VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Qi Wan

DEVELOPEMENTS OF DRAWING CAPABILITY FOR NAO HUMANOID ROBOT

Technology and Communication

2015

FOREWORD

This is my undergraduate thesis in Information Technology at Vaasan ammattikorkeakoulu, University Applied Sciences. I would like to express my gratitude to my supervisor Dr. Yang Liu to his contribution in this thesis and his helpfulness and professionalism. Thank you so much for giving me the chance to get access to Arti4ficial Intelligence. He really inspired me with so many constructive suggestions.

I would take the opportunity to thank the whole IT department of VAMK, specially Dr. Liu Yang, Dr. Chao Gao, Dr. Ghodrat Moghadampour, Mr. Jani Ahvonen, Mr. Santiago Chavez Vega, Mr. Jukka Matila, Dr. Smail Menani. You guided me and encouraged me during these four years of studying at VAMK.

I am so appreciate the support from my parents and all my friends in Botnia RoboCup laboratory.

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Degree Program in Information Technology

ABSTRACT

Author	Qi Wan
Title	Developments of Drawing Capability for NAO Humanoid Robot
Year	2015
Language	English
Pages	54 + 5 Appendices
Name of Supervisor	Yang Liu

This thesis describes and introduces the drawing capability of robots that can draw based on what it sees. The user draws some simple shapes on the whiteboard which is placed in front of the robot. Consequently the robot observes the drawing, takes a snapshot of it, extracts the features of the subject from the drawing, such as contours and corner points, then calculates the arm movements and lets the robot recreate it on paper as accurate as possible. During the project, an Aldebaran Nao robot generation 5 was used.

In this thesis, the technology called Morphological transformation was used to reduces the noise of the picture and enrich some useful features during erosion phase. The algorithm called Canny edge detection was chosen as the optimal choice to detect the edge of the object in the picture captured by the Nao robot. In addition, Horris Corner Detection was applied to find the corner points of the object which is the main feature of the image. Moreover, a mapping from the image domain to the robot space was proposed.

The programming language using in this project is Python. In addition, OpenCV, together with Python was used to extract the features of the example drawing. Choregraph was used to help to design the animations. NAOqi is version 2.1.

In this application, the Nao robot was successful to reproduce the example drawings which was drew by the experimenter. The drawing capability for the Nao robot can be developed with more useful functions in future. And mathematical theory support turns out to be significant when designing the animations of the robot.

Key words Nao robot, vision system, robotic kinematics, animation design

CONTENTS

LIST OF ABBREVIATIONS

PC	Personal Computer
RoboCup	Robot Soccer World Cup
3D	Three-dimensional
2D	Two-dimensional
V5	Version 5
V	Volt
Ah	Ampere Hour
API	Application Program Interface
SDK	Software development kit
IP	Internet Protocol
DHCP	Dynamic Host Configuration Protocol
IDE	Integrated Development Environment
GUI	Graphical User Interface
OpenCV	Open Source Computer Vision Library
VGA	Video Graphics Array
PNG	Portable Network Graphics
E	Elbow Roll
S	Shoulder Roll
P	Shoulder Pitch

LIST OF FIGURES AND TABLES

LIST OF APPENDICES

1. INTRODUCTION

1.1 Purpose of Thesis

This thesis introduces the drawing system which uses a humanoid Nao robot. The Nao robot learns how to draw shapes (or drawings) by a person based on visual examples appearing on the whiteboard. This thesis investigates the ability of a Nao robot to paint. Therefore, the main goal is to explore the capabilities of a robot to move properly one of the arms in order to recreate drawings.

The example drawings demonstrated to the robot should be non-abstract objects, because the robot is designed to draw something like “star”, “triangle” or “flower” but it is impossible to draw objects like “sunshine”, “wind” or “love”. Instead of using static pictures which could be taken from Internet, it was chosen to use hand-drawn pictures as examples in order to resemble the real situation.

1.2 Outline

This thesis contains eight chapters and it starts from the brief introduction of the purpose of this project and the background information of the Nao robot and programming software tools used in this application. In Chapter 2, the physical setting of the project is explained firstly. Then the overview structure and main phases of the whole application is presented and briefly explained with the flow charts. At the same time, the programming language is introduced. Chapter 3 further explains the details about each stage in the phase of Learning from the examples and the practical example *house* is provided. Moreover, in Chapter 4, the calculations of arm movements are analyzed as well as the functions used for controlling the motions of robot. Chapter 5 shows the process and results of experiments which have been done in order to solve the problems occurred and obtain more accurate outcome. Chapter 6 introduces some possible future work while Chapter 7 makes the conclusion of this thesis. In the final chapter, relative reading materials and some pieces of important code are presented.

1.3 Background

1.3.1 Brief introduction of Nao

The humanoid robot which was used in this project is a Nao robot from Aldebaran Robotics. As shown in Figure 1, the robot is 58cm tall with 2 cameras, 4 microphones, a sonar rangefinder, 2 IR emitters and receivers, 1 inertial board, 9 tactile sensors, 8 pressure sensors, a voice synthesizer, LED lights, and 2 speakers. In total the Nao robot has 25 degrees of freedom (DOF) of which the tension can be individually controlled via an electric motor and an actuator. The user can program and control the robot via Ethernet or wireless networking./2/

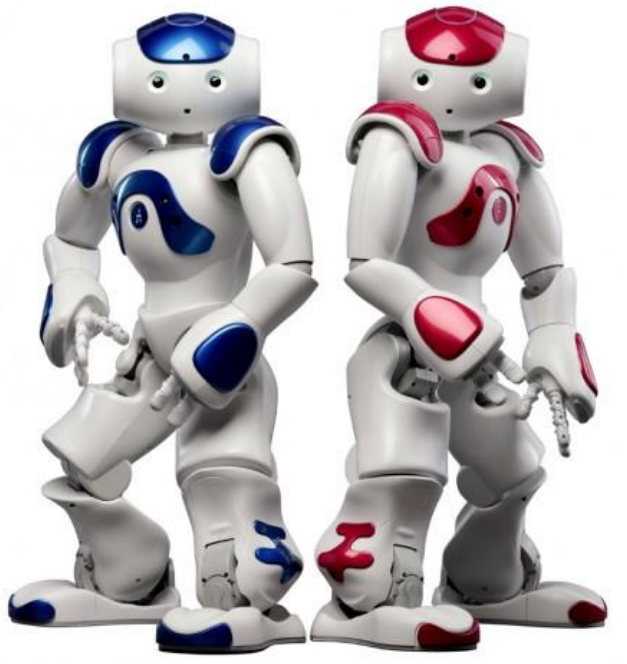


Figure 1. Humanoid Nao robot /2/

1.3.2 Features of Nao Robot v5

In this project, the robot's features that were mainly used are the right arm with a three-fingered hand and camera vision system.

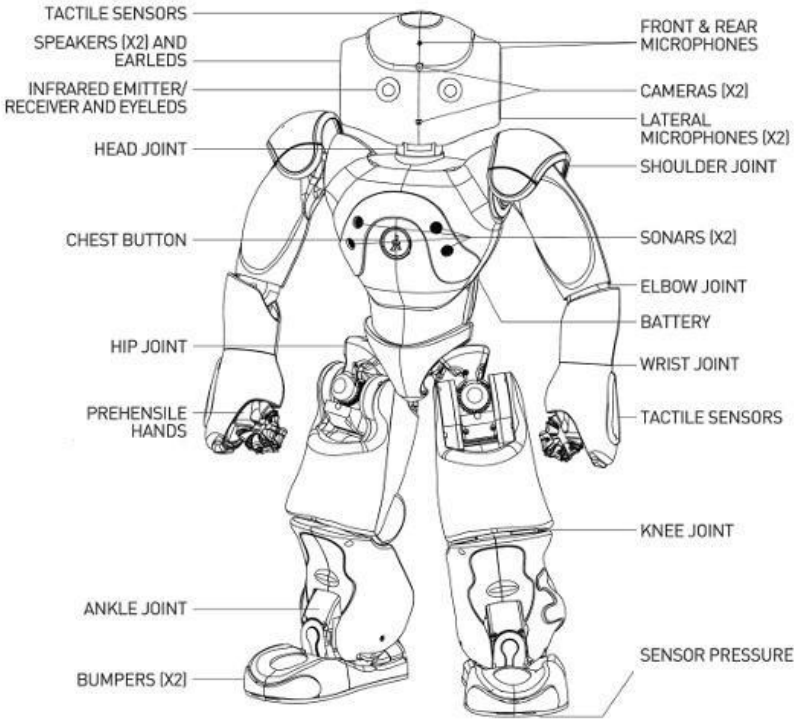


Figure 2. Nao V5 Features /2/

1.3.3 Software and Development Tools

NAO supplies Choregraphe, NAOqi, and Monitor as the main development software.

1.3.3.1 Choregraphe

One of the tools provided by Aldebaran Robotics for creating robot applications is Choregraphe, which is an easy-to-use graphics-based development tool which even makes the beginners easy to handle.

As shown in Figure 3, it provides many useful modules. Users can create robot applications by inserting boxes via a drag-and-drop interface in the graphical programming module. By connecting the output(s) of one box to the input(s) of another one, these boxes can be connected to each other. It is even possible to edit the boxes by pure Python code to create even more advanced applications. With these boxes available users can quickly create applications with different behaviors for the

Nao robot.

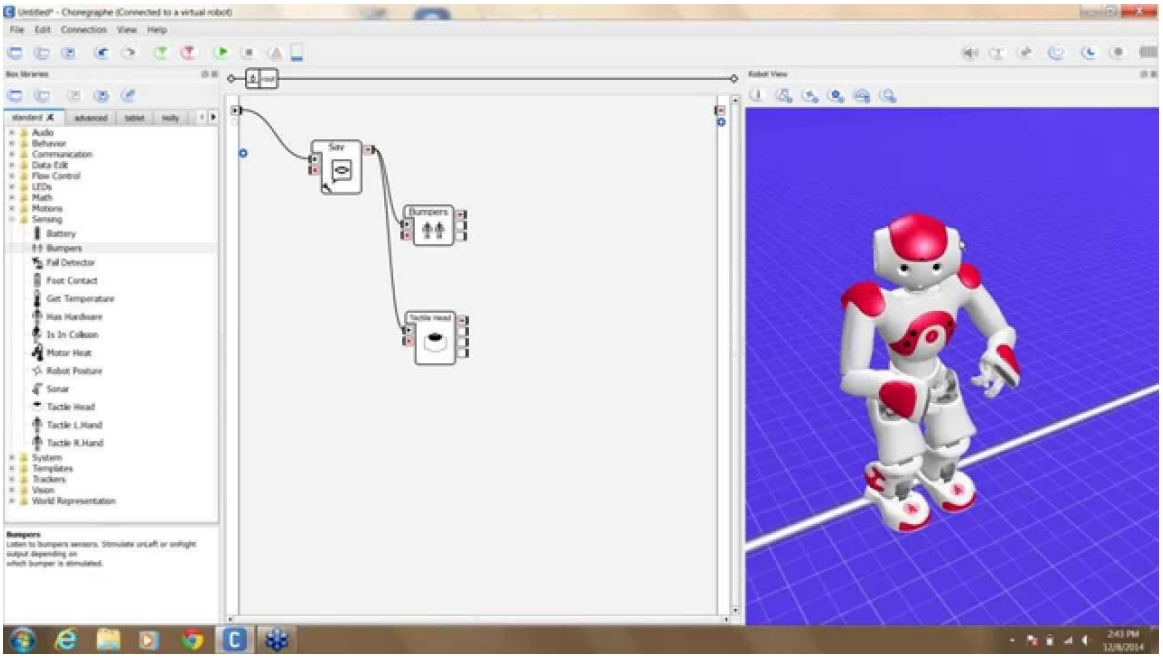


Figure 3. Choregraphe interface/13/

1.3.3.2 NAOqi

NAOqi is the programming framework which is used for all the versions of the NAO robot. It is a cross-platform library that allows programmers to more easily create robotic applications without having to understand how to control those hardware components directly. For instance, there are methods which allow users to set several joint angles in order to control each joint individually or multiple joints at once. Then NAOqi will automatically interpolate the correct movement based on the settings. Using NAOqi helps to quickly develop robot applications for the NAO robot. However, it is impossible to build the full experiment setting with only NAOqi.

1.4 Motivation

Humanoid robotics is always an emerging and challenging research field. They look similar to people so that they can replace and assist people in various fields. Humanoid robots are attracting lots of attention from educators and researchers since they are so valuable and challenging for researching.

It all starts from an interesting video on YouTube: draw what he saw/1/. The robot captured the snapshot of a hand which is in front of it and then the robot tried to recreate his own version of the hand on a whiteboard. At this time, my supervisor suggested

the attractive topic of exploring the drawing capability of the Nao robot. Moreover, this project is not only for entertaining, but also can be developed to be introduced in the elementary schools to help the children to learn to draw. Thus, this application is quite valuable for researching and exploring.

2. APPROACH

In this Chapter, the physical setting and structure of the application will be explained. Additionally, constrains of the approach and the programming language will be presented.

2.1 Physical Setup

It is important to create a safe and exhaustible experimental setting for the robot. But there are some constraints that has to be considered. First of all, during the experiments, the Nao robot is positioned in a sitting position to avoid overloading of the motors placed at the knees. Throughout many trials, the motors of the knees overheat if the robot stays standing for too long. Additionally, a whiteboard where the drawing would be appearing on is placed on the wall and a table with white paper is in front of it. In this way, the Nao robot is perpendicular to the table and facing the whiteboard where the experimenter draws the examples and, when it performs its drawings, the robot can face the table with white paper. The physical setup of the robot is shown in **Figure 4.**

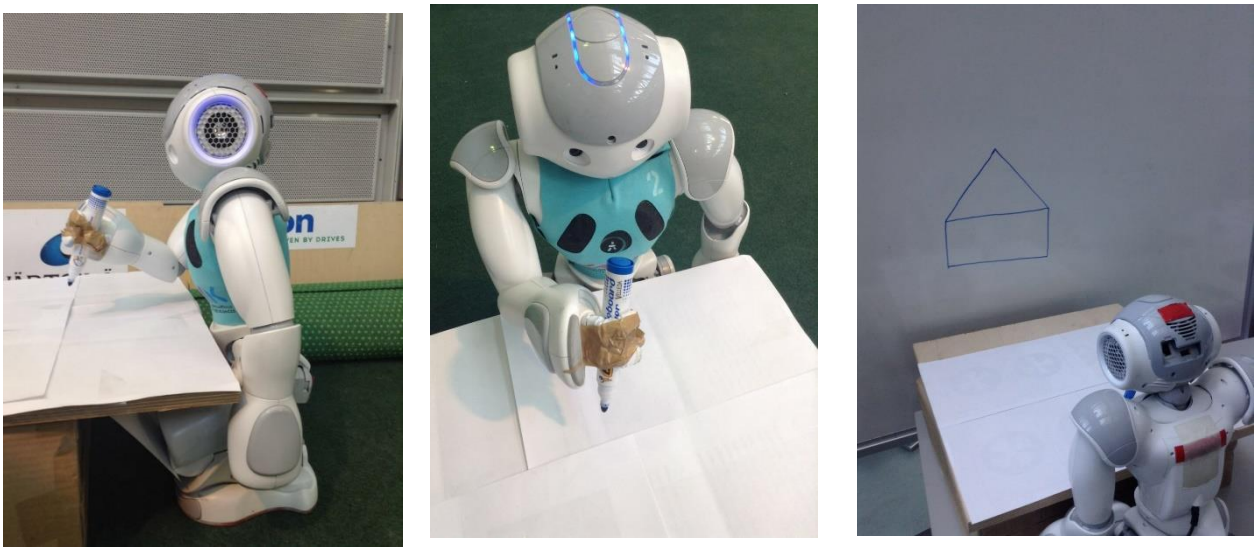


Figure 4. Physical setup of the Nao robot

Another question is that how the Nao robot should handle the marker in the hand. Firstly, the pen point of a small stylus was taped on one finger of the Nao's hand and tried to make it draw on the touchscreens as shown in Figure 5.

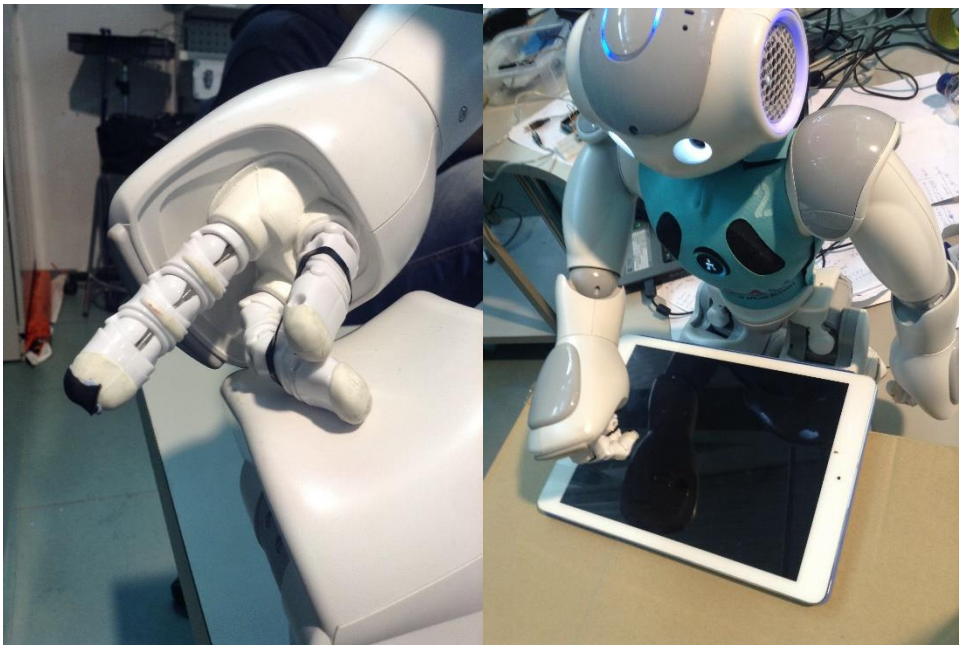


Figure 5, drawing on the touchscreens

Unfortunately, it turned out that it is not the best and safest choice. Then the marker was taken into account. However, it is difficult for the Nao to handle the marker properly since the robot's hand is made of three fingers. In order to make it tight, the marker was taped on NAO's hand as shown in Figure 6.

Figure 6. Maker taped on the Nao's hand

Moreover, the area in which the user can draw the examples so that the robot can see the whole object of a drawing is also a significant constraints which should be taken into account. There are two single cameras located at the forehead and the position of the mouth separately. But in this project, the top camera has only was used as explained in

Chapter 3.1. The visual cone is also limited, thus the experimenter has to be aware of where he or she can draw the examples.

2.2 Structure of the Project

2.2.1 Flow Chart

The following Figure 7 is the flow chart of the whole application. First of all, the experimenter draws a simple object on the whiteboard which is captured by the forehead camera of the Nao robot. The noisy snapshot taken from the robot is preprocessed step-by-step, the features which is going to be used for the robot to recreate a more accurate drawing are extracted by OpenCv together with the Python languages, then the output de-noised and highlighted picture is going to be used for calculating and estimating the arm movements of the robot. At the end, the arm movements are executed and the robot recreate his own version of the example drawing.

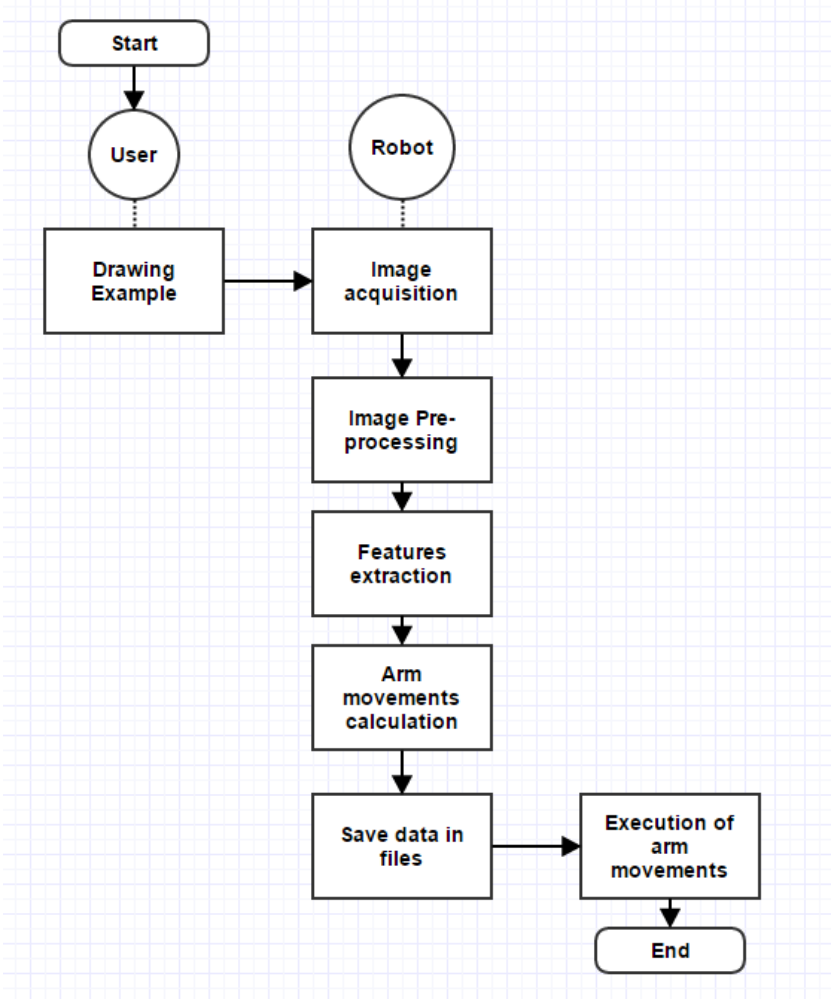


Figure 7. Flow chart of the whole application

2.2.2 Modules

In this section, the important stages of the whole project will be introduced. The whole process can be divided into two main stages: Learning from the examples and Reproduce the drawings as shown in Figure 8. This separation is done in order to emphasize the distinction between those two stages.

Every sub-stages of those two phases will be presented in details and further explained in Chapter 3 and Chapter 4. In this section, the main components of each phase will be introduced.

First of all, as shown in Figure 8, during the first phase, the experimenter makes a simple drawing on the whiteboard and the robot acquires an image of the example drawing. The noisy picture is saved on local computer. The saved image, then, is pre-processed. The sub-stage named pre-processing plays a significant role since the image that is going to be passed to the next sub-stage feature extraction, cannot be noisy. Then the features of the object in the drawing are extracted and highlighted in a graph. The output graph is shown on the computer screen. From the graph, the main features are recognized and used for calculation of the arm movements.

During the second phase: Drawing, the calculations of the joint angles of right arm have been performed at first. There are basically two main steps: transforming points from image domain to paper domain, transforming points from paper domain to robot joint angles. They will be explained in detail in Chapter 4. As soon as the calculations have been done, the lists of all the necessary data are saved in files. The robot receives the settings and executes the movements.

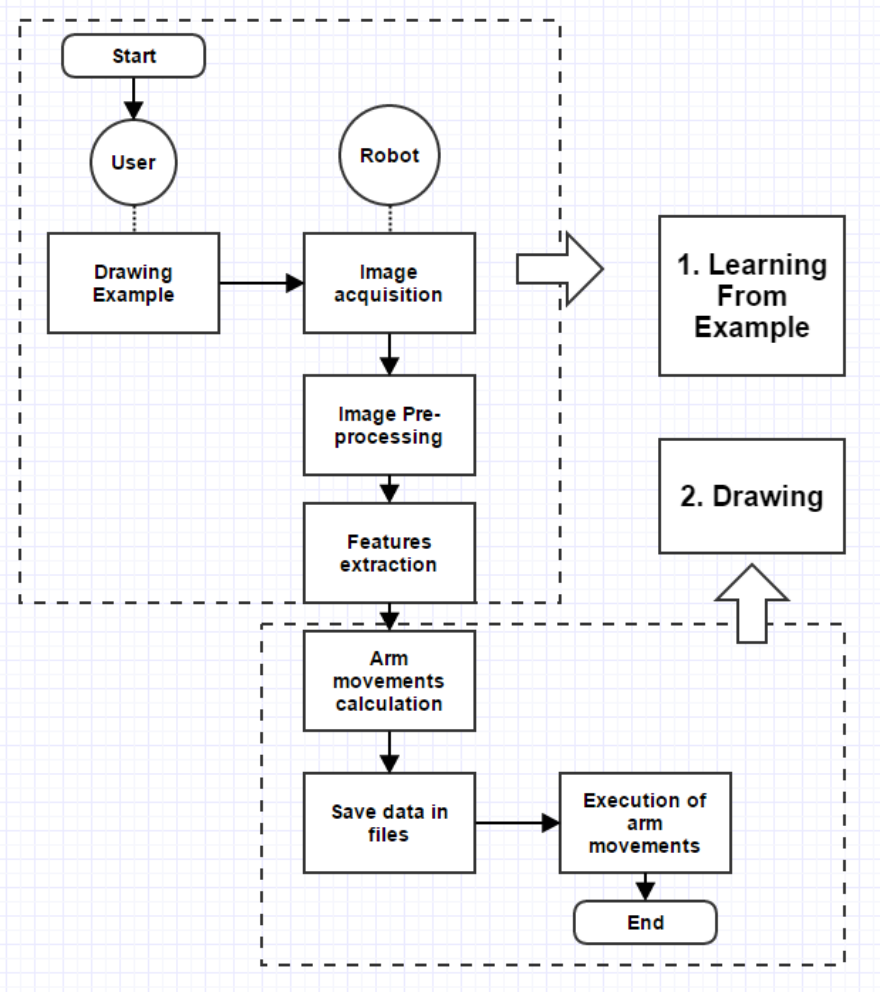


Figure 8. Two main stages of the whole project

2.3 Programming Language

2.3.1 Brief Introduction to OpenCV

OpenCV is so called Open Source Computer Vision Library which contains several hundreds of computer vision algorithms. It is powerful and effective when it is used to process the image. It supports different operation systems such as Windows and Linux with variety of supporting interfaces such as C++, C and Python. In this project, OpenCV and Python were applied together in the phase of analyzing the image.

2.3.2 Brief Introduction to Python

The whole application was all hand-coded in the Python programming language under the Windows platform. Since only Python 2.x is supported by the OpenCV library, the version 2.7.3 of Python was installed and used in this project.

3. LEARNING FROM EXAMPLES

In order to recreate the example drawing successfully, a clear snapshot has to be captured and the features of the image object have to be accurately extracted as well. The detailed explanation of the process for the robot to acquire a picture of the example drawing and find out the features depicted in the image will be provided in this chapter. In the previous chapter, the main components of this stage were introduced, therefore, a practical example *house* will be used in order to show all the sub-steps of this stage throughout this chapter.

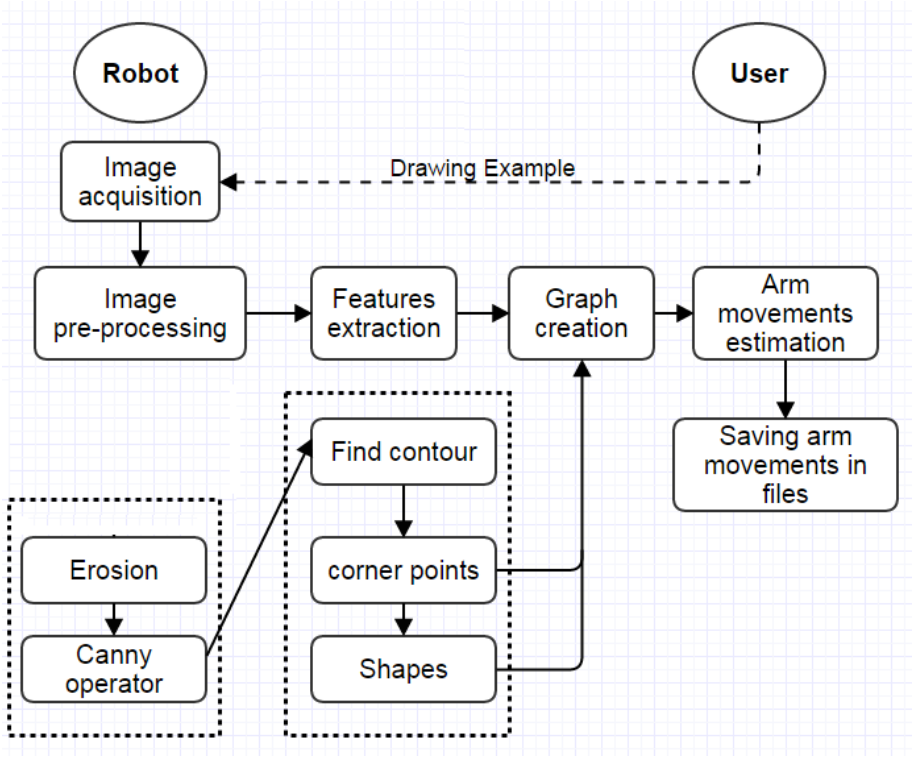


Figure 9. Flow chart of the phase of learning from examples

As shown in Figure 9, the experimenter first makes a simple drawing on the whiteboard which is faced by the NAO robot and the robot takes a snapshot of the drawing. The saved image is preprocessed then. The purpose of the stage of Pre-Processing is to make the object depicted in the saved image clearly recognizable. During this stage, the saved image is morphologically transformed via erosion and the edges of the object is highlighted via Canny edge detector. Then the result image is shown on the computer screen and given as input to the next stage called Features Extraction at the same time. During this step, the important features, which are the contours, the corner points that compose different shapes in the drawing and the approximated shapes, of the image

object are extracted. The result images with those significant features are shown on the screen once the features extraction step is finished.

3.1 Capturing a Snapshot of Examples

First, the experimenter makes a simple drawing on the whiteboard in order for the robot to learn the features from the example. Then, the Nao robot captures a snapshot of the example drawing on the whiteboard which is placed in front of the robot.

Nao robot has two cameras, one single camera placed on its forehead and another one placed at the position of its mouth, which is a unique feature of Nao robot. As we can see in Figure 10, the camera at the forehead has a visual cone pointing forward, while the other camera at the position of mouth points at downwards. Thus, the camera at the forehead has only been used in this project. The camera has a resolution of 1.22 Mega Pixels (MP). So, when robot takes a snapshot of the examples, the resulting picture is 24-bits colored with a size of approximately 640x480 pixels. Figure 10 and Figure 11 shows the locations and features of Nao's robot.

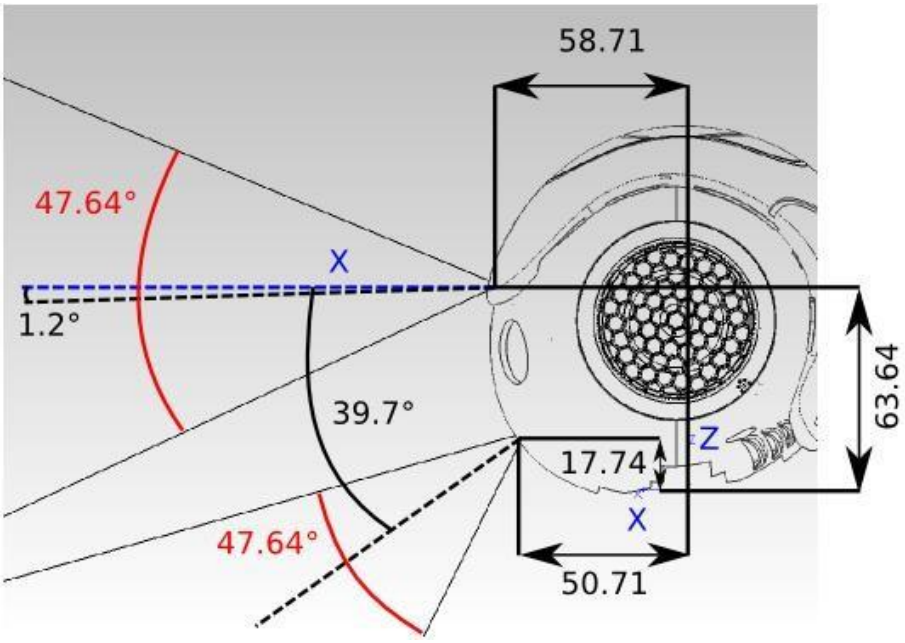


Figure 10. Side View of cameras/2/

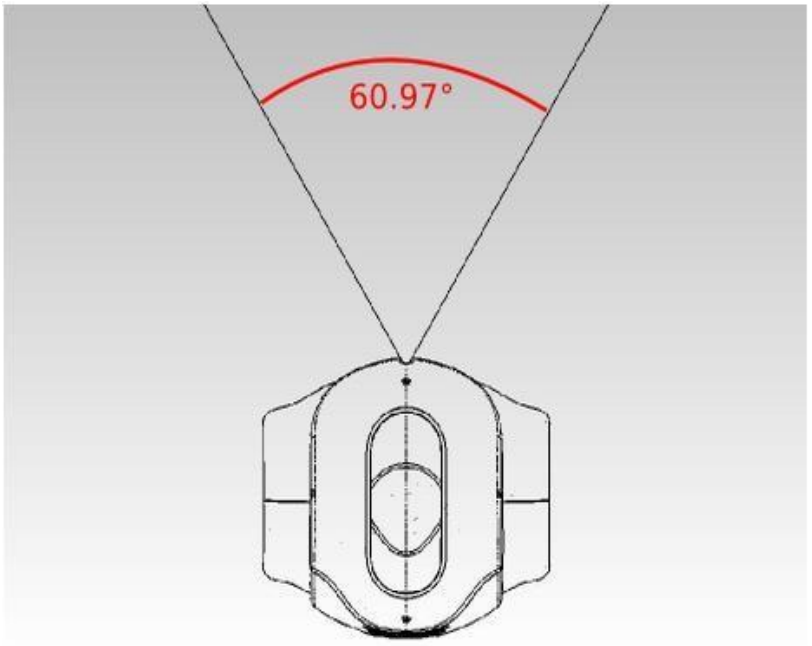


Figure 11. Top View of cameras/2/

The following table is the data sheet of the cameras of the Nao robot provided by the official website.

Table 1. Data sheet of cameras/2/

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

In this thesis, a sketchy house is taken as a practical drawing example in order to present the whole process of the application. Thus, the experimenter makes a hand-drawn house on the whiteboard first and the robot acquires a picture of it then. As concluded in Appendix 2, the method getImage() was used to capture a snapshot. The original picture of a *house* captured by the robot is presented in Figure 12.

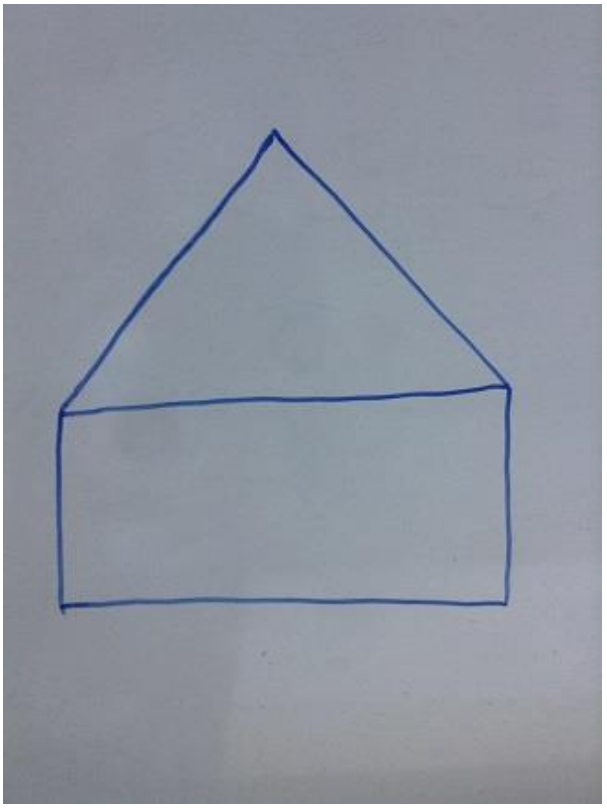


Figure 12. Original drawing example of a house

3.2 Image Pre-processing

It is possible to notice that the snapshot taken from the NAO robot is noisy as shown in Figure 12. Firstly, there might be a variety of noticeable marks on the whiteboard which may have been used for years, thus, subject to usury. Additionally, the light condition in the laboratory may cast shadow on the whiteboard with light reflections. Thus, the noisy input picture taken from Nao's camera needs to be pre-processing in order to obtain more accurate results when we do features extraction.

Therefore, during this pre-processing stage, a noisy and raw picture is taken from the Nao robot in the beginning and, step-by-step, the noises present in the picture are eliminated and reduced and eventually an easily recognizable output image so that the lines and corners of the object are highlighted is supplied.

3.2.1 Erosion

Erosion is the first stage of pre-processing which loads the saved original example drawing from the local folder first, reduces the noise of the picture and enrich some useful features for the next stage. The technology used in this phase is so called

morphological transformation via erosion.

Morphological transformations are simple operations based on the image shape, which is usually performed on binary images. It normally requires two kinds of inputs, one is the original image, and another one is called kernel which is used to decide the nature of operation. There are two basic morphological operators called Erosion and Dilation. Since the morphological transformations are normally utilized on binary images which have background in black and foreground in white, the purpose of erosion is to erode the boundaries of foreground object. Thus, the result of this function is to obtain thinner lines of the foreground object. The dilation function is just opposite of erosion. However, in our application, the snapshot captured by the robot camera has background in bright and foreground in dark. Therefore, in order to increase the thickness of the foreground object, the morphological transformation operator used in this thesis is erosion.

As shown in Figure 13, the erosion function is applied to make bolder lines and have stronger contract color with the background. In this case, it ensures not to lead to a loss of data. In a later phase called feature extraction which will be explained in Chapter 3.3, the evident lines of the objects can be more easily detected.

```
#load the pictures from the folder
original = cv2.imread('house.png')

#apply erosion
kernel = np.ones((2, 2),np.uint8)
originalErosion = cv2.erode(original, kernel, iterations = 1)
```

Figure 13. Loading the picture and apply erosion function

In terms of the setting of kernel in this project, a 2x2 kernel was chosen. Typically, the standard kernel has a square shape and a 3x3 dimension. A bigger size of kernel can return bolder lines of objects. However, the possibility of an overlap of the lines has to be considered as well. At the same time, a smaller 1x1 kernel was also tested which turned out to be useless in making any change in the thickness of lines of the objects in our case. Eventually, a 2x2 kernel was decided to be the optimal choice of our project. As shown in the following Figure 14, (a) shows the original image while (b) is the output image which has been eroded.

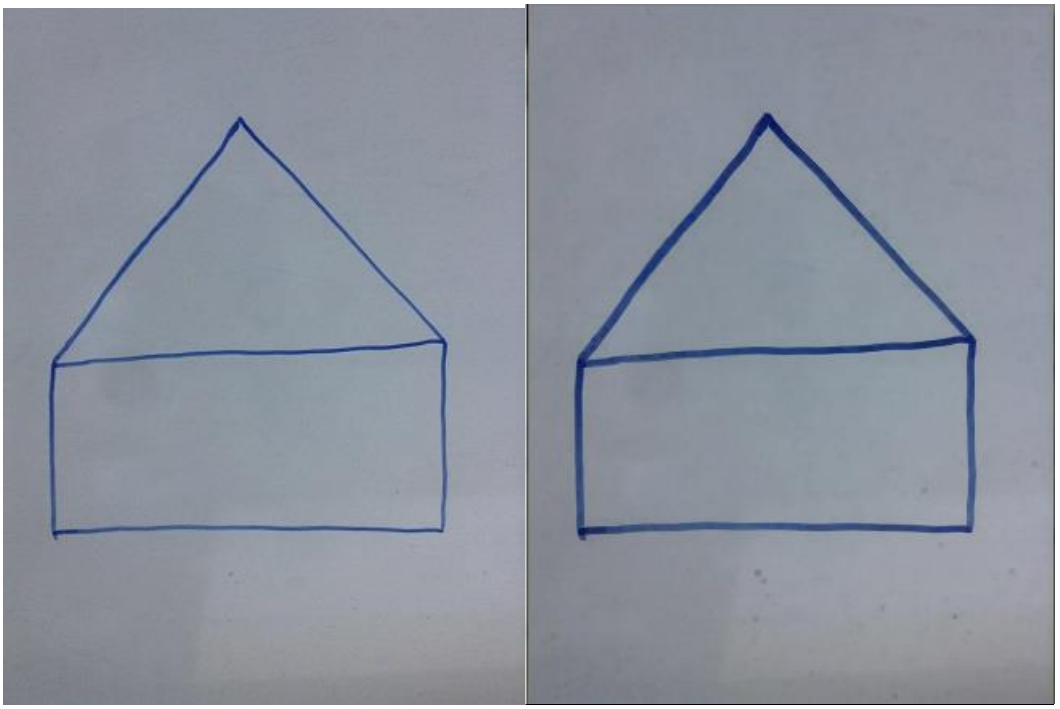


Figure 14. (a) is the original image as the input(house); (b) is the eroded image as the output(house)

3.2.2 Canny Operator

The next significant stage of image processing is so called Canny Edge Detection which is an optimal choice of edge detection algorithm. It was developed by John F. Canny in 1986. The purpose of this stage is to detect the edge of the image in grayscale. It significantly reduces the amount of useless data, while preserving the vital structural properties and features in the image.

To achieve this goal, a multi-stage process is applied in this algorithm:

1. Converting the input image into grayscale.

The pieces of code are shown in Figure 15. It takes the image which has been eroded as the input. It ensures the object to be easily recognized. In Figure 16, the result of the output image is shown, converted to grayscale.

```
originalGray = cv2.cvtColor(originalErosion, cv2.COLOR_BGR2GRAY)
```

Figure 15. Converting the original image into grayscale

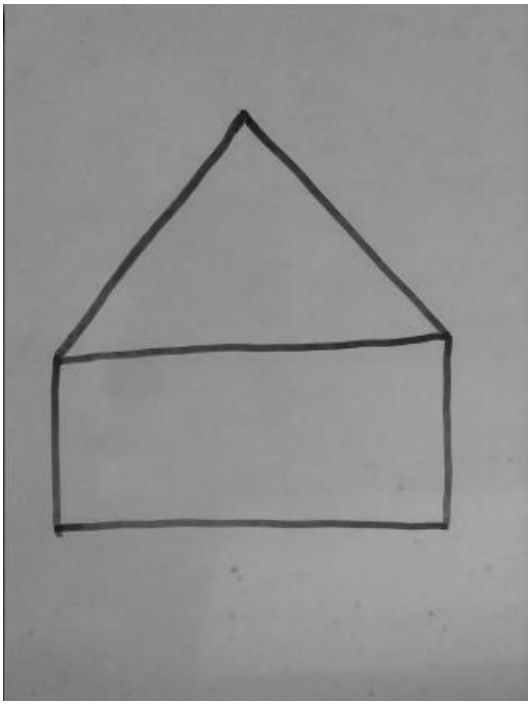


Figure 16. Output image after converting to grayscale(house)

2. Noise Reduction.

   In order to decrease the useless amount of information in the image, the next step is to filter out the noise in the image with GaussianBlur function with a 5x5 Gaussian filter. The pieces of code are shown in Figure 17. In Figure 18, the result after noise reduction can be seen.

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

Figure 17. Reducing the noise in the image

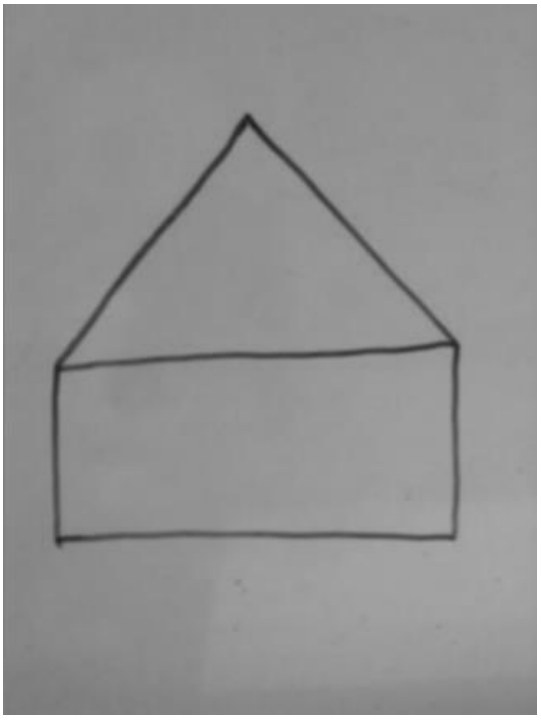


Figure 18. Output image which has been blurred (house)

3. Finding intensity gradients of the image

   For this step, we need to find out the image gradient to highlight regions with high spatial derivatives. There are several operators can be used for this purpose such as Roberts, Sobel and Prewitt. But in this case, Sobel operator was chosen which performs a two-dimensional spatial gradient measurement on an image. Initially, it takes an input grayscale image to calculate the approximate absolute magnitude. Additionally, a pair of 3x3 convention masks were utilized, and Gx, Gy represent x, y direction separately.

   a. The absolute magnitude can be calculated using the Equation (1):

$$|\mathrm{G}| = |Gx| + |Gy| \tag{1}$$

   b. The gradient strength can be calculated using the Equation (2):

$$\mathrm{G} = \sqrt{Gx^2 + Gy^2} \tag{2}$$

   c. The gradient direction can be calculated using the Equation (3):

$$\theta = \tan^{-1}\left(\frac{Gy}{Gx}\right) \tag{3}$$

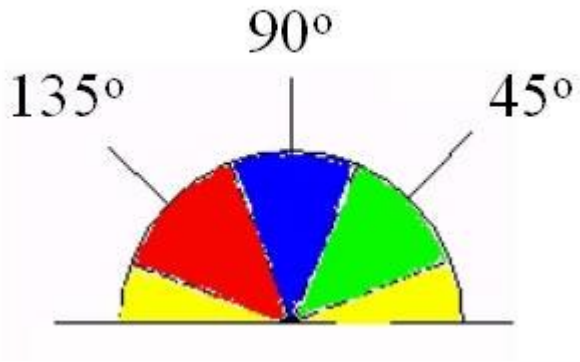


Figure 19. Four possible angles/14/

The gradient direction is always perpendicular to the direction of edges, thus there are only four possible directions when describing the surrounding pixels. As shown in Figure 19, depending on which direction it is closest to, the edge direction can be resolved into the following four directions: 0 degrees (representing the horizontal direction), 45 degrees (along the positive diagonal), 90 degrees (representing the vertical direction), or 135 degrees (along the negative diagonal).

4. Non-maximum suppression

   The next sub-step is called Non-maximum suppression which is applied to track along the regions obtained above. If the pixel value is not at the maximum which means it cannot be treated as the edge, the value equal to zero is set. Consequently, it can get an output image with thin edges.

5. Hysteresis Thresholding

   Furthermore, in order to reduce the gradient array among the remaining pixels which have not been suppressed, hysteresis thresholding was used. Hysteresis has two thresholds values, the minimum and the maximum. If any pixel gradient is higher than the maximum threshold, it is definitely accepted as an edge while any pixel gradient is lower than the minimum threshold, it is definitely colored as the background. In the situation where the pixel gradient is between the two thresholds values, it can be only treated as the edge only if it is connected to a pixel which is greater than the maximum threshold. A recommendation for the radio of the minimum and the maximum is between 1:3 and 1:2.

The OpenCv Canny edge detector combines all the procedures above into one single function called cv2.Canny(). The piece of code is shown in Figure 20.

```
#retrieve edges with Canny
thresh = 175
originalEdges = cv2.Canny(blur, thresh, thresh*2)
```

Figure 20. Function of Canny

A practical example is shown in Figure 21. It is noticeable that the Canny edge detector makes the output image with a completely black background and highlights all the pixels which has been accepted as the edge of the object.

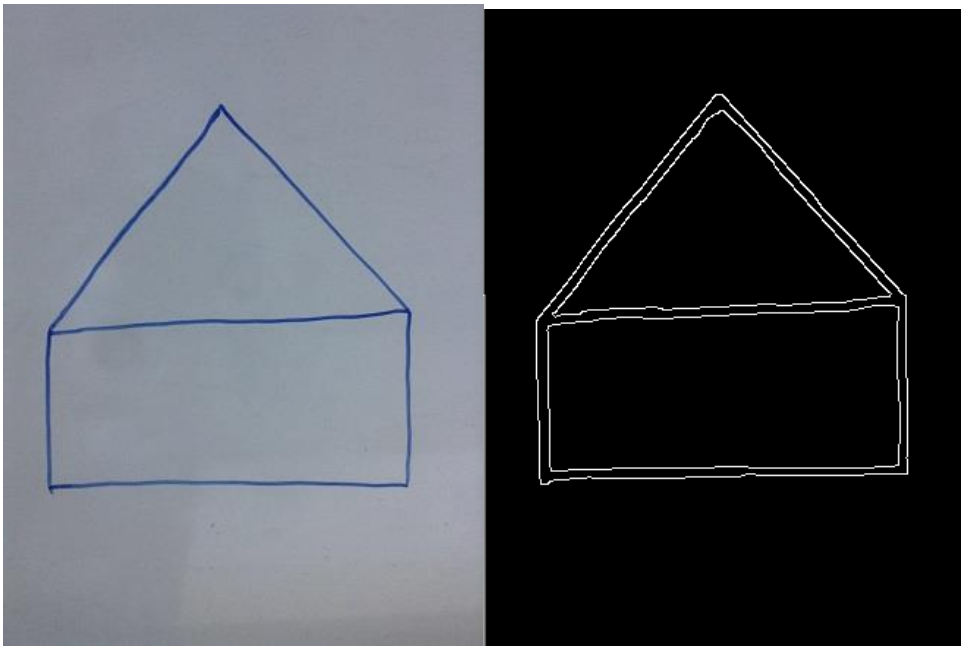


Figure 21. (a) Original image (house); (b) The output image of Canny(house)

3.2 Features extraction

In this section, at first, we have to understand what the features of an image are and why they are so important and useful. In order to recreate a drawing as accurate as possible, the optimal choice is to represent it by means of corner points and approximate shapes.

For the sake of previous steps, an image that has the noise significantly reduced and the features heavily highlighted is obtained as the input of this stage Features Extraction. Thus, the procedure of this stage is to find out the contours points of the object, and try to reduce the amount of the points to find the corner points of the object which is one of the main features of an image. Eventually, according to the corner points, we can find the possible shapes of the object. The result of this important stage is a list of approximate shapes and all the corner points of the object in image domain.

3.3.1 Finding Contours

As explained above, the first step of Features Extraction is to find all the contours of the object in the image. Along the edge of the object, there are plenty of continuous points which have same color or intensity. The contours are just a curve joining all those continuous points. The OpenCv function named cv2.findContours() was used to find and extract all the contours of the object. The piece of code is shown in Figure 22.

```
#extract contours
contours,hierarchy = cv2.findContours(originalEdges, cv2.cv.CV_RETR_CCOMP,
cv2.CHAIN_APPROX_SIMPLE)
```

Figure 22. Function of extracting contours

Thanks to the previous stage Canny Edge Detector, the points of contours can be easily distinguished because they have different color or intensity from the background which is colored as black.

3.3.2 Finding Corner Points

By means of Harris Corner Detection， the corner points of the object in picture can be easily detected. This was done by Chris Harris and Mike Stephens in their paper *A Combined Corner and Edge Detector* in 1988, thus now called Harris Corner Detector. In OpenCv, there is a function call cv2.cornerHarris(). It normally needs four arguments as explained below.

a. Img: it is the input image which should be grayscale and float32 type.
b. BlockSize: it is the size of neighbourhood.
c. Ksize: it is the aperture parameter of Sobel derivative which has been used.
d. K: it is the Harris detector free parameter in the equation.

In Figure 23, the piece of code of finding the corner points is shown.

```
#extract corner points
dst = cv2.cornerHarris(blur,2,3,0.04)
#result is dilated for marking the corners, not important
dst = cv2.dilate(dst,None)
```

```
# Threshold for an optimal value, it may vary depending on the image.
img[dst>0.01*dst.max()]=[0,0,255]
```

Figure 23. Functions of extracting corner points

Figure 24 shows the corner points found in the practical example of house, which is highlighted.

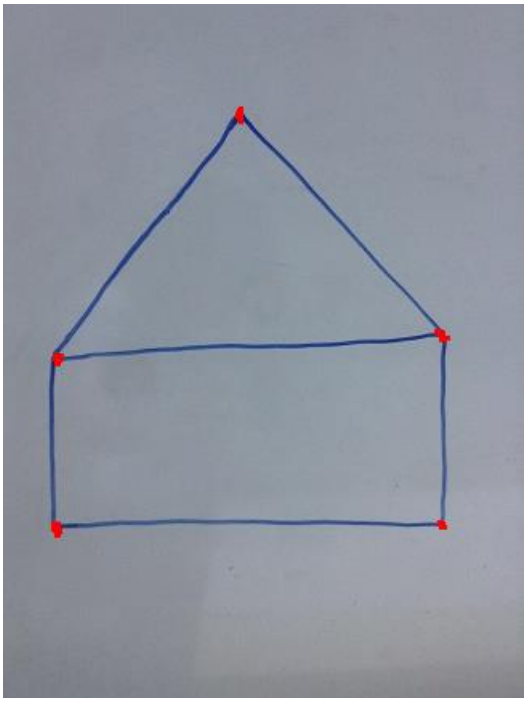


Figure 24. Corner pointes highlighted

3.3.3 Shapes

For this step, the approximate shapes composing the object is going to be detected. In the end, after all the corner points and the shapes composing the drawing object have been extracted, a new graph is created based on the founding. The new graph contains all the important features extracted and all the useful information.

As concluded in Appendix 1, the OpenCv function called cv2.approxPolyDP() is applied to determine the possible shapes composing the object in picture.

The Figure 25 shows the possible shapes found in the picture. It is noticeable that the object in the picture is consist of a triangle which is colored in green and a square which is colored in red. Thus, it can be stated that this house in the example image is composed by a pentagon which is colored in blue. This conclusion matches the result of the previous stage of finding the corner points.

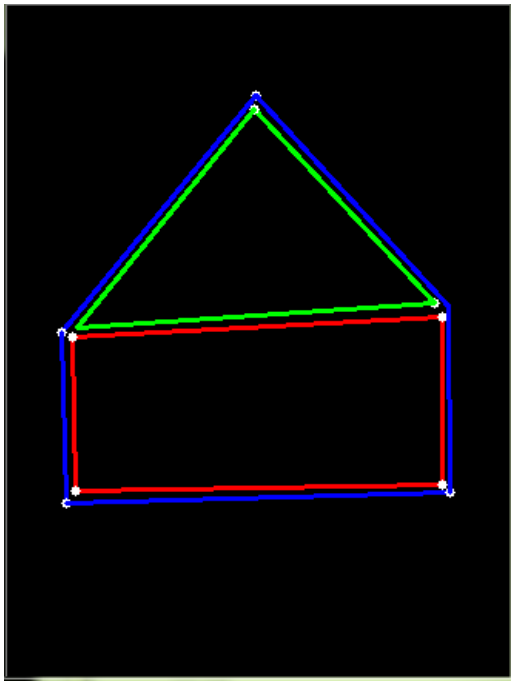


Figure 25. Three possible shapes

3.3 Another practical example

In this section, one more practical example is used to explain the whole process of image processing. Figure 26 shows the original picture captured by the robot. It is a hand painted rectangle which was drew by the experimenter.

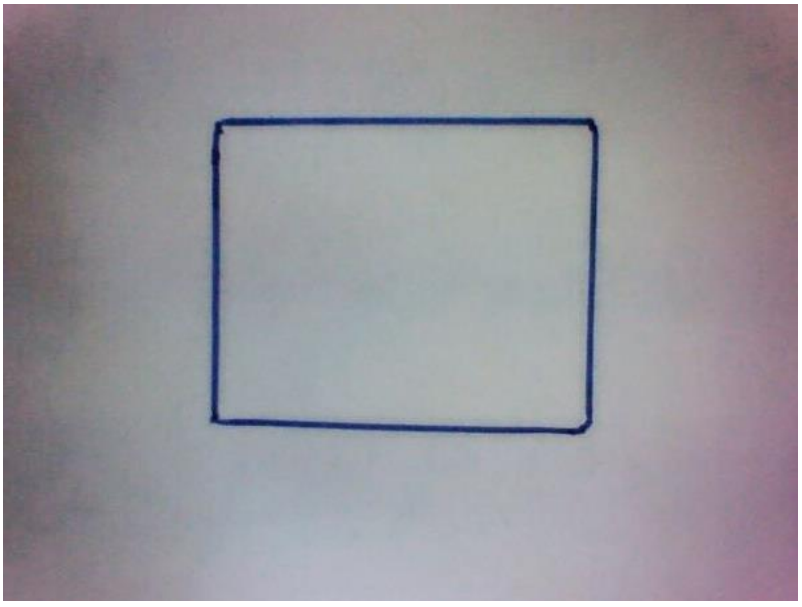


Figure 26, Original image of a rectangle.

By means of one of the Morphological transformations which is called erosion, as shown in Figure 27, the output image have bolder lines and have stronger contract color with the background. It makes the object easy to be detected. The function of OpenCv which was chosen is called cv2.erode().

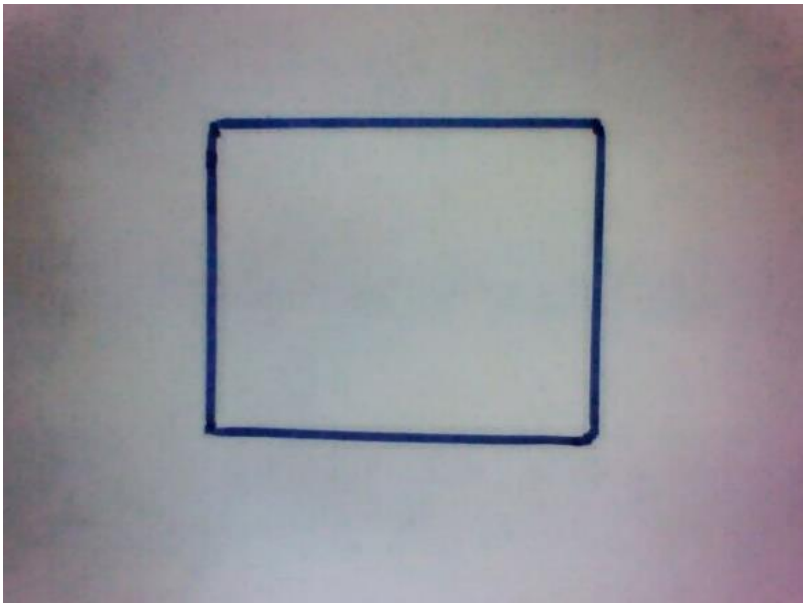


Figure 27, Output image of Erosion(rectangle)

For the first sub-step of Canny Edge detection, the input image has to be converted into grayscale. The function of OpenCv which was chosen is called cv2.cvtColor(). And the output image is shown in Figure 28.

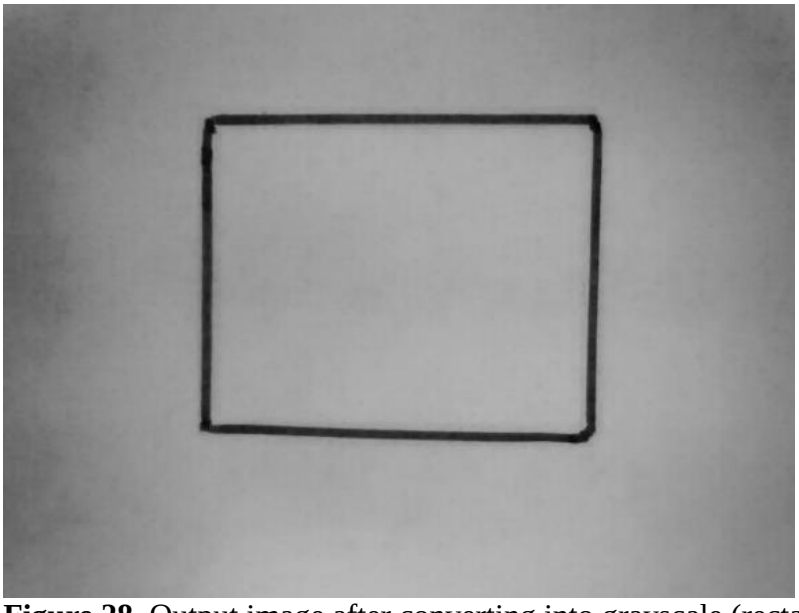


Figure 28, Output image after converting into grayscale (rectangle)

Next, in order to decrease the useless amount of information in the image, the noise in the image has to be filtered out with a 5x5 Gaussian filter. The function of OpenCv was chosen is called cv2.GaussianBlur. The output image after reducing the noise is shown in Figure 29.

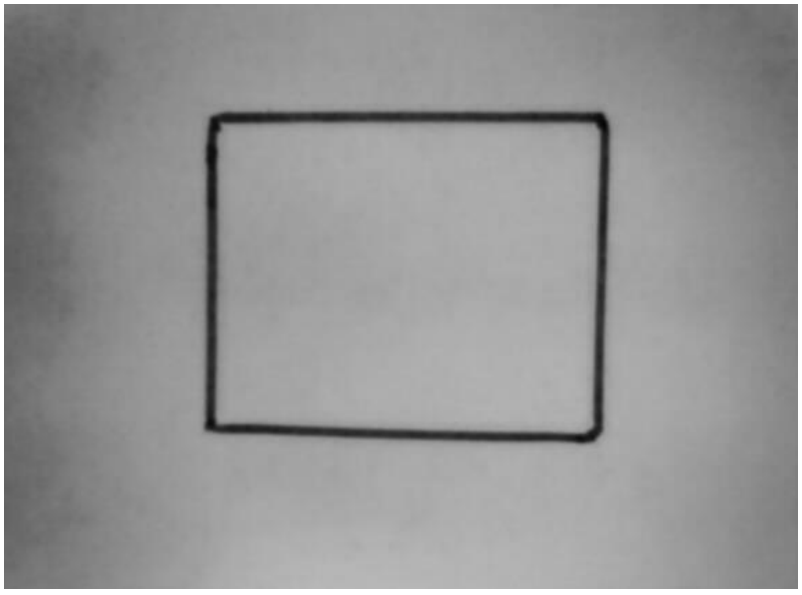


Figure 29, Reducing the noise (rectangle)

Then, by means of Canny edge detector, output image has a completely black background and highlights all the pixels which has been accepted as the edge of the object. The function of OpenCv was used is called cv2.Canny(). The output image of a rectangle is shown in Figure 30.

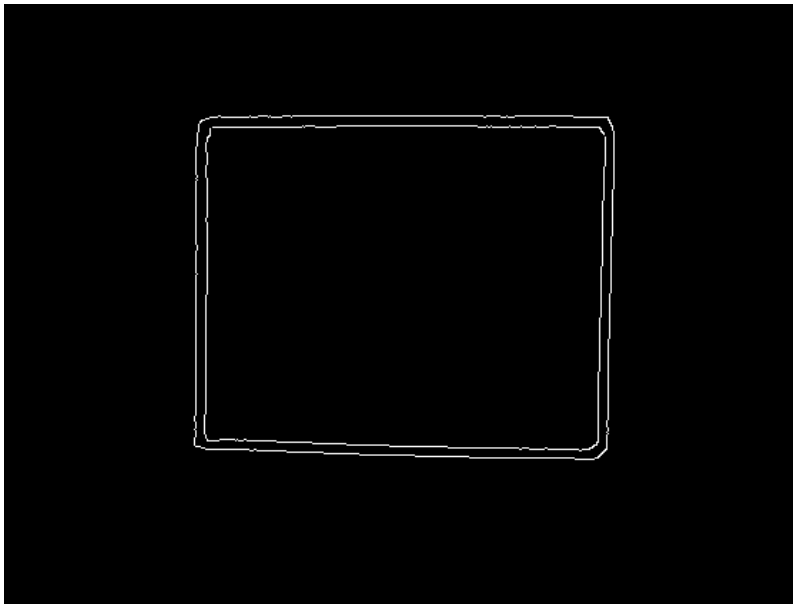


Figure 30, Edge of the object is highlighted (rectangle)

After the previous steps, an image that has the noise significantly reduced and the features heavily highlighted is obtained and it was used as the input of this stage Features Extraction. And the procedure of this stage is to find out the contours points of the object, and try to reduce the amount of the points to find the corner points of the object which is one of the main features of an image. Thus, the output image of a rectangle is shown in Figure 31.

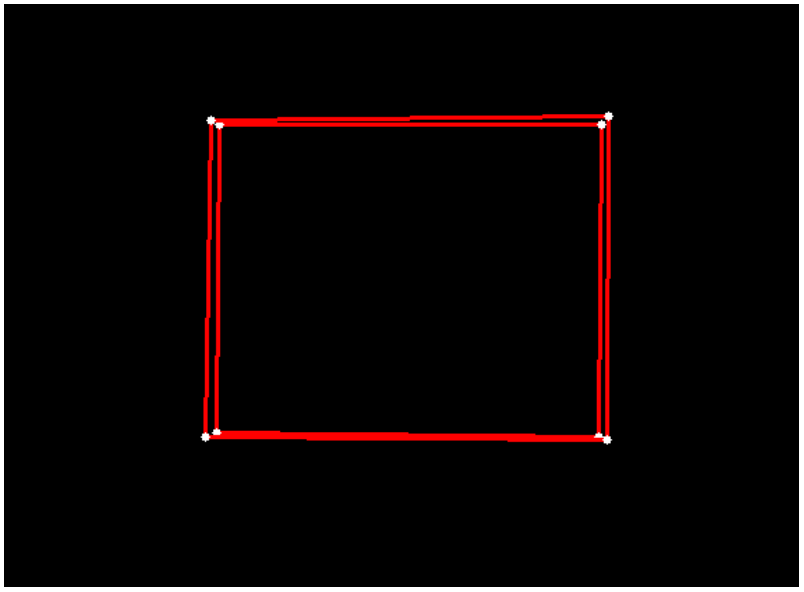


Figure 31, Corner points and approximate shapes (rectangle)

4. PERFORM THE DRAWINGS

In this Chapter 4, the drawing phase will be presented and explained in details. In order to perform the drawing as accurately as possible, the robot needs to know the correct setting data of his joints. Thus, the calculation of the arm movements will be presented as well in this chapter.

4.1 Constraints of Arm

In order to make the arm move in a proper way with the marker during the whole process of drawing, four Degrees of Freedom (DoF) of the right arm of the Nao robot were used. They are Right Shoulder Roll, Right Elbow Roll, Right Shoulder Pitch and the hand.

Shoulder Roll and Elbow Roll were used to make Nao reach the right point on the whiteboard where the Nao robot should draw. Shoulder Pitch was used for reaching the whiteboard with the marker while the robot is drawing. The hand with three fingers is used to hold the marker during the whole process.

As shown in Figure 32, Shoulder Roll and Elbow Roll allows the arm move in right-left direction, while the Shoulder Pitch allows the arm move in up-down direction.

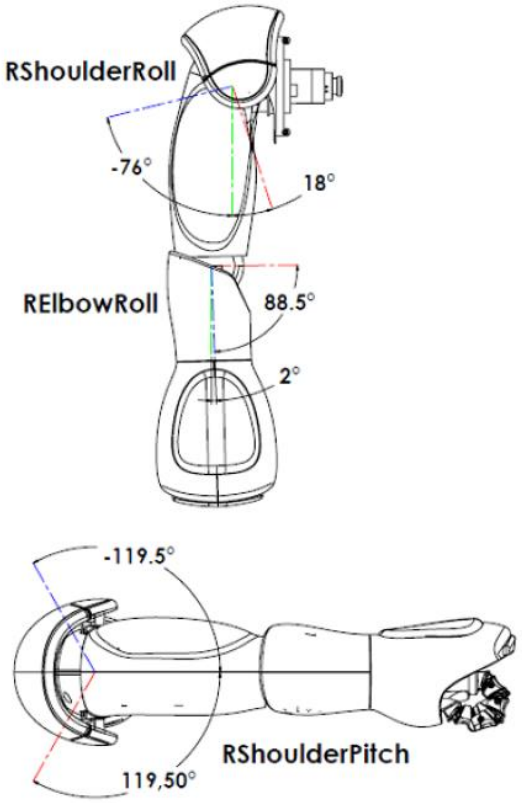


Figure 32. Degrees of freedom of Right Shoulder Roll, Right Elbow Roll and Right Shoulder Pitch/2/

The length of the robot's arms is given in the following table. The data was used in calculating the Right Shoulder Roll and Right Elbow Roll which will be further explained in Chapter 4.2.2.

Table 2. The data sheet of the length of Nao robot's arm /2/

	Length (cm)
UPPER ARM	10.5
FORE ARM	11.37

And due to the limitation of the degrees of freedom and the position of the robot, the drawing area on which the robot can draw is constrained. The detailed experiments of the possible drawing area will be presented in Chapter 5.1.

4.2 Transformations

This section describes how the points of the lines and shapes that the robot is supposed to draw can be transformed from image domain to the joints angles. In order to make the robot perform the same lines and shapes on a piece of paper as accurate as possible, this transformation plays a vital role and it is done in two steps. The first step is to transform each point from the image domain to the paper domain. Finally, each point in the paper domain has to be transformed to the corresponding robot's joints angles.

4.2.1 From Image Domain to Paper Domain

Any snapshot of the examples captured by the robot presents pixels at certain (x, y) positions which are used for relating the position where a point laid in the image. But, as shown in Figure 33**,** the domain of the robot is a combination of the robot's joints and the white paper where the drawing is supposed to be recreated. The X axis of Nao robot is positive toward Nao's front, the Y axis is in right-to-left direction and the Z is in down-to-up direction.

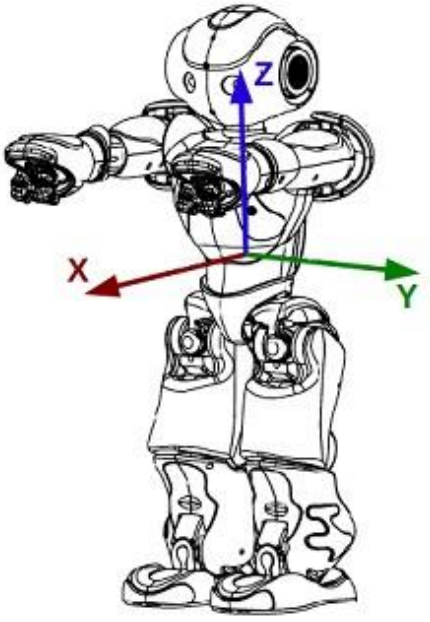


Figure 33. The domain of Nao robot./2/

Therefore, a linear transformation is going be used in order to relate the image domain to the paper domain with the following equations:

$$x_p = x_{pmin} + \frac{Xpmax - Xpmin}{y_{imax} - y_{imin}} (y_{imax} - y_i) \quad (4)$$

$$y_p = y_{pmin} + \frac{y_{pmax} - y_{pmin}}{Ximax - Ximin} (Ximax - Xi) \quad (5)$$

So, this step takes a corner point P = (x_i, y_i) extracted from the example image and translates it into the point P' = (x_p, y_p). Every coordinate (x_i, y_i) which is indicating the point in image domain can be translated into the coordinate (x_p, y_p) which is indicating the same point in paper domain. The values x_{imax}, x_{imin}, y_{imax} and y_{imin} are the maximum and minimum values of xi and yi for all the corner points of lines and shapes in the example image we want the robot to draw. Since they are different from one image to another, these values have to be calculated every time. And the values x_{pmax}, x_{pmin}, y_{pmax} and y_{pmin} are the maximum and minimum values of the area of the drawing that the robot is supposed to recreate. Since the area of the paper has been set, these value can be calculated only once.

4.2.2 From Paper Domain to Robot's Joints Angles

The points (x_p, y_p) in the paper domain has to be translated into the joint movements which are represented as angles of the Shoulder Roll and Elbow Roll. Therefore, the robot can raise its right arm parallel to the paper and then use its Shoulder Roll and Elbow Roll angles to reach the points on the paper. In this step, it takes P' = (x_p, y_p) which is indicating the corner point in paper domain and translates it into a combination of angles S, E which represents the right arm Shoulder Roll and Elbow Roll angles respectively, and Shoulder Pitch. In this project, the Shoulder Pitch is used to let the marker touch the paper during the whole process of drawing. Since the values of (x_p, y_p) are different from one image to another, the function of calculating the S, E and Shoulder Pitch has to be called every time.

As shown in the following equations, the corner point P' = (x_p, y_p) can be drew on the paper by moving the fore arm and upper arm of the robot with the definition of S and E which are taken from /3/.

$$x_p = a\cos(S - \delta) + b\,cos(S - \delta + E + \theta) \quad (6)$$

$$y_p = a\,sin(S - \delta) + b\,sin(S - \delta + E + \theta) \quad (7)$$

Where a and b are the lengths of the fore arm and upper arm of the robot respectively and S and E represents the angles of the Shoulder Roll and Elbow Roll respectively. Since there is an offset of 1.5cm between the shoulder joint and the elbow joint in y direction as shown in Figure 26, δ and θ are the angles resulting from it.

Then to find the reverse transformation, by squaring and adding equations (6) and (7), the following equation is obtained:

$$x_p^2 + y_p^2 = a^2 + b^2 + 2abcos(E + \theta) \tag{8}$$

Then E can be solved:

$$\mathrm{E} = -\theta + \cos^{-1}\left(\frac{x_p^2 + y_p^2 - a^2 - b^2}{2ab}\right) \tag{9}$$

Then S can be solved by firstly expanding the cosine and sine functions in equations (6) and (7) in terms of (S - δ) and (E + θ):

$$x_p = \big(a + b\,cos(E + \theta)\big)\cos(S - \delta) - bsin(E + \theta)\sin(S - \delta) \tag{10}$$

$$y_p = \big(a + b\,cos(E + \theta)\big)\sin(S - \delta) + bsin(E + \theta)\cos(S - \delta) \tag{11}$$

Then equations (10) and (11) can be solved by multiplying equation (10) by (a + b cos(E + θ)) and equation (11) by (b sin(E + θ)) then adding and subtracting:

$$\cos(\mathrm{S} - \delta) = \frac{x_p(a + b\cos(E + \theta)) + y_p bsin(E + \theta)}{a^2 + b^2 + 2abcos(E + \theta)} \tag{12}$$

$$\sin(\mathrm{S} - \delta) = \frac{y_p(a + b\cos(E + \theta)) - x_p bsin(E + \theta)}{a^2 + b^2 + 2abcos(E + \theta)} \tag{13}$$

Finally S can be solved by dividing equation (13) by equation (12):

$$\mathrm{S} = \delta + \tan^{-1}\frac{y_p\big(a + bcos(E + \theta)\big) - x_p bsin(E + \theta)}{x_p\big(a + bcos(E + \theta)\big) + y_p bsin(E + \theta)} \tag{14}$$

Therefore, using equations (6) and (11), any point P' = (x_p, y_p) in the paper domain can be transformed to angles S and E.

Moreover, the unit conversion plays an important role but is easy to be ignored. The corner points in image are saved as (x,y) pixel positions in image domain. However, in Nao robot's space, the distance is described in meters and the angle is described in radians. Thus, the (x,y) pixel positions have to be converted in meters, and it is done by using the following formula:/13/

$$\text{d pixel(px)} = \frac{d}{3779.52755905511}\ meters(m)$$

4.3 Executing Movements

The execution of the right arm's movements is done in the following way: the robot places its arm in a specific position first, then Shoulder Roll elements, Elbow Roll elements and Shoulder Pitch elements are extracted and given as input at the same time. Thus, the robot places its hand in the new position by following those three angles. Until all the elements have been extracted, the procedure is looped and the robot continues reproducing the example drawing.

The robot hold the marker during the whole process of drawing, therefore, when it moves its arm and hand from one position to another, the marker leaves a mark on the paper. This mark on the paper is so called a line. Thus, the coordinates of corner points were enough for reproducing the example drawings.

An example of a line with the relevant brief codes will be used to explain the whole process of executing robot's arm movements.

First, in order to bring the Nao robot's arm to a ready position, a function which is called setAngles() was used to set the angles of Right Shoulder Pitch, Right Shoulder Roll and Right Elbow Roll. The piece of code is shown in Figure 34.

```
#bring arm to start position
angleShoulderPitch = 0.26
motionProxy.setAngles("RShoulderPitch", angleShoulderPitch, fractionMaxSpeed)
angleShoulderRoll = -0.922360807587999
motionProxy.setAngles("RShoulderRoll", angleShoulderRoll, fractionMaxSpeed)
angleElbowRoll = 1.2807857103011893
motionProxy.setAngles("RElbowRoll", angleElbowRoll, fractionMaxSpeed)
```

Figure 34, Code for setting the arm in ready position

In this section, an example of a line is used to explain the whole procedure. The original image is shown in Figure 35.

Figure 35, Original image of a line

Then, by following every steps of image preprocesssing and features extraction which were described in Chapter 3, the output image is shown in Figure 36. And the analysis result indicates that the object in the original picture is a line which contains two possible corner points.

Figure 36, Corner points found in an example of a line

The coordinates of those two corner points in pixel in image domain are shown in Figure 37. And the result is also saved in a text file.

Figure 37, Coordinates of corner points of a line

The corner points in image are saved as (x,y) pixel positions in image domain. However, in Nao robot's space, the distance is described in meters and the angle is described in radians. Thus, the (x,y) pixel positions have to be converted in meters. In order to achieve this purpose and write data in a file, the following piece of code was used as shown in Figure 38. As explained in Chapter 4.1, there are constrains of robot's arm and to avoid reaching Nao's space limits, the (x,y) values were multiplied by 0.3

```
for point in pointsArray:
    value=[]
    newX= (float(point[1])/1850)*0.3
    newY=(-(float(point[0])/1600)*0.3)
    value.insert(0,newX)
    value.insert(1,newY)
    listOfPoints.append(value)
f = open(os.path.join(currentPath,"xyPositions.txt"),"w")
for i in listOfPoints:
    f.write(str(i)+"\n")
f.close()
```

Figure 38, Piece of code used for converting pixels to meters.

In this application, only RShoulderPitch, RShoulderRoll and RElbowRoll were used to move the robot's right arm to wanted positions. S and E can be easily calculated by means of equations (14) and (9) which was explained in Chapter 4.2.2. And the optimal choice of Right Shoulder Pitch will be introduced in Chapter 5 with more practical experiments. Thus, the final step is to load those values, set them to the Nao robot and let the robot's arm move in orders. As shown in Figure 39, the function from NAOqi was used to execute the arm movements is called angleInterpolation(). Each joint of robot can have lists of different lengths but number of angles and number of times must be the same for each joint. There are three effectors called RShoulderRoll, RElbowRoll and RShoulderPitch./8/. The movementList contains all the angles of those three joints.

```
effectorList = [" RShoulderRoll "," RElbowRoll ","RShoulderPitch "]
timeLists=[]
for i in range(len(movementsList)):
    timeLists.append(i+1)
isAbsolute = True
motionProxy.angleInterpolation(effectorList, movementsList, timeLists, isAbsolute)
time.sleep(1)
```

Figure 39, Piece of code used for executing motions

5. EXPERIMENTS AND RESULTS

5.1 Drawing Region for the Nao Robot

As mentioned in chapter 4.1, in this experiment, the drawing area is constrained by the allowed arm movements of the degrees of freedom and the body position of the Nao robot. Thus, the drawing area that the region of the paper that the Nao robot can draw in without walking has to be learned through so many times of experiments. Eventually, the reasonable drawing area was found as shown in Figure 40.

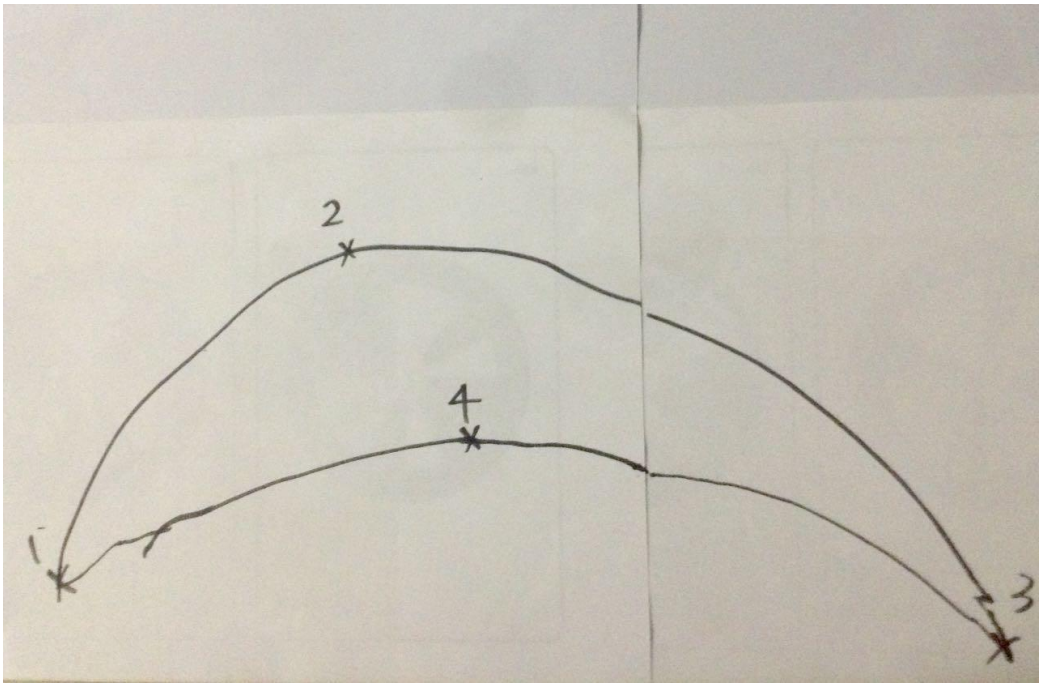


Figure 40. The drawing region on white paper of Nao robot

The following table shows the four points data of the drawing area of robot shown in Figure 40.

Table 3. E and S value of four boundary points

Points	E (Elbow Roll)(rad)	S(Shoulder Roll)(rad)
1	1.536	0.314
2	0	0.314
3	0	-1.326
4	1.536	-1.326

5.2 Shoulder Pitch Range

In the work of El-Barkouky, Mahmoud, Graham, & Farag (2013), the definition of Shoulder Pitch is introduced. As explained in formulations in Chapter 4, it is noticeable that the marker is not always touching the paper when the robot is drawing lines going in the direction of the robot body. Thus, it is necessary to introduce a formulation of the Shoulder Pitch. From the formulations explained in Chapter 4, it was found out that it is the Elbow Roll that is driving the motions of Shoulder Pitch. The more the Elbow gets closer to the robot, the more the marker distances from the whiteboard.

Therefore, it was decided to divide the drawing region into three sub-areas: upper, middle and lower. In order to get the optimal definition of the Shoulder Pitch for each sub-area, practical tests were repeated many times. During these tests, the Nao robot drew different points in every sub-area with different values of Shoulder Pitch. As shown in Figure 41, there are eight points used to determine the Shoulder Pitch.

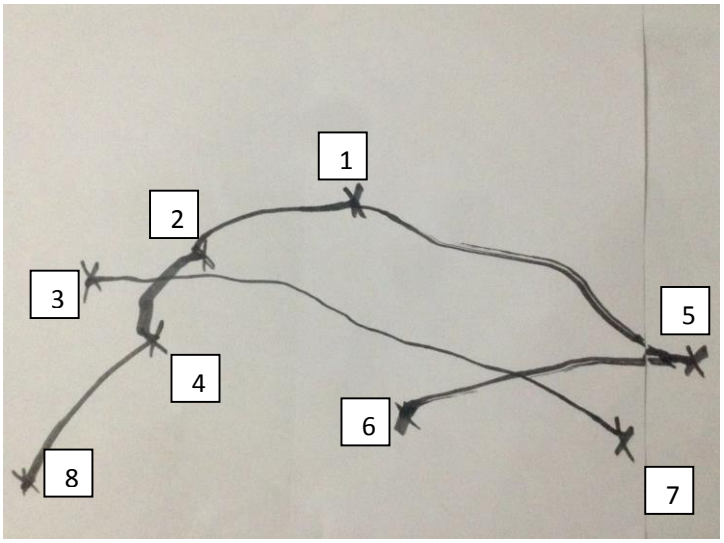


Figure 41, The lines and points drawn by the robot to determine the Shoulder Pitch

The following table shows the results of the practical test which is presented in Figure 41.

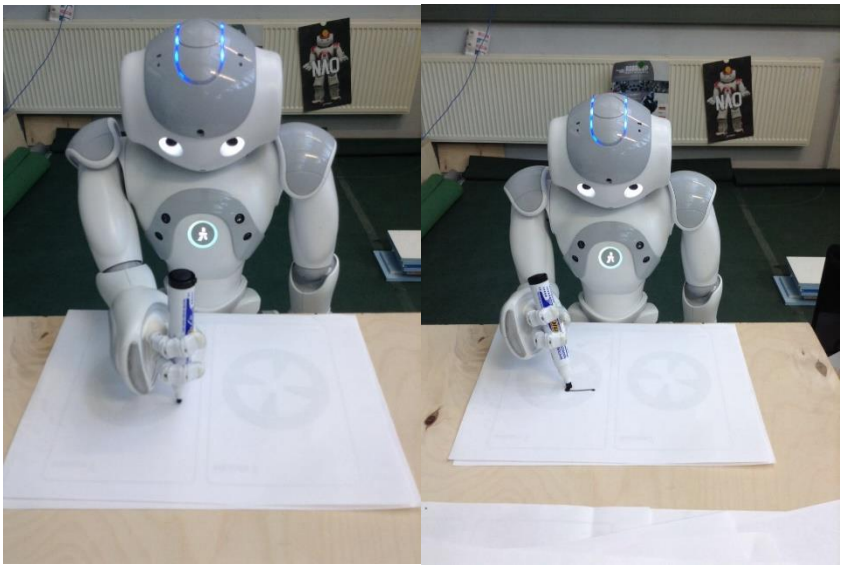
Table 4. Results for eight points for determine Shoulder Pitch

Area	Points	E(Elbow Roll)(rad)	Shoulder Pitch(rad)
Upper	1	0.5426	0.181
Upper	2	0.5333	0.188

Middle	3	1.023	0.263
Middle	4	0.7612	0.274
Middle	5	0.37809	0.282
Lower	6	1.3317	0.35
Lower	7	0.9028	0.368
Lower	8	1.002	0.374

From the practical tests presented above, it is possible to define an optimal value of Shoulder Pitch in three different sub-area. When the right arm is in the upper area, the Shoulder Pitch is between 0.181 and 0.183. When the arm of robot is in the middle area, the Shoulder Pitch is between 0.263 and 0.274. And when the arm is in the lower area, the Shoulder Pitch is between 0.35 and 0.374. So, the Shoulder Pitch can be set to 0.18 when the robot's arm is in the upper area, to 0.26 when the robot's arm is in the middle area, and to 0.365 when the robot's arm is in the lower area. By this formula, the robot's arm can be ensured to keep touching the paper with the marker during the whole process of performing the drawings.

To test the reliably of this formula of Shoulder Pitch, the robot drew a rectangle which presents points that are in different areas of its drawing region. The whole process can be seen in Figure 42. It turns out that the marker touches the paper all the time when the robot is drawing.



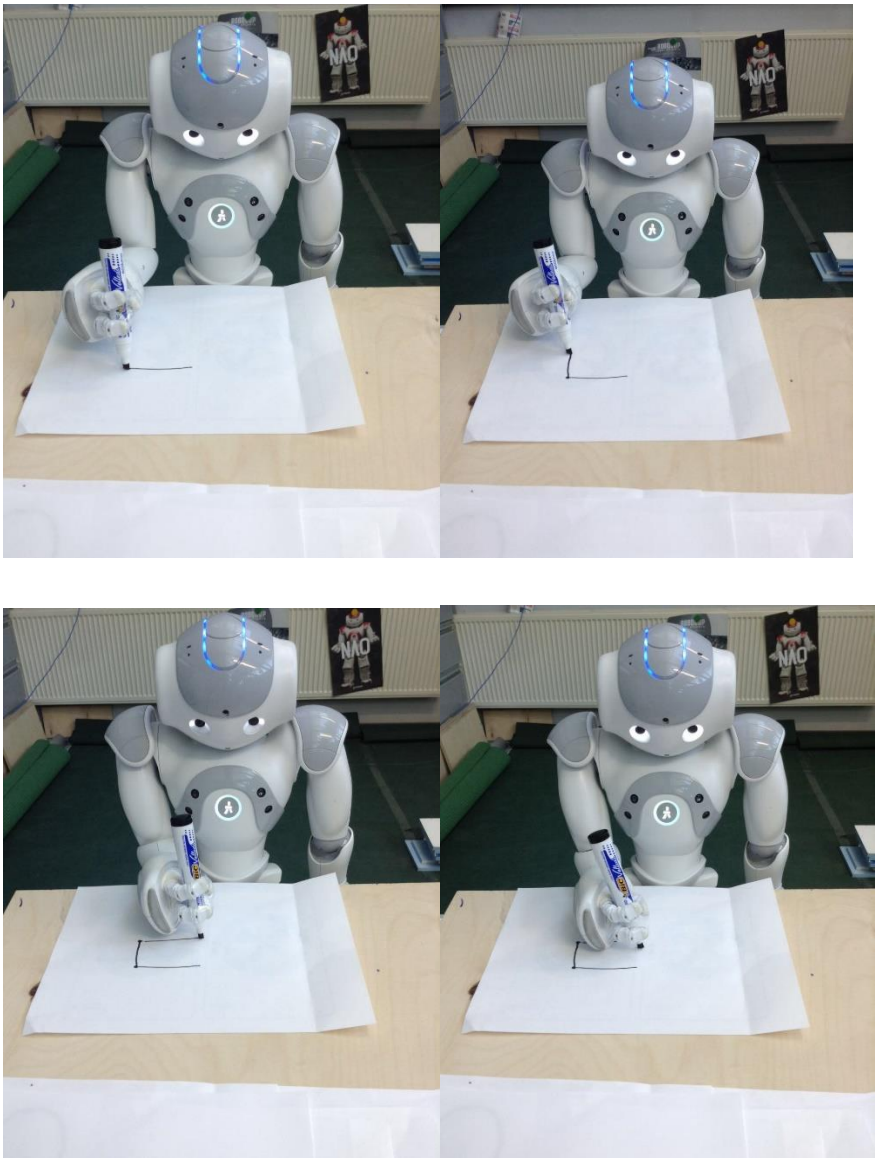


Figure 42. Drawing process of a rectangle.

Figure 43 shows the drawing result of the rectangle.

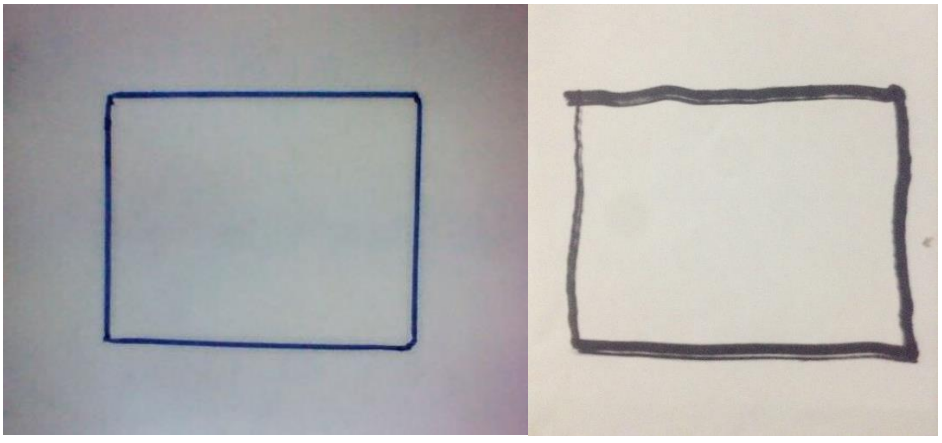


Figure 43. (a) Original image of the rectangle; (b) Drawing result from the robot

5.3 Results Drawn by the Nao Robot

As shown in Figure 44**,** the result of the robot's performance can be different since it is impossible to stable the light conditions in the laboratory and keep a certain distance between the robot and the whiteboard on which the drawing example has been made. According to trial-and-errors, indeed, it is necessary to keep the certain and optimal project settings.

Moreover, the overheating of the robot also has a detrimental influence of the final experiment results.

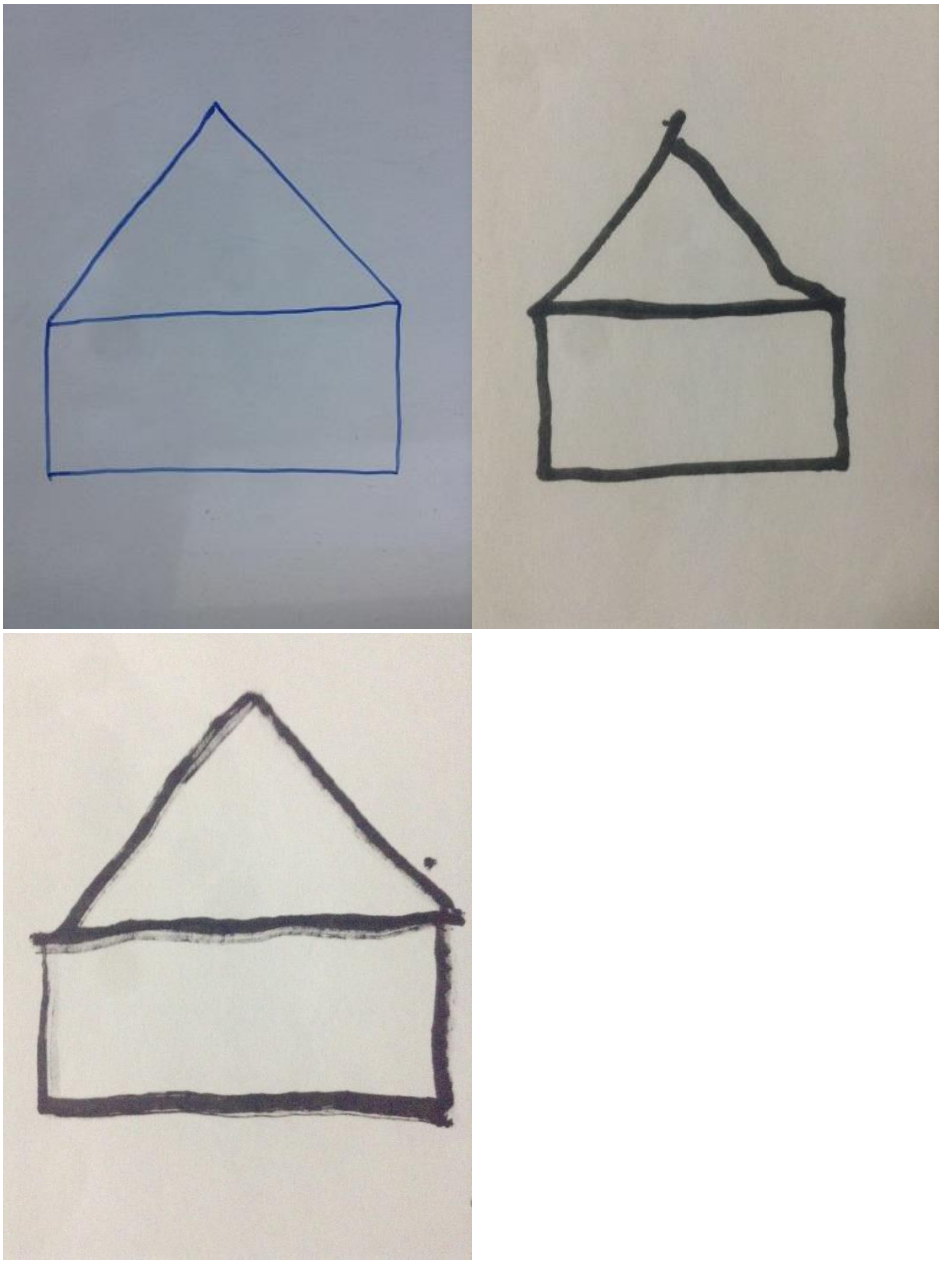


Figure 44. The drawing results of sketchy house with several trials.

As shown in Figure 45, other attempts of recreating the drawing examples have proved that the application is, to some extents, robust and stable in processing and representing the human's drawings. It can accept different drawings and let the robot draw its own version on paper as accurate as possible.

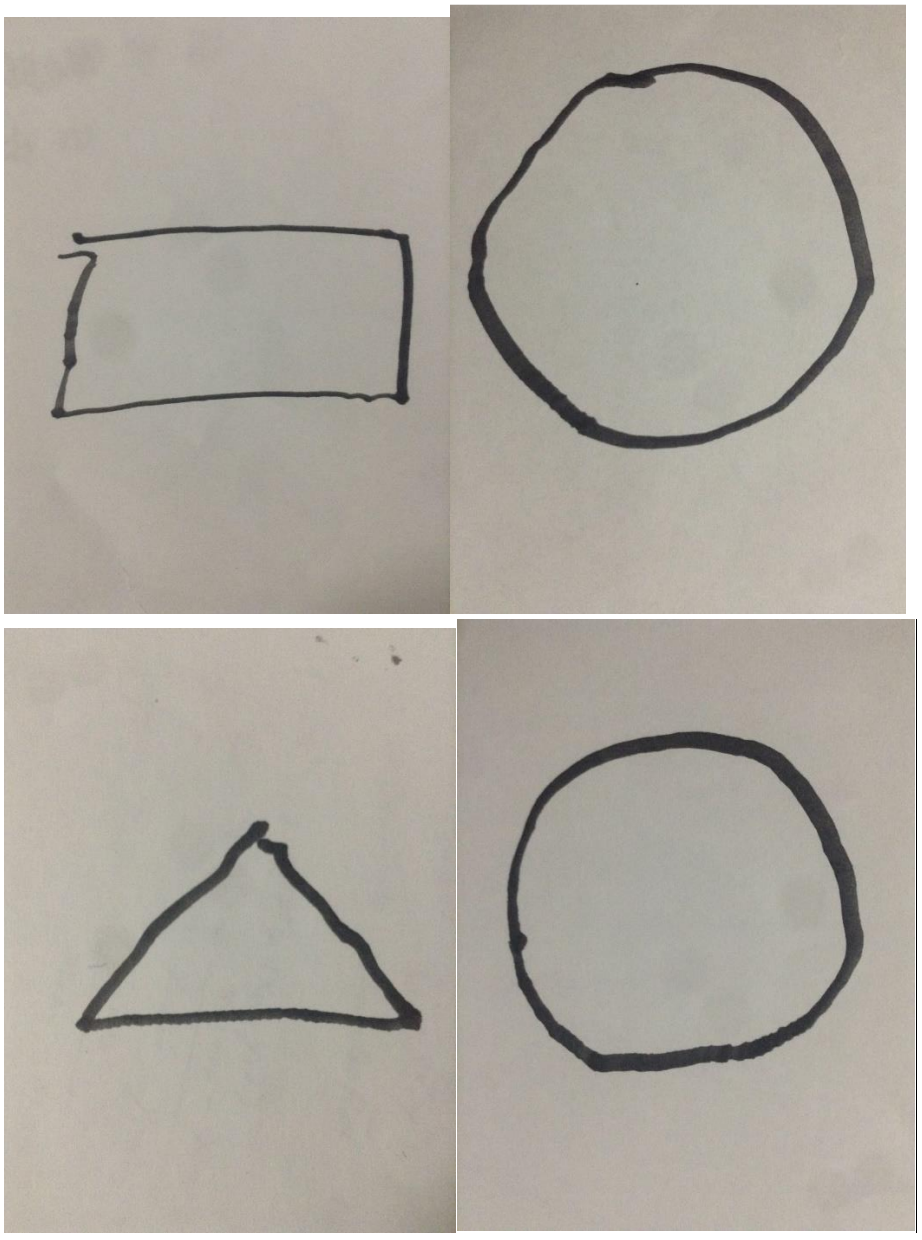


Figure 45. Some other results drawn by Nao robot

6. CONCLUSION

In this thesis, a drawing system using the Nao humanoid robot has been introduced. The user draws on the whiteboard and the robot captures the snapshot of the example drawing and tries to recreate its own version as accurate as possible. The application has proven to be, to some extent, successful and reliable.

There are two main contributions in this thesis: an accurate method for extracting the features of the image using OpenCV library and an analytical transformation from the points in image domain to the joints of robot in robot domain.

The idea of this project started from a video from YouTube, and it sounded impossible but so attractive. When encountering this kind of new topic, it is better to start from reading the related literature, which contributes to getting access to the basic knowledge and finding out the breakthrough of this huge project. And the practical tests are necessary when the project reaches the bottleneck. It is possible to find answers or a new train of thought during the repeated experiments.

Accomplishment of this long-term project really inspired me to continue the research on Nao humanoid robot or even artificial intelligence.

7. FUTURE RESEARCH

A. The limited drawing area of the Nao robot can be larger.

As mentioned in Chapters 4 and 5, there are so many constrains due to the limited degrees of freedom and the body position of the Nao robot. Therefore, the results of drawings are limited in size and the more complex drawings cannot be done since the drawing area of the Nao robot is constrained. In order to widen the range of the drawing region, the Nao robot can be designed to stand up, move in parallel and sit down again to find the new domain for drawing.

B. The more intelligent way for the Nao robot to hold the marker.

As mentioned in Chapter 2, since there are only three fingers in the robot's hand, it is impossible to keep holding the marker in the same position as in the beginning by the Nao robot during the whole drawing. In this application, the marker was taped to the robot's hand, however, in future work, the problem can be solved in more intelligent way. For instance, if the pressure that the Nao robot applies to the marker can be measured in real-time and kept constant, the marker would be tight and the result drawings can be smoother.

C. Nao Robot writes any word user asks, and spells the word as he writes it down.

The arm movements of a specific example drawing can be saved in files, thus, why not to create a database for the drawing system. In future work, the developments and improvements of the Speech Recognition are required. In this case, a label for the example drawing has to be utilized and the files of different objects should be saved based on the label. A prepared database of the words is needed. Moreover, the situation of recognizing the unknown words also has to be taken into account.

This improvement can bring the interaction between human and robots to a higher level which is so similar to human-human interaction. Therefore, this drawing system can be implemented as an educational robotic drawing system: an interaction can be created in which a user teaches the robot how to perform drawings with labels.

8. REFERENCES

/1/ Calzada, F. (2012). NAO robot draws what he sees. Drawing my hand. http://www.youtube.com/watch?v=RjSw6xpucnM. Accessed 01/03/2015

/2/ Aldebaran official website. https://www.aldebaran.com/en.Accessed 30/08/2015

/3/ El-Barkouky, A., Mahmoud, A., Graham, J., & Farag, A. (2013). An interactive educational drawing system using a humanoid robot and light polarization. IEEE International Conference on Image Processing,

/4/ OpenCV 2.3.11.0 documentation
http://docs.opencv.org/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

/5/ Image Processing in OpenCV: Morphological Transformations
http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html. Accessed 15/09/2015

/6/ NAOqi Framework.
http://doc.aldebaran.com/1-14/dev/naoqi/index.html?highlight=naoqi%20frame-%20work.Accessed 25/03/2015

/7/ PyDev Python IDE for Eclipse.
http://marketplace.eclipse.org/content/pydev-python-ide-eclipse. Accessed 25/03/2015

/8/ Aldebaran Documentation: Joints.
http://doc.aldebaran.com/2-1/family/robots/joints_robot.html. Accessed 15/09/2015

/9/ Harris Corner Detector.
http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html#harris-corners Accessed 25/09/2015

/10/ Features Detection.
http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_table_of_contents_feature2d/py_table_of_contents_feature2d.html. Accessed 25/09/2015

/11/ Francesco Sgaramella (2014) Master Thesis in Artificial Intelligence: A new framework for robots that can draw based on labeled examples

/12/ Ahmed El-Barkouky, Ali Mahmoud, James Graham and Aly Farag(2012): An interactive educational drawing system using a humanoid robot and light polarization.

/13/ AldebaranRobotics(2014), Aldebaran Choregraphe Weninar http://youtu.be/2byYfgypse8. Accessed 30/08/2015

/14/ Caiye917015406 (2012), Canny Edge Detection Tutorial http://blog.csdn.net/liurong_cn/article/details/7866472. Accessed 01/09/2015

APPENDIX 1.

DETERMINE THE SHAPES AND DRAW THE NEW IMAGE BASED ON SHAPES

```
#determine the shapes and draw the new image based on shapes
for b,cnt in enumerate(contours):
    if hierarchy[0,b,3] == -1: #Avoid inner originalContours
        approx = cv2.approxPolyDP(cnt,0.025*cv2.arcLength(cnt,True), True)
        if len(approx)==5:
            if cv2.isContourConvex(approx):
                print "pentagon" + " with area = " +str(cv2.contourArea(cnt))
                clr = (255, 0, 0)
                #draw the pentagon in the new image
                create_graph(approx, clr)
            else:
                print str(len(approx))+" lines"
                clr = (204, 255, 0)
                #draw 5 lines
                create_graph(approx, clr)
        elif len(approx)==6:
            if cv2.isContourConvex(approx):
                print "hexagon" + " with area = " +str(cv2.contourArea(cnt))
                clr = (0, 102, 255)
                #draw the hexagon in the new image
                create_graph(approx, clr)
            else:
                print str(len(approx))+" lines"
                clr = (204, 255, 0)
                #draw 6 lines
                create_graph(approx, clr)
        elif len(approx)==2:
            print "line"
            #draw the line in the new image
            cv2.line(newimg, (approx[0][0][0], approx[0][0][1]), (approx[1][0][0],
approx[1][0][1]), (204, 255, 0), 2)
            cv2.circle(newimg, (approx[0][0][0], approx[0][0][1]), 3, (255,255,255),
-1)
```

```
                cv2.circle(newimg, (approx[1][0][0], approx[1][0][1]), 3, (255,255,255),
-1)
        elif len(approx)==1:
                print "point"
                #draw the point in the new image
                cv2.circle(newimg, (approx[0][0][0], approx[0][0][1]), 3, (255,255,255),
-1)
        elif len(approx)==3:
                if cv2.isContourConvex(approx):
                        print "triangle" + " with area = " +str(cv2.contourArea(cnt))
                        clr = (0, 255, 0)
                        #draw the triangle in the new image
                        create_graph(approx, clr)
                else:
                        print str(len(approx))+" lines"
                        clr = (204, 255, 0)
                        #draw 3 lines
                        create_graph(approx, clr)
        elif len(approx)==4:
                #check if it is a square
                if cv2.contourArea(approx) > 1000 and cv2.isContourConvex(approx):
                        approx = approx.reshape(-1, 2)
                        max_cos = np.max([angle_cos(approx[i%len(approx)], approx[i-
2], approx[i-1] ) for i in xrange(2, len(approx)+1)])
                        min_cos = np.min([angle_cos(approx[i%len(approx)], approx[i-2],
approx[i-1] ) for i in xrange(2, len(approx)+1)])
                        if min_cos >= -0.1 and max_cos <= 0.3:
                                print "square" + " with area = " +str(cv2.contourArea(cnt))
                                #draw the square in the new image
                                cv2.line(newimg,(approx[0][0], approx[0][1]), (approx[1][0],
approx[1][1]), (0,0,255), 2)
                                cv2.line(newimg,(approx[1][0], approx[1][1]), (approx[2][0],
approx[2][1]), (0,0,255), 2)
                                cv2.line(newimg,(approx[2][0], approx[2][1]), (approx[3][0],
approx[3][1]), (0,0,255), 2)
                                cv2.line(newimg,(approx[3][0], approx[3][1]), (approx[0][0],
approx[0][1]), (0,0,255), 2)
                                cv2.circle(newimg, (approx[0][0], approx[0][1]), 3,
```

```
(255,255,255), -1)
                    cv2.circle(newimg,    (approx[1][0],    approx[1][1]),    3,
(255,255,255), -1)
                    cv2.circle(newimg,    (approx[2][0],    approx[2][1]),    3,
(255,255,255), -1)
                    cv2.circle(newimg,    (approx[3][0],    approx[3][1]),    3,
(255,255,255), -1)
            else:
                print str(len(approx))+" lines"
                clr = (204, 255, 0)
                create_graph(approx, clr)
        elif len(approx) >=7:
            if cv2.isContourConvex(approx):
                #check if it is a circle
                area = cv2.contourArea(cnt)
                x, y, w, h = cv2.boundingRect(cnt)
                radius = w/2
                if abs(1 - (area / (math.pi * pow(radius, 2))))<=0.1:
                    print "circle" + " with area = " +str(cv2.contourArea(cnt))
                    M = cv2.moments(cnt)
                    cx = int(M['m10']/M['m00'])
                    cy = int(M['m01']/M['m00'])
                    #draw the circle
                    cv2.circle(newimg, (cx, cy), radius, (0,255,255), 2)
                    #draw the center of the circle
                    cv2.circle(newimg, (cx, cy), 3, (69,13,254), -1)
                    #otherwise check if it is an octagon
                elif len(approx)==8:
                    print "octagon" + " with area = " +str(cv2.contourArea(cnt))
                    clr = (91, 0, 153)
                    #draw the octagon in the new image
                    create_graph(approx, clr)
            else:
                print str(len(approx))+" lines"
                clr = (204, 255, 0)
                create_graph(approx, clr)
```

APPENDIX 2.

GET IMAGE FROM NAO CAMERA

```
#First get an image from Nao, then show it on the screen with PIL.
def getImage(IP,PORT):
    camProxy = ALProxy(“ALVideoDevice”,IP, PORT)
    resolution = 2        # VGA
    colorSpace = 11      # RGB
    fps = 5
    videoClient = camProxy.subscribe("python_client", resolution, colorSpace, fps)
    # Get a camera image.
    # image[6] contains the image data passed as an array of ASCII chars.
    naoImage = camProxy.getImageRemote(videoClient)
    # Now we work with the image returned and save it as a PNG    using ImageDraw
    package.
    # Get the image size and pixel array. #[6]: binary array of size height * width *
    nblayers containing image data.
    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.
    imgName = "house.png"
    im.save(imgName, "PNG")
    im.show()
    camProxy.unsubscribe(videoClient)
```