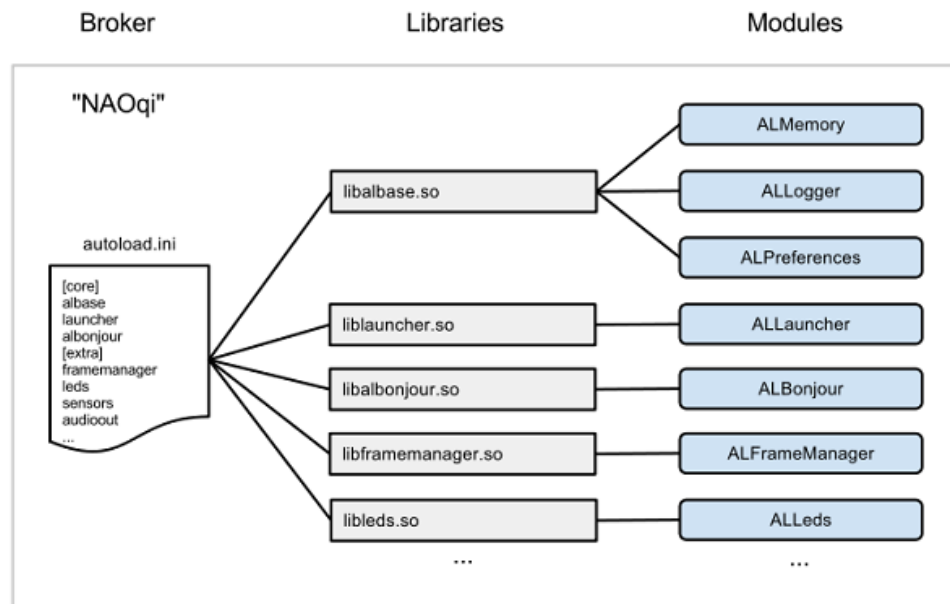


Figure 20. Relations between broker, libraries and modules

- **Proxy**

A proxy is an object that will behave as the module it represents. For example, if you create a proxy to the ALMotion module, you will get an object including all the ALMotion methods.

There are two options to create a proxy to a module:

- Local call

Using the name of the module. Current code and the targeted module must be in the same module.

- Remote call

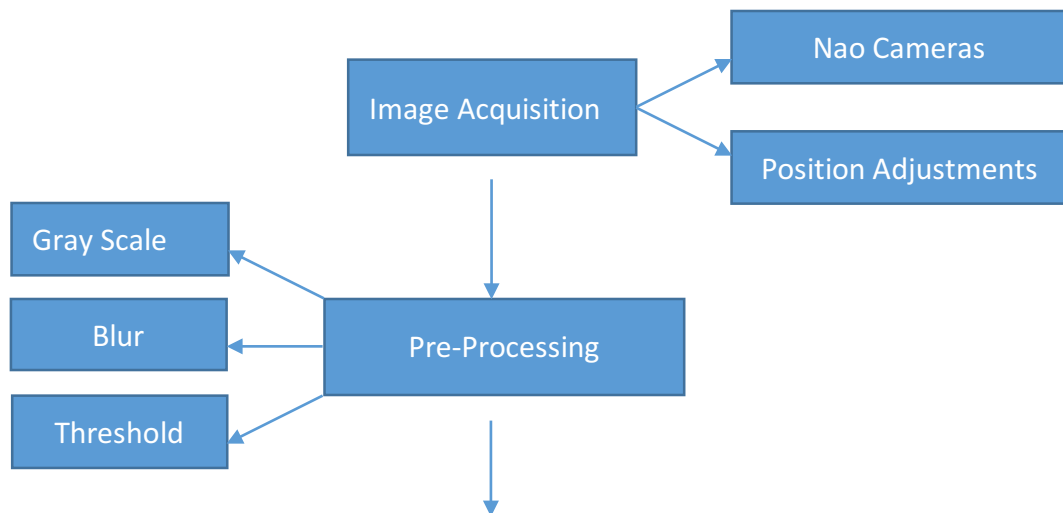
Using the name of the module, the IP and port of a broker. The module must be in the corresponding broker.

4. VISION MODULE

In order to let *Nao* playing cards successfully, it is necessary to make *Nao* understand information on each card like a human being. Therefore, computer vision technology was used in this section for information extracting.

Computer vision, in general, is a field that includes methods for transferring high-dimensional data from the real world to numerical or symbolic information. Information from images will be determined whether or not the image contains some specific feature that can be extracted and transferred into a language that a computer can be understood. The whole vision module was divided into five parts: pre-processing, feature extraction(segmentation), recognition, comparing as well as decision making.

In details, *Nao* will acquire images through two identical video cameras located in the forehead. And software *Choregraphe* will be used to select cameras so that *Nao* could see the cards on the desk by the bottom view. In the next step, *OpenCV* as well as *Python* language were used to further process, analyze, and understand images.



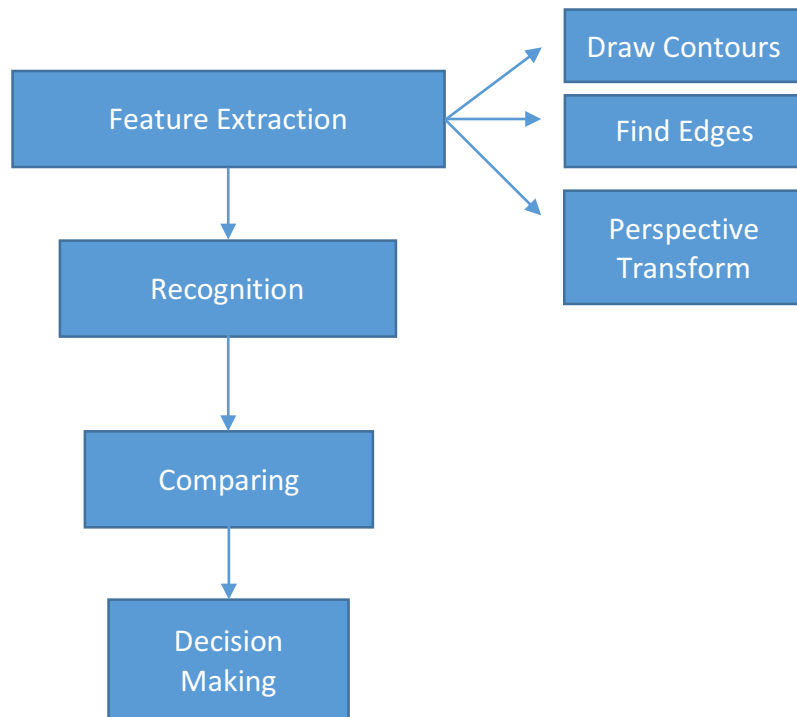


Figure 21. Vision Module Structure

4.1 NAO Vision Introduction

4.1.1 Camera Specifications

There are two identical cameras in Nao head, one is located in the forehead while another one is just inside the mouth. Both of the cameras are providing resolution up to 1280×960 at 30 frames per second. In general, these two cameras are used for image processing like recognizing balls, identifying people, etc.



Figure 22. Location of *Nao* Cameras

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	1.9 μ m*1.9 μ m
	Dynamic range	70 dB

	Signal/Noise ratio (max)	37dB
	Responsivity	2.24V/Lux-sec (550 nm)
Output	Camera output	<i>1280*960@30fps</i>
	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

Figure 23. Camera Specifications

4.1.2 Vision Range

Each Nao camera provides a vertical vision range of 47.64 degrees as well as a horizontal vision range of 60.97 degrees. Two Figures shown below illustrate the vision range of two cameras.

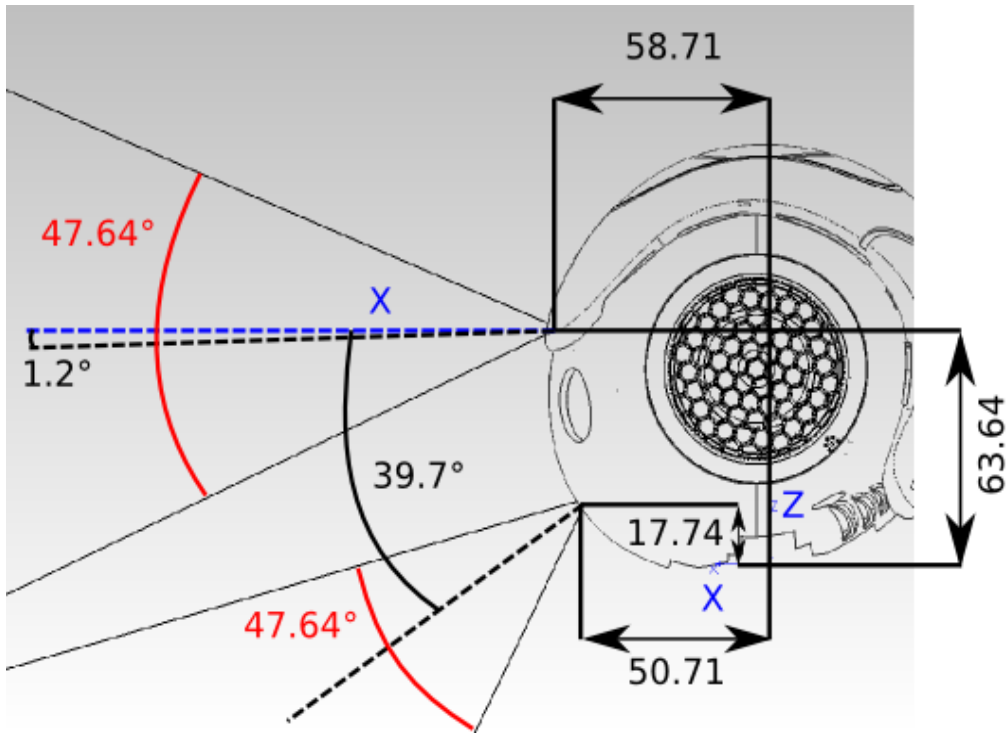


Figure 24. Vision Range

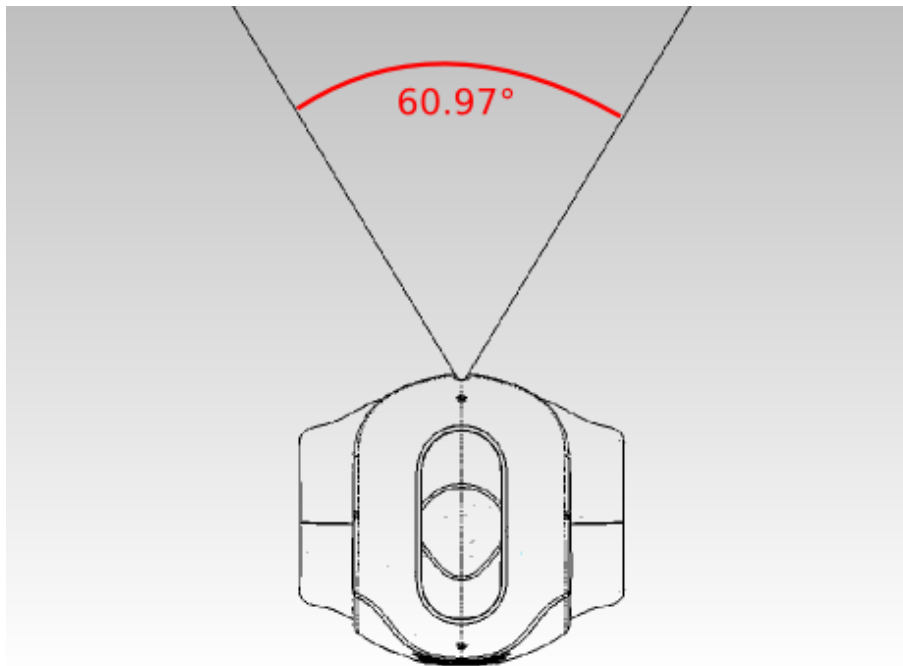


Figure 25. Horizontal Range

4.2 Image Acquisition

4.2.1 Original Images

To start with, Nao will sit behind the desk with the safety posture “REST” like follows.



Figure 26. Initial Position

The initial view of *Nao* is shown in Figure 27. It can be seen from the display window that *Nao* cannot capture card images due to the limited vision ranges.

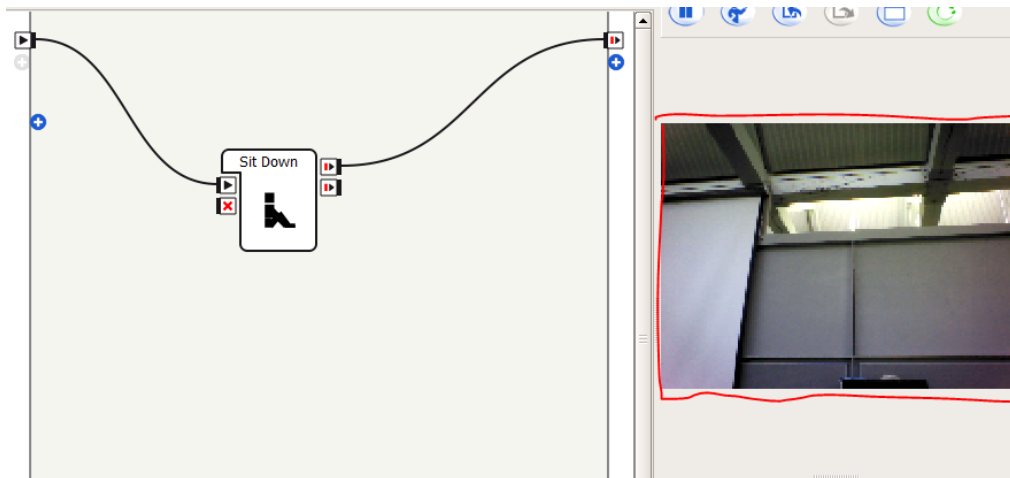


Figure 27. Initial View of *Nao*

4.2.2 Head Adjustments

To achieve our goals, the relations between angles and distances have been calculated as below. Through Figure 28, we can assume that *Nao* can see a distance range from 20 at least to 120 centimeters with the bottom camera if *Nao* head stand vertically.

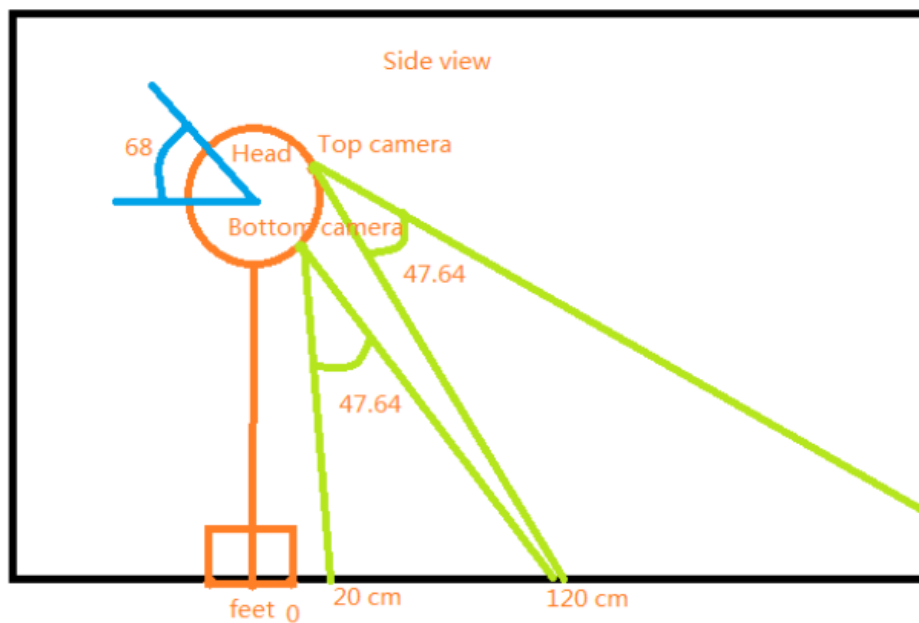


Figure 28. *Nao* Vision Range

Therefore, the angle of *Nao HeadPitch* has been modified as follows. Figure 24 shows the final view after modification.

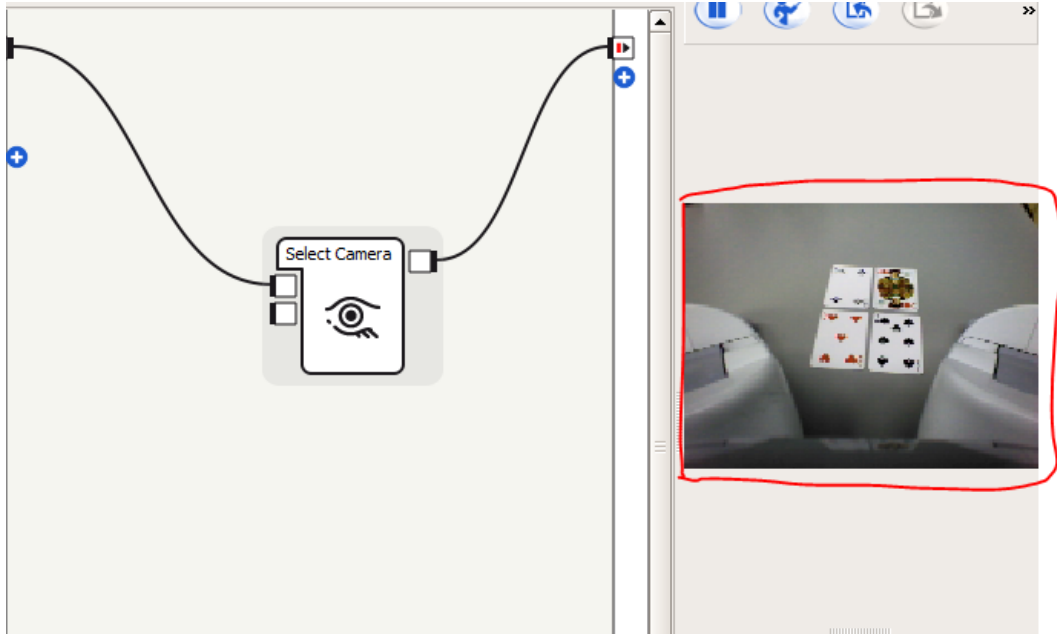


Figure 29. *Nao's View After Adjustments*

4.2.3 Image From *Choregraphe*

In this section, *camProxy*, an object, was created for image acquisition by connecting *ALProxy* module with *Nao IP* and *Port*. The initial image received from the camera was *VGA (Video Graphics Array)* and *RGB (combinational color of Red, Green, Blue)*. Besides, the resolution of the original image is 1280×960 pixels. In other words, the image captured from *Nao* camera first became a two dimensional array, then through inside functions from *Naoqi*, the image data is passed as an array of *ASCII* chars. Moreover, the time for image transfer was shown to record the delay.

```

"""
First get an image from Nao, then show it on the screen with PIL.
"""
camProxy = ALProxy("ALVideoDevice", IP, PORT)

resolution = 3    # 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)

```

Figure 30. Python Code

In the next step, we got the image size with height and width, as well as the pixel from the *ASCII* array returned from last step. Through the size and pixel, the initial RGB image could be transferred to a PIL image and finally converted into the BGR image through OpenCV function `cv2.cvtColor`. The reason behind this scene is that many manipulations will rely on a RGB image and a RGB image are easier for computers to manipulate.

```

# 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)

# Create BGR cv2 image using the PIL image
cvImage = np.array(im)
cvImage = cv2.cvtColor(cvImage, cv2.COLOR_RGB2BGR)

```

Figure 31. *Nao* Connection through *Python*

4.3 Pre-Processing

Before we really extract the features on each card, pre-processed each image would be necessary. This is because the majority of noises can be removed.

4.3.1 Change Color-spaces

There are more than 150 color-space conversion methods available in *OpenCV*. However, in this thesis, the most widely used one is *BGR* to *Gray* since cards are not identified by color.



Figure 32. Gray Image

4.3.2 Smoothing Images

Aiming at blurring the images with various low pass filters and applying filters further to images, image blurring function was used to remove most of the high frequency noises. There are four types of blurring techniques in *OpenCV* --- *Averaging*, *Gaussian Blurring*, *Median Blurring*, as well as *Bilateral Filtering*. Here we will introduce the two mainly used ones:

- **Averaging**

This method simply takes the average of all the pixels convolving the image with a normalized box filter. A 3×3 normalized box filter would like this:

$$K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- **Gaussian Blurring**

Instead of a box filter, Gaussian blur allows customize:

- The width and height of the kernel which should be positive and odd.
- Standard deviation in X and Y direction, σ_X and σ_Y respectively.

Since this method is highly effective in removing Gaussian noise, it is used for noise filtering in our case.



Figure 33. Blurring Image

4.3.3 Image Thresholding

In order to focus on the surface of cards, artifacts were gotten rid of through the simplest segmentation method in OpenCV --- image thresholding.

The basic design is straight forward. If the pixel value is greater than the threshold value, it is assigned to black in our case, otherwise it will appear white. *OpenCV* provides five different styles of thresholding:

- `cv2.THRESH_BINARY`,
- `cv2.THRESH_BINARY_INV`,
- `cv2.THRESH_TRUNC`,
- `cv2.THRESH_TOZERO`,
- `cv2.THRESH_TOZERO_INV`

In this case, cv2.THRESH_BINARY was used. Moreover, the source image should be a gray scale image. As we can see from figure 34 shown below, most of the features were separated from the rest region.

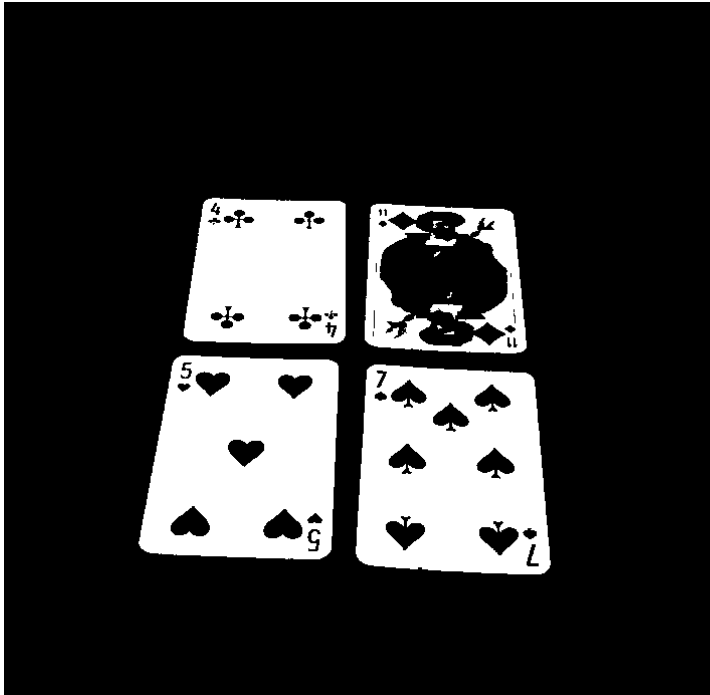


Figure 34. Threshold Image

4.4 Feature Extraction

4.4.1 Draw Contours

To be more specific and clear, we continued to draw all the contours in the image including the edges as well as the content shown on the cards. Thus, it can be clearly seen from Figure 33 that all features of the cards have been separated from the rest areas. Most importantly, after the manipulation, it can be clearly estimated from the picture that four rectangular edge regions correspond to the surface of the cards.

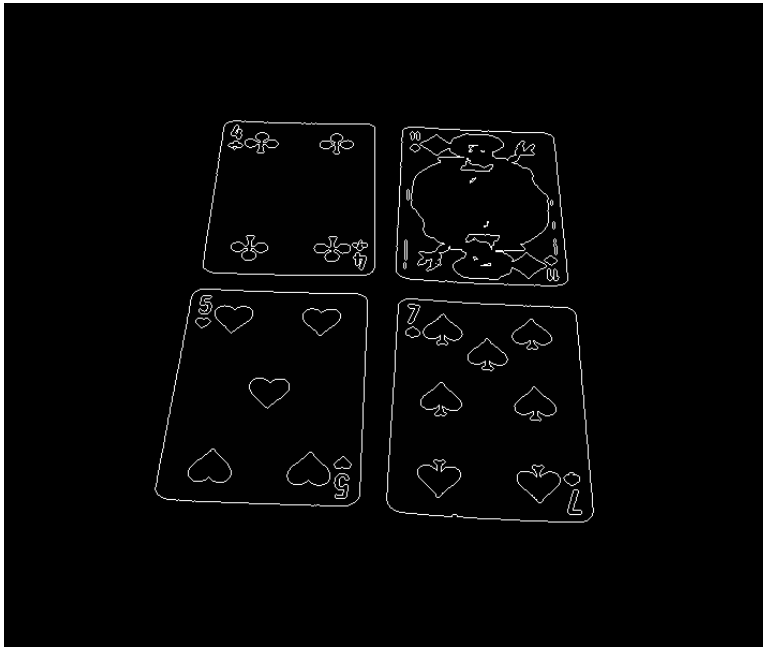


Figure 35. Contours of Cards

First, the preprocessed image is passed to OpenCV contour methods to calculate the hierarchy between contours and return a list of contours that has been found.

4.4.2 Find Edges

In order to find each card, the edge of each card was detected so that the size of card can be identified, thus *Nao* can recognize it.

In the first place, the numbers of contours are declared that need to be detected. As known, the surface of each card is relatively large compared to the rest regions. So the contours received from previous step can be sorted by calculating the area of each contour from the largest to the smallest. Then the four cards in the image can be estimated by finding the largest four contours (Figure 36).

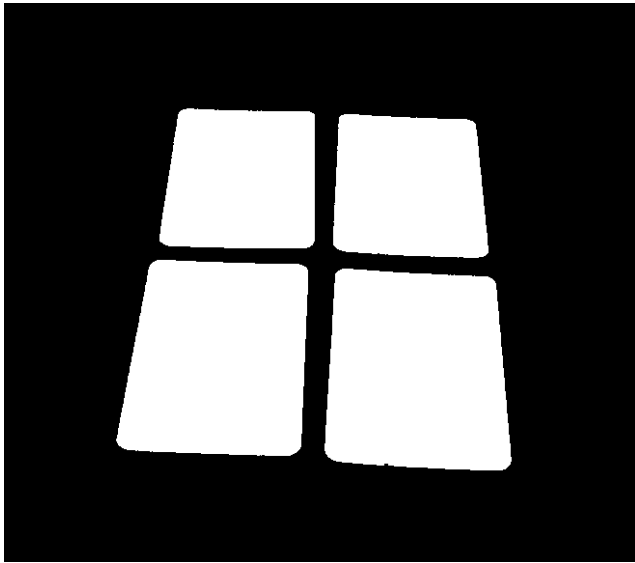


Figure 36. Find Areas of Cards

Next, those four largest contours are looped over to find the exact edges of the four cards. In order to approximate each contour, the polygonal curves of a contour are estimated by applying an approximation precision of each card edge. In this project, 2% of the perimeter of the contour are used. Moreover, four points are used to identify the edges. This is because only four points in the corner are needed to determine a rectangular which can greatly reduce the numerical calculation.

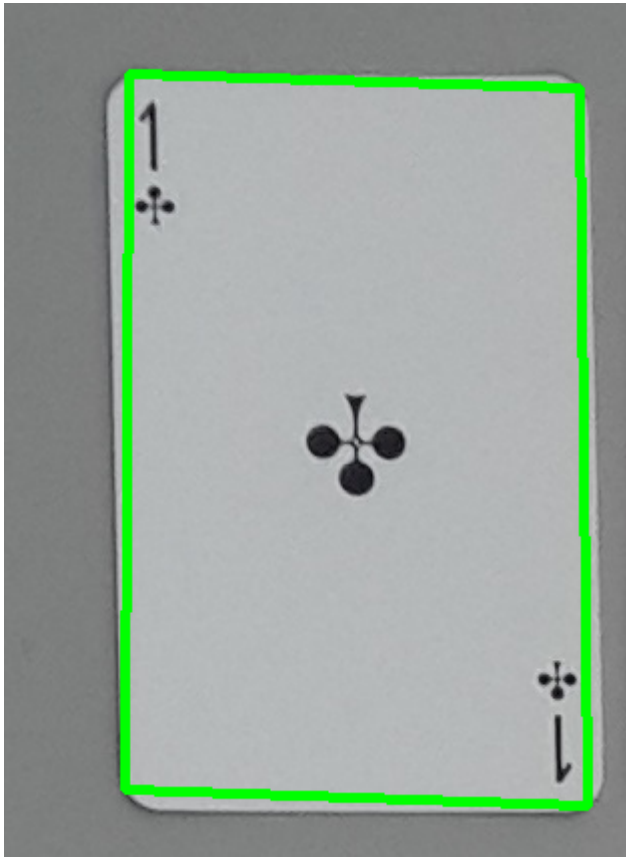


Figure 37. Edges of Cards

4.4.3 Perspective Transform

The basic algorithm for object detection is creating a model to teach computers learning from those images. Each image is represented as a two dimensional array known as pixels, therefore, every single difference of lights or positions would be identified as a totally different array. From the Figure 37, we can see that each card is not in the top-down view of the image because of perspective problems.

To solve this problem, for each rectangle, we performed a perspective transform. As a result, each cards on the image could be covered and segmented individually for card recognition. Then each card will be registered into a rectangular representation before further processing.

In previous section, a polynomial from the contour was approximated and a bounding box was found, moreover, a list of four points of each rectangular was returned. In this section, those (x, y) coordinates were sorted in a top-left, top-right, bottom-right as well as bottom-left order. The reason behind this scene is that the top-left point has the smallest sum of $x+y$ while the bottom-right point associates with the largest sum. As for the other two points, it is found that the top-right points has the smallest difference which is $x-y$, with the bottom-left point relating to the largest difference. By performing above NumPy numerical processing, a consistent ordering of points is gained, otherwise the perspective image may be twisted.

In the last step, the dimension of the new warped image was specified. The width of each card is the largest distance of x -coordinates between the bottom-right and bottom-left or top-right and top-left. Similarly, the height of each card is the largest distance of the y -coordinates between top-right and bottom right or the top-left and the bottom-left.

Therefore, each points are picked in order to obtain a perspective image of each card.

The following figures 38 and 39 show the results:



Figure 38. Perspective Transform



Figure 39. Separate Each Card

4.5 Recognizing Cards

There are 52 cards in total except for the two joker cards. After having cropped registered representations of each card, each cropped image was saved into the directory in order.

An approach was used in this part that connects the digits of each card with the file names. To be specific, there are four different card suits in total: *Spades*, *Hearts*, *Diamonds* and *Clubs* respectively (Figure 40). Each suit contains the same number of cards, which is 13 cards each and 52 cards in total (Figure 41). Cards of the same suit were saved together in the order of 0 to 12 with the corresponding file names like Figure 42 below shows. For instance, if a card is *Heart* 1, its file name would be “13.jpg”. If a card is *Spades* 3, its file name would be “2.jpg”.



Figure 40. Four types of Cards

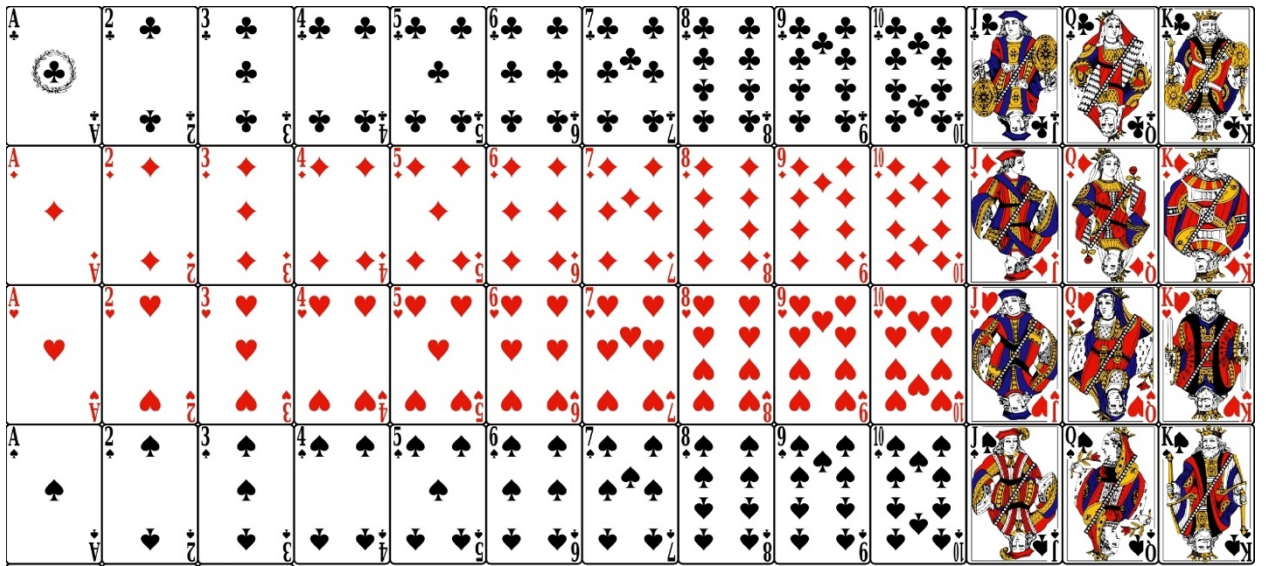


Figure 41. Representation of 52 Cards

Card Suits	Corresponding File Names	Corresponding Card Numbers
Spade	From 0 to 12	From 1 to 13
Heart	From 13 to 25	From 13 to 26
Diamond	From 26 to 38	From 27 to 39
Club	From 39 to 51	From 40 to 52

Figure 42. Relations Between Card Numbers and File Names

4.6 Comparing Differences

Now that *Nao* knows what a single card looks like, how does *Nao* specify the card number with the incoming candidate card?

In the thesis, the candidate card is matched with every recognized card by simply comparing two cards through previous contours. To be specific, all the input images, are firstly going to be preprocessed to have a final contour look. Noises as well as artifacts are gotten rid of in this process. Secondly, and most importantly, an absolute difference of one image from another will be performed as Figure 43 shows. Finally, the sum of intensity of each pixel which is not the same is returned.

Figure 43 shows the difference when matching Heart 2 with Heart 5.

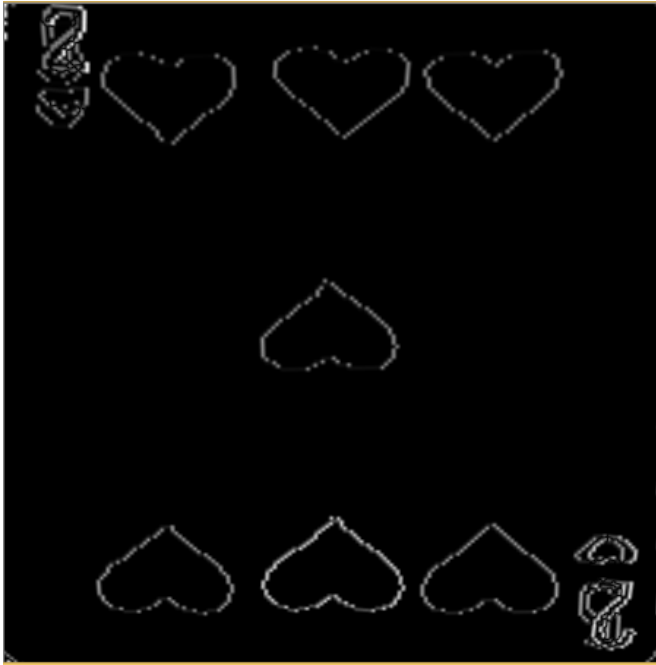


Figure 43. Difference between Heart 2 and Heart 5.

Figure 44 shows the similarity when matching Heart 2 with itself.

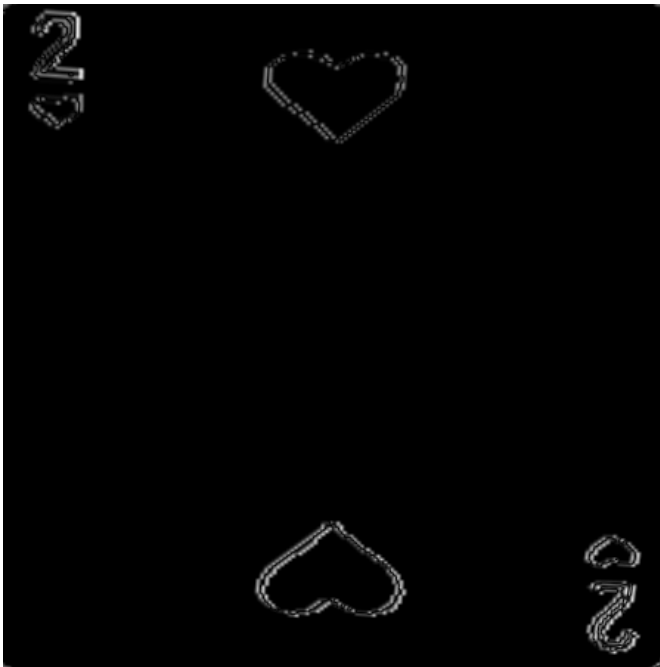


Figure 44. The Similarity of Comparing Heart 2 with Itself

Therefore, matching each card becomes a simple process of comparing the incoming card against the recognized cards, and looping over it to find the minimum difference.

Once the minimum difference was found, the corresponding filename of that card can be easily accessed. Because the looping time will be recorded and the filename is same with the looping time based on the naming rule in Section 4.5 “Recognize Cards”.

Figure 46 shows the final results of recognizing cards after comparing differences with the database:

The original cards are *Heart 2*, *Heart 3*, *Spades 11*, *Spades 12*.



Figure 45. Original Cards

```
>>> ===== RESTART =====
>>>
Loop: 1
Card Number: 2
Loop: 2
Card Number: 3
Loop: 49
Card Number: 11
Loop: 50
Card Number: 12
>>>
```

Figure 46. Python Results of Recognizing Cards

5. NAO BEHAVIORS

In the beginning, Nao will take a card from the dealer and put in the front desk with his right arm. After he receives two cards, object recognition will be started and the result is announced.

5.1 Joints Introduction

Nao body is assembled with several pieces of joints which link the body parts of the robot together. To perform the rotation of the body parts, a frame was placed at each joint as shown in Figure 47. The *X* axis takes charge of roll movements while pitch rotations take place around the *Y* axis and yaw rotations around the *Z* axis.

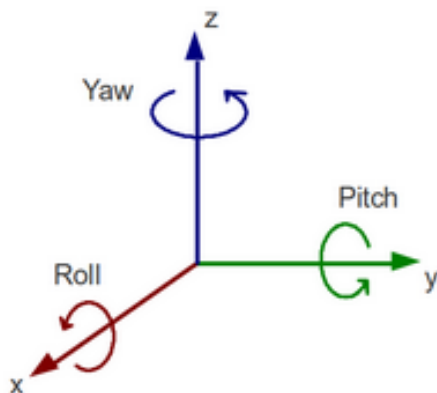


Figure 47. *Nao* joint rotation frame

5.1.1 Head Joints

Two joints are controlling *Nao*'s head movement, *HeadYaw* and *HeadPitch* respectively. *HeadYaw* is in charge of moving left and right from 119.5 to -119.5 degrees while *HeadPitch* takes care of moving *Nao*'s head forward and backward from 29.5 to -38.5 degrees. The actual value can be received from the sensor by using *ALMemory* key name. Figure 44 shows the basic information about these two joints.

Joint name	Motion	Range (degrees)	Range (radians)
HeadYaw	Head joint twist (Z)	-119.5 to 119.5	-2.0857 to 2.0857
HeadPitch	Head joint front and back (Y)	-38.5 to 29.5	-0.6720 to 0.5149

Figure 48. Nao Head Joints

And Figure 49 illustrates the range these two joints enjoy.

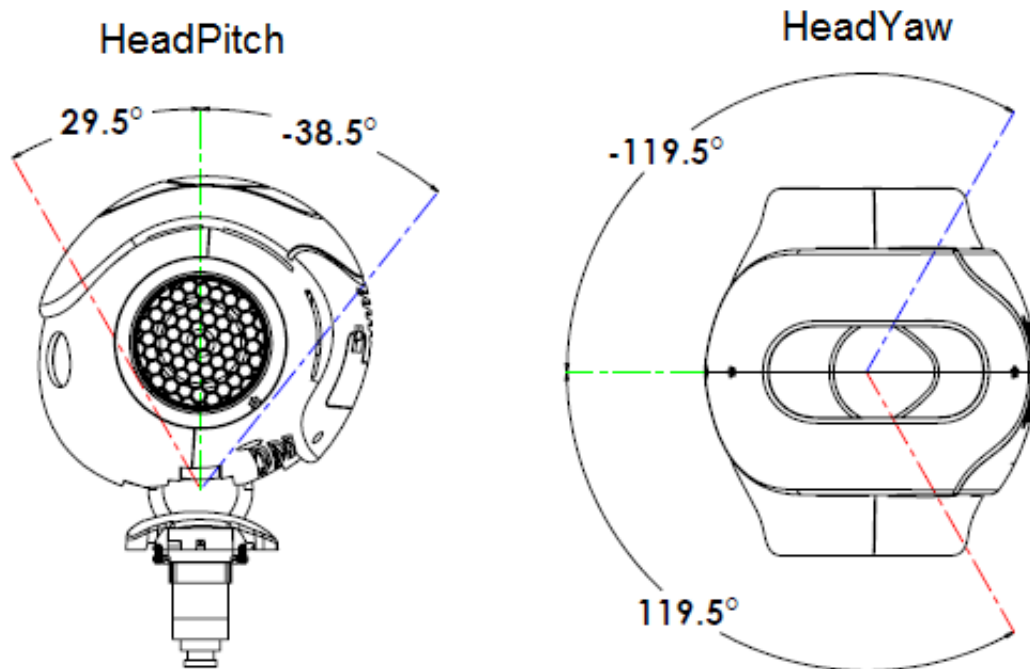


Figure 49. Head Joints Movement

5.1.2 Arm Joints

There are two arms installed on *Nao* assembled human arms---left arm and right arm. Each arm has five actuators, *ShoulderPitch*, *ShoulderRoll*, *ElbowYaw*, *ElbowRoll*, *WristYaw* respectively. The actual value can be received from the sensor by using *ALMemory* key name.

For each joint, the *ShoulderRoll* controls move shoulder to its side, and the *ElbowRoll* take charge of lifting the elbow in the vertical direction while the *ElbowYaw* as well as the *WristYaw* moves in X direction. Moreover, the *ShoulderPitch* take care of shoulder movements in Y direction.

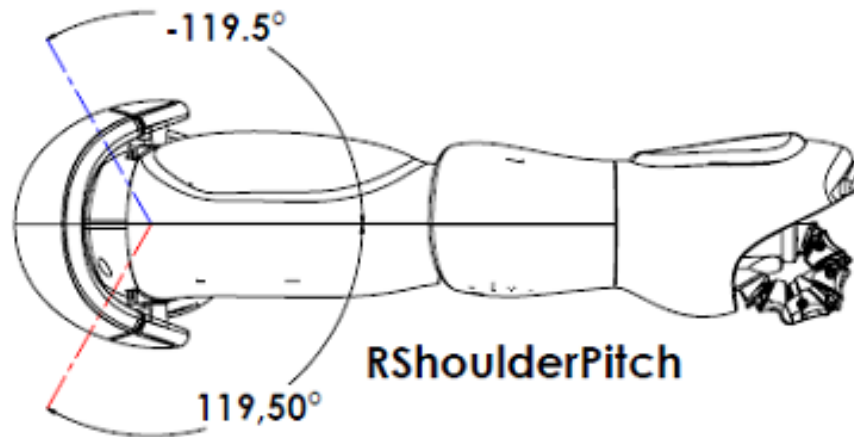
The following figure 50 shows the basic information about the right arm joints.

Joint name	Motion	Range (degrees)	Range (radians)
RShoulderPitch	Right shoulder joint front and back (Y)	-119.5 to 119.5	-2.0857 to 2.0857
RShoulderRoll	Right shoulder joint right and left (Z)	-76 to 18	-1.3265 to 0.3142
RElbowYaw	Right shoulder joint twist (X)	-119.5 to 119.5	-2.0857 to 2.0857
RElbowRoll	Right elbow joint (Z)	2 to 88.5	0.0349 to 1.5446
RWristYaw	Right wrist joint (X)	-104.5 to 104.5	-1.8238 to 1.8238

Joint name	Motion	Range (degrees)	Range (radians)
RHand	Right hand	Open and Close	Open and Close

Figure 50. The Specifications of Right Arm

Figure 51 illustrates the range these joints enjoy.



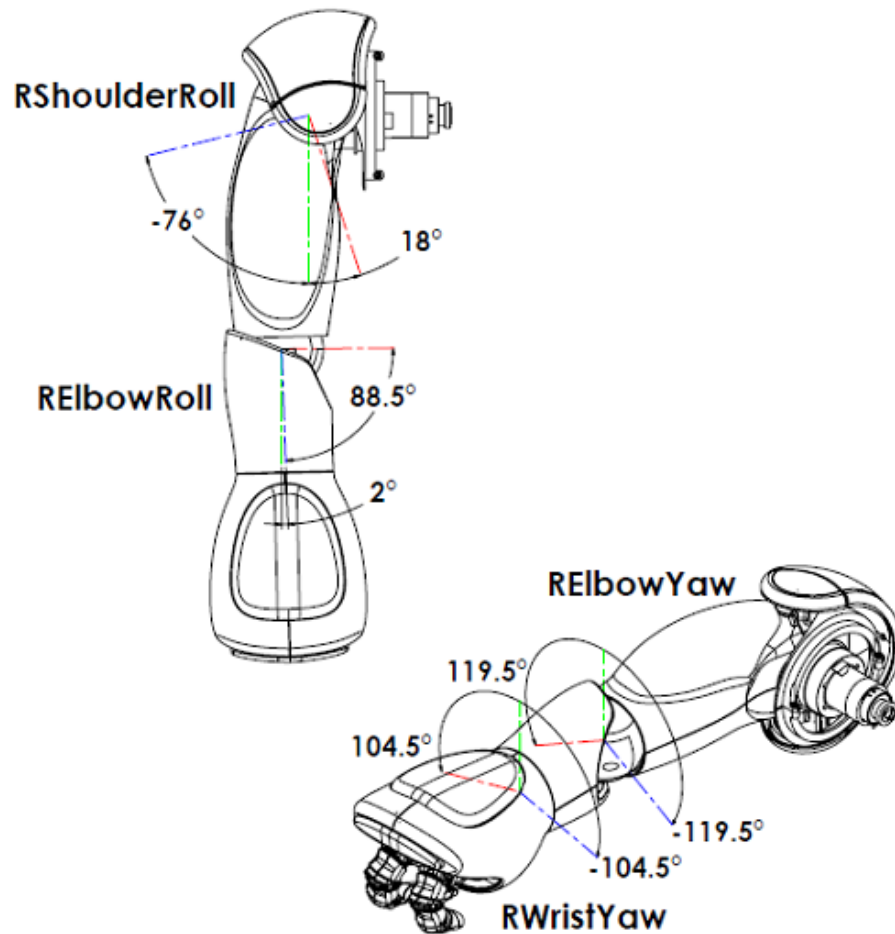


Figure 51. Arm Joints Movements

5.2 Nao Behaviors

In this section, inverse kinematics and *Naoqi* functions from *Proxy* are used to implement *Nao*'s right arm control.

5.2.1 Animation

First, a series of movement through "*Animation mode*" are designed inside *Choregraphe* software.

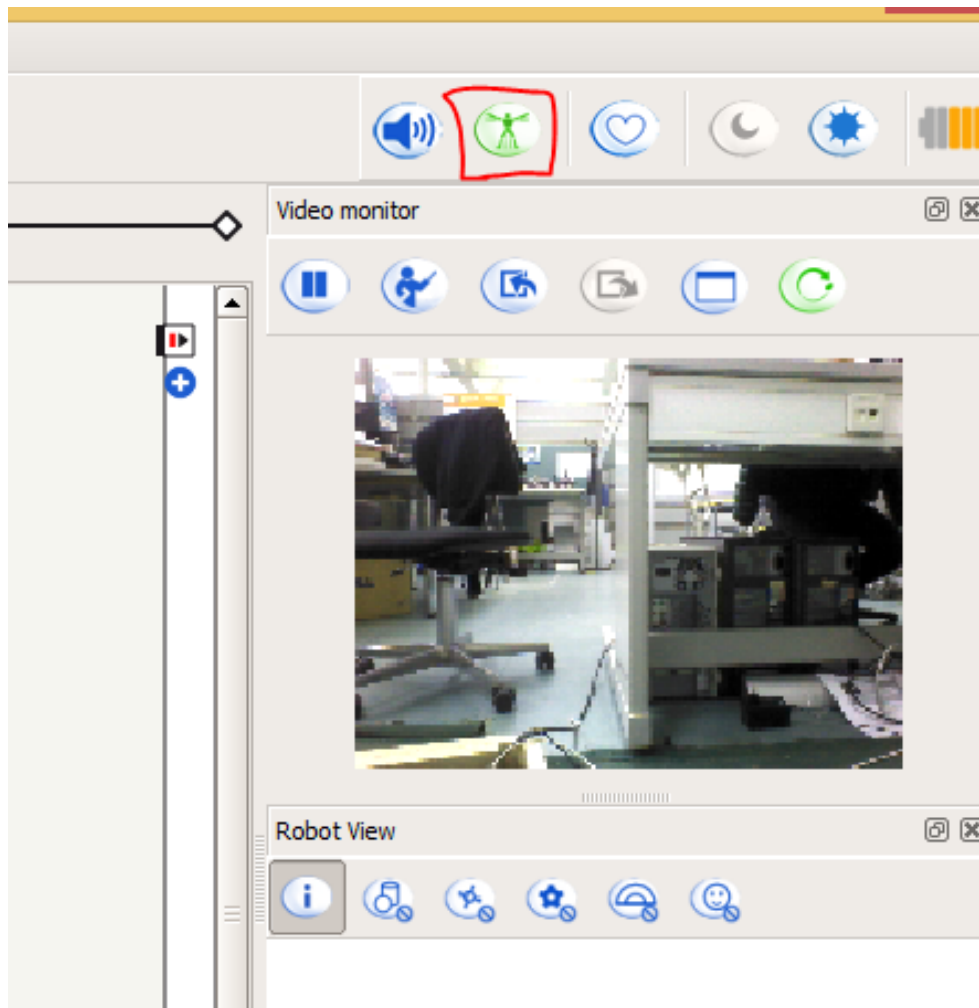


Figure 52. Animation Logo on *Choregraphe*

The whole animation is divided into two sections:

➤ **Waiting for a card**

Step 1: Nao starts with a rest position and set relative stiffness on.

Step 2: Lift right arm to the middle length of the body. The reason behind this motion is that Nao can not move the arm directly to the front position because the table in the front might hit Nao's arm. Then, from the middle length of the body move horizontally to the front of the body.

Step 3: With the splayed fingers, Nao asking for a card from the dealer.

Step 4: Close the fingers and hold the incoming card tightly.

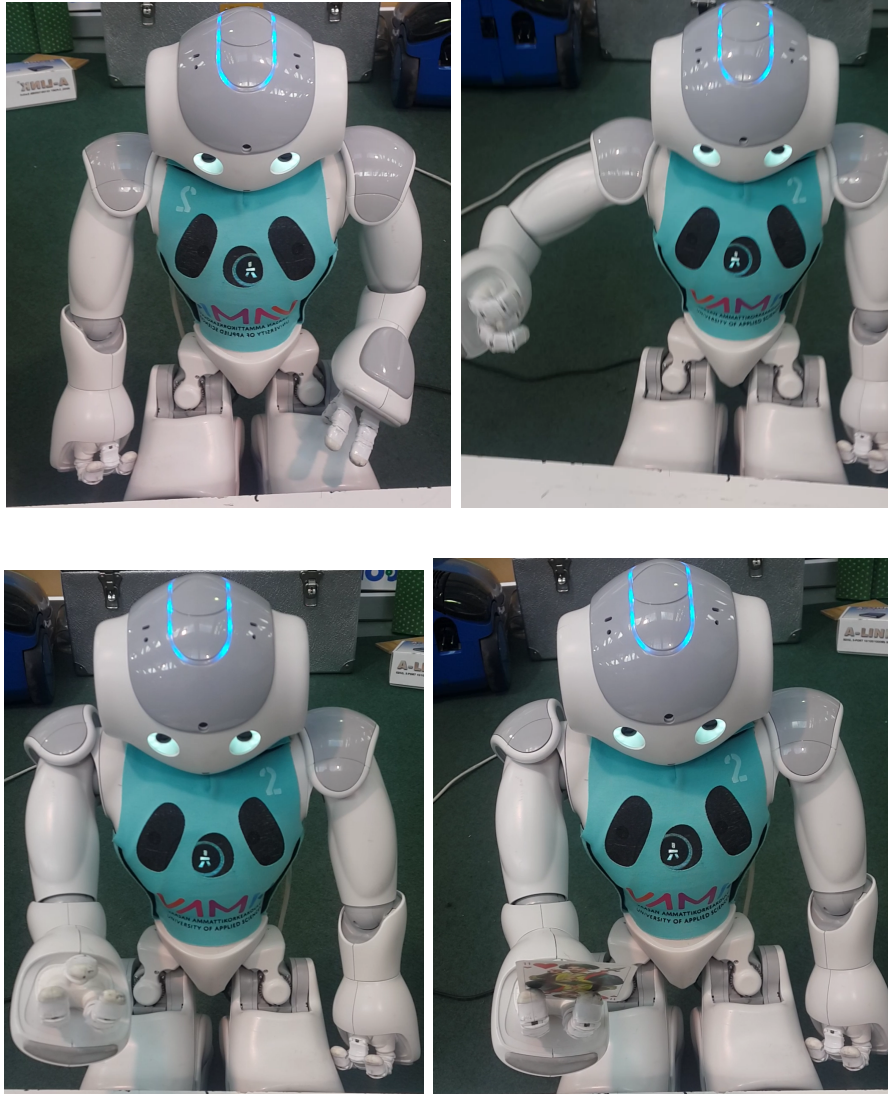


Figure 53. First Part of Animation

➤ **Put it on the table**

Step 1: Arrive the targeted position and rotate the wrist while still holding the card.

Step 2: Open the fingers and let the card lay on the table.

Step 3: Lift the right arm to the middle length of the body and back forward to the safety position.

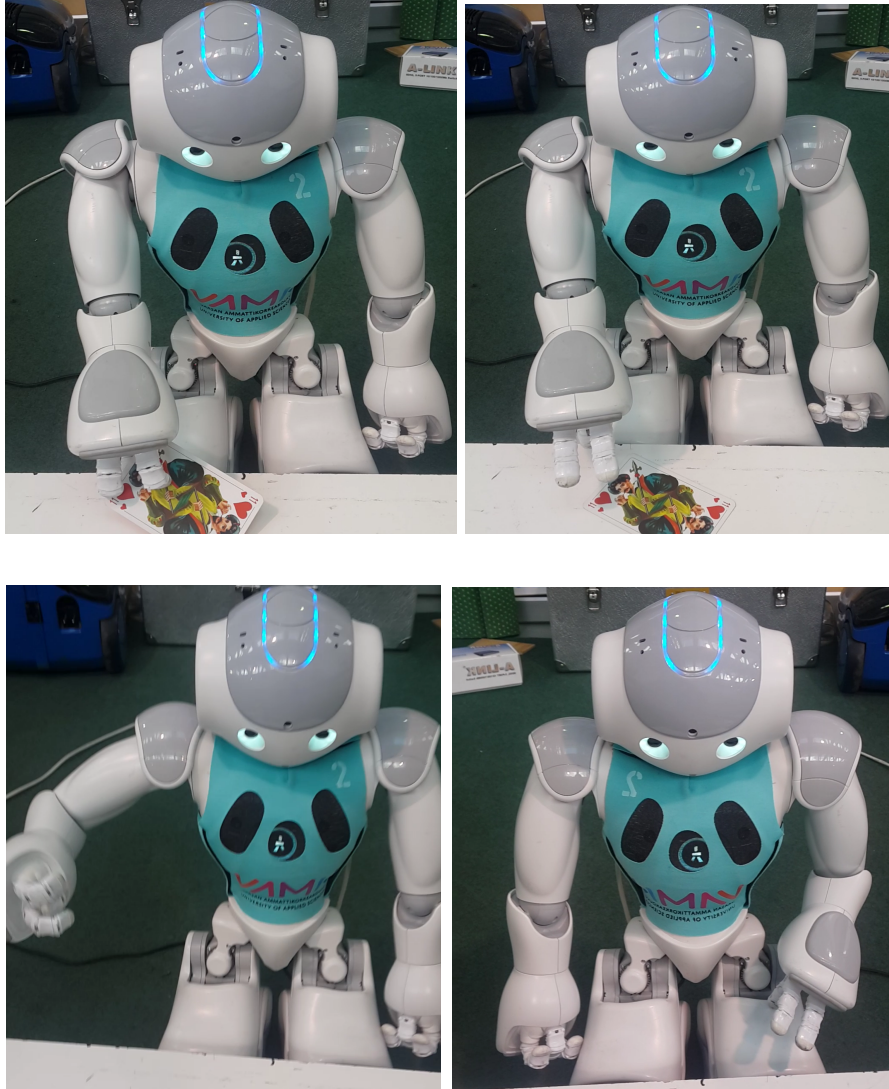


Figure 54. Second Part of Animation

5.2.2 Get Angles

Next, angles of *Nao* joints at desired positions are gained through *getAngles* function. For the purpose of accuracy, two angles---sensor angles and command angles---were compared and shown errors on an *IDLE* screen.

5.2.3 Set Angles

Then, angles of the joints are set through function *setAngles* from *motionProxy*. There are three parameters in the function:

- Name

The name or names of joints, chains, “*Body*”, “*JointActuators*”, “*Joints*” or “*Actuators*”.

- Angles

One or more angles in radians.

- FractionMaxSpeed

The fraction of maximum speed to use.

5.2.4 Arm Move

In order to control Nao’s arm, Function *positionInterpolation* moves an end-effector to the given position and orientation over time. This function includes six parameters:

- Space

When executing a task, the space is determined when the task begins, and remains constant throughout the rest of the interpolation.

There are three task spaces, `FRAME_TORSO = 0`, `FRAME_WORLD = 1`, `FRAME_ROBOT = 2`

FRAME_TORSO: this is attached to NAO’s torso reference, and it moves with NAO as he walks and changes orientation as he leans. This space is useful when there are very local tasks that make sense in the orientation of the torso frame.

FRAME_WORLD: this is a fixed origin that is never altered. It is left behind when

NAO walks, and will be different in z rotation after NAO has turned. This space is useful for calculations which require an external, absolute frame of reference.

FRAME_ROBOT: this is average of the two feet positions projected around a vertical z axis. This space is useful, because the x axis is always forwards, and it provides a natural ego-centric reference.

- Effector Name

Effector name	Position	End transform
“Head”	At the neck joint	Position3D(0.0, 0.0, 0.0)
“LArm”	Inside the hand	Position3D(HandOffsetX, 0.0, -HandOffsetZ)
“LLeg”	Below the ankle	Position3D(0.0, 0.0, -FootHeight)
“RLeg”	Below the ankle	Position3D(0.0, 0.0, -FootHeight)
“RArm”	Inside the hand	Position3D(HandOffsetX, 0.0, -HandOffsetZ)
“Torso”	A reference point in the torso	Position3D(0.0, 0.0, 0.0)

- Path

Vector of 6D position arrays (x,y,z,wx,wy,wz) in meters and radians. x is to the front of the robot, y to its side while z is the elevation. Then comes the rotation: rotation around the x-axis, rotation around the y-axis and z-axis.

- AxisMask

It can be defined which axis to use by an Axis Mask. For example, 7 for position only, 56 for rotation only and 63 for both.

```
#define AXIS_MASK_X 1
#define AXIS_MASK_Y 2
#define AXIS_MASK_Z 4
#define AXIS_MASK_WX 8
#define AXIS_MASK_WY 16
#define AXIS_MASK_WZ 32
```

Figure 55. AxisMask Specifications

- Durations

Vector of times in seconds corresponding to the path points.

- IsAbsolute

If true, the movement is absolute otherwise it is relative.

By function *positionInterpolation*, most of the arm movements can be achieved. 3D space is used for positions instead of 6D since it is enough for the thesis. The time interval is one second for each movement. A piece of code is shown below in Figure 47.

```

effector = "LArm"
space     = motion.FRAME_ROBOT
axisMask = motion.AXIS_MASK_VEL
isAbsolute = False
#T
pathList = [
[ +0.00, +0.00, -0.005, 0.0, 0.0, 0.0],
[ +0.00, +0.035, -0.005, 0.0, 0.0, 0.0],
[ +0.00, +0.035, +0.05, 0.0, 0.0, 0.0], #up
[ -0.01, +0.012, +0.05, 0.0, 0.0, 0.0], #upmove
[ -0.01, +0.012, -0.01, 0.0, 0.0, 0.0],
[ -0.04, +0.010, -0.01, 0.0, 0.0, 0.0],
[ -0.04, +0.010, +0.05, 0.0, 0.0, 0.0] #up
]

times      = [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0] # seconds

motionProxy.positionInterpolation(effector, space, pathList,
                                   axisMask, times, isAbsolute)
time.sleep(1.0)

```

Figure 56. Code for Nao Arm

5.3 Results Announcement

After Nao puts cards in a desired position and recognizes the cards, Nao speaks out results when a round comes to an end using a proxy.

ALProxy is an object that can connect to all methods. There are three parameters in *ALProxy* class.

- Name --- name of the module
- IP --- IP of the robot
- Port --- the port on which NAOqi listens (9559 by default)

A piece of code is shown in Figure 48:

```

def speakDecision(self, decision):
    if decision==HIT:
        self.tts.say("Hit")
    elif decision==STAND:
        self.tts.say("Stand")
    elif decision==LOSE:
        rnd = random()
        if rnd<0.1:
            self.tts.say("The battle is lost. But the war has just began!")
        elif rnd<0.4:
            self.tts.say("I lost this one, but It is not how hard you hit. It's how hard you get hit and keep moving forward.")
        elif rnd<0.6:
            self.tts.say("I lost, but people say: Lucky at cards, unlucky in love. Fortunately, girls love me!")
        elif rnd<0.7:
            self.tts.say("I hate losing. Second place doesn't interest me. I have fire in my belly!")
        else:
            self.tts.say("If I could speak Macedonian, I would say some really really! mean words!")
    elif decision==WIN:
        rnd = random()
        if rnd<0.3:
            self.tts.say("Try losing some weight, not just games in Black Jack!")
        elif rnd<0.4:
            self.tts.say("You lost again. Please level up, you are to week for me!")
        elif rnd<0.5:
            self.tts.say("Stop losing. Try using the force next time.")
        elif rnd<0.6:
            self.tts.say("It seems like you really like losing. If you like it, you should put a ring on it.")
        elif rnd<0.7:
            self.tts.say("Me and Charlie Sheen must be related. Hashtag: winning.")
        else:
            self.tts.say("Who you gonna blame now for losing? Branko?")

```

Figure 57. Python Code for Saying

6. BLACKJACK ALGORITHMS

In previous sections, *Nao* could compare the coming card with the database and figure out what digit the card represents for. In this section, the *Blackjack* game strategy is introduced.

Balckjack(BJ), also known as *twenty-one*, is the most popular gambling game throughout world. It is made up of 22 cards except 2 joker cards. The basic rule of *BJ* is that *Nao* comparing the sum value of two incoming cards with 21. If *Nao* gets 21 points directly (called *BJ*), *Nao* will win the round. Otherwise, *Nao* will lose the round.

7. IMPROVEMENTS

As for further improvements, comparing the coming cards with the database could be replaced by *Artificial Neural Network (ANNs)*.

ANNs is a computational or mathematical model that simulates the structures and functions of biological neural networks, particularly the brain of animals. It is used to estimate functions through amounts of unknown inputs.

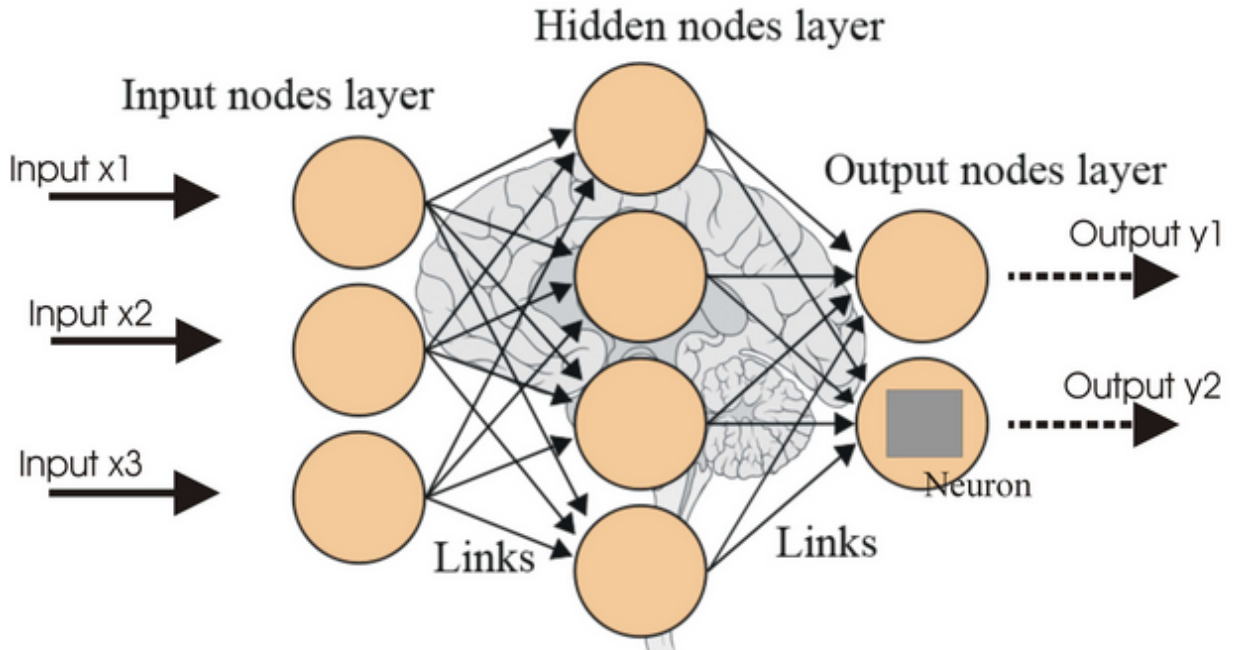


Figure 58. Artificial Neuron Network

The mechanism of *ANNs* is exchanging information between each other through large number of interconnected “*neurons*” as figure 50 shows. Each node (*neurons*) represents a specific output function with its value “1” or “0” called *Activation Function*. And each connection between two nodes stands for a weighted value for the connection signal called *Weight*. *Weight* is equivalent to the artificial neural network

memory. The output of the network gains different *Weights* and *Activation Functions* according to various network connections. The network itself represents the approximation of some algorithms in the nature or some expression of a logical strategy.

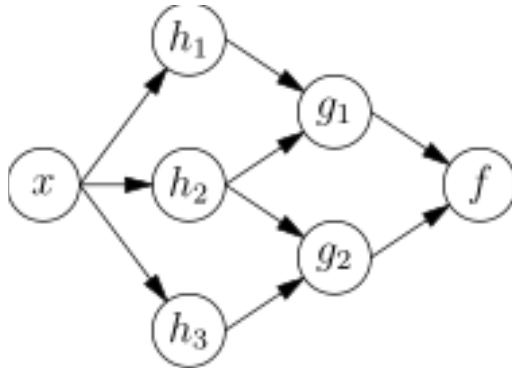


Figure 59. Neuron Nodes

So far, the computer still has not matched or even surpassed human capabilities, we have just made some progress in teaching the computer to see objects. This is like a small child learning to utter a few nouns. It is an incredible accomplishment, but it is only the first step. Soon, another developmental milestone will be hit, to teach a computer to see a picture and generate sentences, then the marriage between big data and machine learning algorithm has to take another step. Now, the computer has to learn from both pictures as well as natural language sentences generated by humans. Just like the brain integrates vision and language, in this thesis a model was developed that connects parts of visual things like visual snippets with words and phrases in sentences.

However, progress is made little by little as giving sight to the machines. First, we teach them to see. Then, they help us to see better. For the first time, human eyes won't be the only ones pondering and exploring our world. We will not only use the machines for their intelligence, we will also collaborate with them in ways that we cannot even imagine.

8. CONCLUSION

This thesis introduces computer vision as well as the behavior design of a humanoid *Nao* robot mainly based on *OpenCV* and *Python* language: *Nao* playing *Blackjack*.

To begin with, the whole process is basically divided into three parts: receiving cards, recognizing cards and making decisions. In the first part, *Nao* will receive cards from the rack and put it in the front. After finding locations, function *positionInterpolation* was used for controlling *Nao*'s arm to specific positions. The initial position of the arm was obtained by function *getAngles* mainly, and the latter positions were gained through the first position relatively.

In the second part, *Nao* processes the image of the cards and tries to recognize these cards. *OpenCV* first preprocesses the image received, then draws contours of cards and sorts those points by areas. Thus four cards on the image are found through detecting edges. Next, a "bird-eye view" will be applied to each card. This is because the positions of each card are different and those cards may be placed in different angles or directions, definitely not just in a rectangle in front of *Nao*'s eyes. After detected edges of each card, *Perspective Transform* is performed on each card. Therefore, the real information on each cards are processed without other useless information. Finally, *Nao* compares the processed image with the database, finds the most similar card together with its file name. Then through an algorithm, *Nao* knows the number of the cards.

In the last part, *Nao* makes decisions based on *Blackjack* rules. Three options, hit, stand, split will be performed and *Nao* will speak out the results loudly after each round.

After completing this thesis, my ability of self-learning improved very much since everything was totally new for me, especially the computer vision part, *OpenCV*. However, this is what life calls for. Different approaches have been made, such as

reading documentation, consulting experts, etc. A good point is how important planning a big task into small chapters. By following a schedule, a big task can be divided into small simpler tasks and after persistent work on them, the goal is finally achieved!

REFERENCES

1. Computer vision from Wikipedia
https://en.wikipedia.org/wiki/Computer_vision
2. Nao- Video camera from documentation
http://doc.aldebaran.com/2-1/family/robots/video_robot.html#robot-video
3. OpenCV website
<http://opencv.org/>
4. Choregraphe website
<https://community.aldebaran.com/>
5. Numpy&SciPy explanation
<http://www.engr.ucsb.edu/~shell/che210d/numpy.pdf>
6. Smoothing Images documentation from OpenCV
http://docs.opencv.org/3.0beta/doc/py_tutorials/py_imgproc/py_filtering/py_filtering.html#filtering
7. Image processing documentation from OpenCV
<http://doc.aldebaran.com/2-1/family/robots/bodyparts.html>
8. Balckjack rules
<https://en.wikipedia.org/wiki/Blackjack>
9. Artificial Neuron Network
<https://zh.wikipedia.org/wiki/%E4%BA%BA%E5%B7%A5%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C#.E5.9F.BA.E6.9C.AC.E7.B5.90.E6.A7.8B>
10. Computer Vision
https://en.wikipedia.org/wiki/Computer_vision