Binglong Feng (e1001928)

HUMANOID ROBOT NAO PLAYING “SPOT THE DIFFERENCES” GAME

Information Technology
2016
FOREWORD

Till now, I have been staying in Vaasa for 4 years and I believe the 4-year studying life in Vaasan Ammattikorkeakoulu (Vaasa University of Applied Sciences) is the most unforgettable experience throughout my whole life. First of all, I want to give my greatest gratitude to the school for providing us such a peaceful environment to study in.

In addition, I would like to thank my supervisor Dr. Menani Smail for giving me continuous help during the whole process of accomplishing my thesis. Without his time and efforts, I would not have been able to finish my thesis so successfully and smoothly.

Furthermore, I would like to express my appreciation to all the stuff in VAMK including Dr. Yang Liu, Dr. Chao Gao, Dr. Ghodrat Moghadampour, Mr. Jukka Matila, Mr. Santiago Chavez Vega, Mr. Jani Ahvonen and the other teachers who have taught me and contributed a lot to my thesis.

Last but not the least, I would like to thank all the previous and current fellows in the Robotics Lab including Lu Wei, Lin Han, Li Xiaotian, Li Xiuyang for creating perfect working environment for me. Here I also want to give many thanks to my girlfriend Min Aoyu for her constant encouragement and support. I sincerely hope all the fellows will have great achievements and a bright future.

Feng Binglong

Vaasa, Finland

08/05/2016

Keywords

NAO robot, computer vision, OpenCV, Python
This thesis is an application about the computer vision and image comparing strategy in the NAO robot, whose main purpose is to enable the NAO Robot to play “Spot the Differences” game autonomously and precisely. The whole application consists of a strategy module and a behavior module. The strategy module is the main part of the thesis which contains the perspective transformation method, image slicing process in python, template matching algorithm and the simulated mouse clicking. Behavior module is the behavior designing of the NAO robots throughout the entire application. The result of this application is that NAO can find out all four different places between two pictures and complete the game correctly. Also by using these comparing strategies, NAO can do it much faster than human beings with satisfactory answers.

The 5th generation of the Aldebaran Robotics company’s NAO robot is used in this thesis and the programming language mainly used is Python. The entire application is based on the Window 10 operating system on the PC together with the NAOqi operating system running on the NAO Robot. Opencv version 2.4.12 was chosen to be the software for processing the images and Choregraphe 2.1.3 was used to design the behaviors.
## CONTENTS

FOREWORD

ABSTRACT

1 INTRODUCTION......................................................................................................7

   1.1 Background Introduction.................................................................................. 7
   1.2 Overall Structure.............................................................................................. 8

2 PROBLEM DEFINITION.........................................................................................10

   2.1 Requirements Analysis................................................................................... 10
   2.2 Used Technologies.......................................................................................... 11
       2.2.1 Python.................................................................................................... 11
       2.2.2 OpenCV.................................................................................................. 11
       2.2.3 Aptana Studio 3....................................................................................... 11

3 PROPOSED SOLUTION...........................................................................................12

4 STRATEGY MODULE............................................................................................14

   4.1 Withdrawing Two Pictures And Saving To Separate Files.............................. 14
   4.2 Divide The Pictures Into Several Pieces And Do The Comparison For Each Area .................................................................................................................. 17
   4.3 Click On The Most Different Blocks With Simulate Mouse......................... 19

5 BEHAVIOR MODULE...........................................................................................19

6 TROUBLESHOOTING..........................................................................................22

   6.1 Filter the noise............................................................................................... 22
   6.2 Brute-Force Matcher....................................................................................... 23
   6.3 DrawMatches Function................................................................................... 24

7 FUTURE RESEARCH............................................................................................27

   7.1 Improve the Algorithm of Image Comparison.............................................. 27
   7.2 Implement Real Time Comparison From the Camera...................................... 27

8 CONCLUSION........................................................................................................28

REFERENCES........................................................................................................29

APPENDICES
### LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RoboCup</td>
<td>Robot Soccer World Cup</td>
</tr>
<tr>
<td>OpenCV</td>
<td>Open Source Computer Vision Library</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>PIL</td>
<td>Python Imaging Library</td>
</tr>
<tr>
<td>RGB</td>
<td>Red, green and blue</td>
</tr>
<tr>
<td>HSV</td>
<td>Hue, Saturation and Value</td>
</tr>
<tr>
<td>VGA</td>
<td>Video Graphics Array</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SIFT</td>
<td>Scale-Invariant Feature Transform</td>
</tr>
<tr>
<td>ORB</td>
<td>Oriented FAST and Rotated BRIEF</td>
</tr>
<tr>
<td>BF</td>
<td>Brute-Force</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES AND TABLES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>NAO Robots</td>
<td>P.9</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Flow Chart of the Whole Application</td>
<td>P.11</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Python Setup</td>
<td>P.13</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Opencv Setup</td>
<td>P.14</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Pynaoqi Setup</td>
<td>P.15</td>
</tr>
<tr>
<td>Figure 6</td>
<td>PIL Setup</td>
<td>P.16</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Webots Setup</td>
<td>P.16</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Aptana Studio Installation</td>
<td>P.17</td>
</tr>
<tr>
<td>Figure 9</td>
<td>All Packages Installed Correctly</td>
<td>P.18</td>
</tr>
<tr>
<td>Figure 10</td>
<td>NAO Resting Posture</td>
<td>P.19</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Result of camImage</td>
<td>P.21</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Set Corner Points</td>
<td>P.22</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Result After Perspective Transformation</td>
<td>P.24</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Last Result of Image Comparison</td>
<td>P.26</td>
</tr>
<tr>
<td>Figure 15</td>
<td>NAO Standing Posture</td>
<td>P.29</td>
</tr>
<tr>
<td>Figure 16</td>
<td>NAO Point at Screen</td>
<td>P.30</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Result of Feature Matching</td>
<td>P.34</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Result of Imagechop</td>
<td>P.35</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

1.1 Background Introduction

NAO robots were developed by a French robotic company named Aldebaran Robotics whose headquarter is located in Paris. NAO robot is a small size programmable robot which is the first generation of the humanoid robot invented by the company. Its body has 25 degrees of freedom, electric motors and actuators are the mainly constructor of its hardware. Now NAO is widely used in most of the countries all around the world for both educational and research purposes.

Figure 1. NAO robots

The project in terms of developing the NAO robot can be traced from 2005. Till now, the NAO robot has become one of the most common learning and teaching tools in many academic institutions. In August 2007, the NAO robot replaced Sony's puppy robot Aibo and became the standard platform of the RoboCup (Robot Soccer World Cup), which is a well-known robotic cup event. The Bothnia robot team representing VAMK has been participating in RoboCup world championship since 2006 and ranked top 10 in SSL(Small-Size League), being the only qualified RoboCup SSL team among the whole Nordic countries. Since July 2015, a new type of hotel has been established in a theme park where NAO is
used to provide reception and conciergerie services for the customers. He welcomes the guests by giving them information in plenty of languages.

1.2 Overall Structure

This thesis provides one of the possible methods out of many to make the NAO robot play the game “Spot the Differences” autonomously and smoothly. More specifically, the thesis explains how this procedure is achieved by using OpenCV in Python.

The thesis is divided into eight sections. In section one, the backgrounds of the whole thesis will be introduced, including some basic information about the NAO robot and the motivation of this thesis. The third section introduces the relevant technologies applied in this application such as the programming language and the software. The actual operation part starts from the fourth section which is getting the images from NAO’s camera. The fifth section is the explanation of the algorithms used after retrieving the pictures, including images chopping, key points detection, features matching and so on. The sixth section introduces the behavior module as well as strategy design in playing “Spot the Differences”. The seventh chapter will show some future research related to this thesis and the last section is the conclusion of the entire thesis.
Figure 2. Flow Chart of the Whole Application
As shown in Figure 2, the NAO robot uses a top camera as the input to get an original image. After that the strategy firstly applies perspective transformation to the original image to compare two pictures. Then the application slices each picture into several blocks and uses template matching to find the best matching blocks in two pictures. The simulating mouse click is then used to make robot click at the most different blocks. Feature matching method is the last step to make sure the robot would not click on the blocks with no difference. If there is no difference between the two pictures robot would say “I Can Not Find the Differences”. Otherwise the NAO robot would point at the screen and say “Yes, I Found Them!” instead.

2 PROBLEM DEFINITION

2.1 Requirements Analysis

In order to make the NAO robot play the game “Spot the Differences” successfully, the coordinates of the different places have to be returned by the application before the simulate mouse can click on the certain areas.

Images are the combination of many pixels, each pixel has its own value. The numpy array of the certain image object contains all the information of the image and it can represent the image. Before the image numpy array comparing step, the perspective transformation must be applied on the two pictures first.

Because any image can be represented by a numpy array which specifies the value of the pixel in certain rows and columns of the picture. So the differences parts between the two pictures is actually the pixels in the corresponding places with different values. If the pixels which are in the same position of the two pictures have the same value, these places are exactly the same. But in different parts the values of pixels have different values. Returning the coordinates of pixels which have the different value can enable the NAO robot to detect the different places between two pictures.
On the current stage, since the camera of the robot is the only input of the program, the relative distance between the robot and the screen can not be changed. The coordinates of the corner points must be set as default values in the application to enable the robot calculate the coordinates of the different parts.

2.2 Used Technologies

2.2.1 Python

The reason for choosing Python as the main programming language in this thesis is that it is earlier to get start with and more straight forward compared to other possible languages such as C++, Java and C#. Although the execution speed of the C++ code is much faster than the Python code when completing complicated tasks.

2.2.2 OpenCV

The full name of OpenCV is: Open Source Computer Vision Library. OpenCV is a cross-platform computer vision library based on the open source. It can run on most of the operating systems such as Linux, Windows and Mac OS. It is a lightweight and efficient software which contains a series of C functions and a few C++ classes, while providing the development interfaces of Python, Ruby, MATLAB and other languages.

2.2.3 Aptana Studio 3

Aptana is a very powerful open source software for developing in the Python IDE. It was chosen to be the Python SDK for this application because it is more independent and lightweight than the Eclipse to which need to add a Pydev.
3 PROPOSED SOLUTION

At first the most direct and efficient method in theory was used which is to compare all the pixels’ value and fill the pixels which have different values with a certain color for example blue. The codes are as follows:

```python
a = Image.open("camImage1.png")
b = Image.open("camImage2.png")

diff = ImageChops.difference (a, b)

BL = Image.new('RGB', diff.size, 'blue')  # Make a blue layer the same size
BlueDiff = ImageChops.multiply(BL, diff)
Result = ImageChops.blend(BlueDiff, a, 0.1)
Result.save("Result.png","PNG")

imgFile = cv2.imread('Result.png')
```
print("Comparing Differences...")
cv2.imshow('Result', imgFile)

But after the running of these codes, some really unpleasant results were obtained.

![Figure 18](image.png)

**Figure 18.** Result of Imagechop

It can be obviously seen from the image that almost the entire picture is covered with a blue layer. In this case, the different points can not be easily recognized by the robot. As for the reason why this happened is because the two pictures obtained from the perspective transformation method can not just be exactly the same even in the place with no difference.

After reading and searching plenty of materials online, the most suitable method for my case, partial template matching, was finally chosen.
4 STRATEGY MODULE

4.1 Withdrawing Two Pictures And Saving To Separate Files

In order to slice the two pictures to be compared out of the one original picture, the perspective transformation have to be applied to the pictures before the comparing process.

Firstly cv2.imread was used to get the mat object of the original picture and the image information like width, height, and channels.

```python
img = cv2.imread('camImage.png')
rows, cols, ch = img.shape
```

After that the four corner points(red points showed in the picture below) of each picture to be compared would be set because these two original pictures were supposed to be rectangular on the screen.

![Set Corner Points](image)

**Figure 12.** Set Corner Points

Then the matplib, a very powerful image drawing library, was used to get the coordinates of these eight points. The cv2.getPerspectiveTransform function took
the coordinates of these points as the arguments to get the M transformation value for both left picture and right picture.

```python
pts1 = np.float32([[53,96],[273,100],[52,374],[268,376]])
pts2 = np.float32([[348,103],[557,110],[342,378],[547,378]])
pts3 = np.float32([[0,0],[414,0],[0,532],[414,532]])

M1 = cv2.getPerspectiveTransform(pts1,pts3)
M2 = cv2.getPerspectiveTransform(pts2,pts3)
```

The pts3 is the coordinates of the four rectangle corner points the user wanted to transfer the image to. So the image gotten from the transformation is a 414*532 image.

cv2.getPerspectiveTransform(src,dst) function has two parameters:

- src – Coordinates of quadrangle vertices in the source image.
- dst – Coordinates of the corresponding quadrangle vertices in the destination image.

The function calculates the 3x3 matrix of a perspective transform so that:

```
\[
\begin{bmatrix}
    t_i x'_i \\
    t_i y'_i \\
    t_i \\
\end{bmatrix} = \text{map\_matrix} \cdot \begin{bmatrix}
    x_i \\
    y_i \\
    1 \\
\end{bmatrix}
\]
```

Where \( dst(i) = (x'_i, y'_i), src(i) = (x_i, y_i), i = 0,1,2,3 \)

After that, the transformation with cv2.warpPerspective function will be applied according to the M value of each picture. M is the map_matrix in the equation.

```python
dst1 = cv2.warpPerspective(img,M1,(414,532))
dst2 = cv2.warpPerspective(img,M2,(414,532))
```

The function warpPerspective transforms the source image using the specified matrix:
\[ \text{dst}(x, y) = s \begin{pmatrix} M_{11}x + M_{12}y + M_{13} \\ M_{21}x + M_{22}y + M_{23} \\ M_{31}y + M_{32}y + M_{33} \end{pmatrix} \]

```python
im1 = Image.fromarray(dst1)
im2 = Image.fromarray(dst2)
im1.save('camImage1.png')
im2.save('camImage2.png')
```

**Figure 13.** Result After Perspective Transformation

Lastly just get the image from the mat object array and save them under the root directory of the whole application for later use.
It can be seen from the saved pictures that they are both in the HSV color mode and they must be transferred to Gray Scale for the better comparison in the next step.

### 4.2 Divide The Pictures Into Several Pieces And Do The Comparison For Each Area

By calling cv2.imread function and adding a 0 argument, the image would be transferred into Gray Scale automatically.

```python
def block_diff(img1, img2, nrow, ncol):
    h, w = img2.shape
    dh = h // nrow
    dw = w // ncol
    h2 = dh * nrow
    w2 = dw * ncol
    diff = np.zeros((h2, w2))
    for i, j in product(range(nrow), range(ncol)):
        sy = i * dh
        sx = j * dw
        slice_index = np.s_[sy:sy+dh, sx:sx+dw]
        block = img2[slice_index]
        match = cv2.matchTemplate(img1, block, cv2.TM_CCORR_NORMED)
        offset_x, offset_y = cv2.minMaxLoc(match)[-1]
        original = img1[offset_y:offset_y+dh, offset_x:offset_x+dw]
        np.abs(block.astype(np.float) - original, diff[slice_index])
    return diff

diff = block_diff(img1, img2, 9, 7)
plt.figure(figsize=(16, 12))
```
plt.imshow(diff, cmap="gray")
plt.show()

The application would slice each image into several blocks according to the width and height of the picture. In this case, since the resolution of images gotten from the perspective transformation is 414*532, it could be divided into 7*9 square blocks whose side length is 60.

Then the function of cv2.matchTemplate and cv2.minMaxLoc(match) are used to find the most matching block in image2 for each block in image1. The absolute value of difference will be used to figure out the four most different blocks because there are only four different places in the original picture.

![Figure 14. Last Result of Image Comparison](image.png)
The most white parts and black parts showed in the picture are the differences of two images.

4.3 Click On The Most Different Blocks With Simulate Mouse

```python
from pymouse import PyMouse
m = PyMouse()

m.position() #gets mouse current position coordinates
m.click(X1,Y1,1) #the third argument "1" represents the mouse button
m.click(X2,Y2,1)
m.click(X3,Y3,1)
m.click(X4,Y4,1)
```

The X1, Y1 is the coordinate of the central point of the block with most differences. (X2,Y2), (X3,Y3), (X4,Y4) are the other 3 blocks’ central point coordinates.

This is the entire content of the strategy module.

5 BEHAVIOR MODULE

This module is the designing of how the NAO robot would behave throughout the execution of the whole application.

In this module three proxies would be called: ALMotion, ALRobotPosture, ALTTextToSpeech. ALMotion is the responsible for the moving actions of the robot while the ALRobotPosture controls the posture status of NAO. The TTS(Text To Speech) proxy need to be called to make NAO say something.

```python
motionProxy = ALProxy("ALMotion", robotIP, PORT)
postureProxy = ALProxy("ALRobotPosture", robotIP, PORT)
tts = ALProxy("ALTextToSpeech", robotIP, PORT)
```
tts.say("Let Me See...")

# Wake up robot

motionProxy.wakeUp()

# Send robot to Stand Init

postureProxy.goToPosture("StandInit", 0.5)

frame = motion.FRAME_ROBOT
useSensor = False

effectorInit = motionProxy.getPosition(chainName, frame, useSensor)

# Active RArm tracking

isEnabled = True

motionProxy.wbEnableEffectorControl(chainName, isEnabled)

coef = 1.0

if chainName == "LArm":
    coef = +1.0

elif chainName == "RArm":
    coef = -1.0

targetCoordinate = [+0.20, -0.10*coef, +0.80]

# wbSetEffectorControl is a non blocking function

# time.sleep allow head go to his target
motionProxy.wbSetEffectorControl(chainName, targetCoordinate)
motionProxy.post.openHand('RHand')
tts.say("Yes. I Found Them!")
time.sleep(4.0)

# Deactivate Head tracking
isEnabled = False
motionProxy.wbEnableEffectorControl(chainName, isEnabled)

# Go to rest position
motionProxy.rest()

When the application begins, the NAO robot would say ‘Let Me See...’ first, then it would be waked up by calling motionProxy.wakeUp(). After that postureProxy.goToPosture("StandInit", 0.5) is used to set the NAO to initial standing posture. The initiation steps must be done before further movement execution because the state of capturing the images is a resting posture and remember that the whole body balancer must be inactivated at the end of the script.

After finding out all the differences in two pictures and clicking on all of them with the virtual mouse, the NAO robot would start to lean his body forward and rise his right arm from the initial place to a certain preset height. Moreover, the robot would open his right hand with fingers pointing at the screen and saying ‘Yes, I Found Them!’

Lastly, the NAO robot would go back to its initial resting posture and the whole execution came to an end.
6 TROUBLESHOOTING

6.1 Filter the noise

Applying Gaussian Filtering to the image1 and image2 before the feature matching process is to remove the Gaussian noise in the picture and minimize the errors of mismatching.
# Gaussian Filter

```python
blur1 = cv2.GaussianBlur(img1,(1,1),0)
blur2 = cv2.GaussianBlur(img2,(1,1),0)
```

(1,1) is the width and height of the kernel which must be positive odd numbers. The smallest kernel was chosen to get the best result in this application. The third argument 0 means the standard deviation in the X and Y directions are calculated from size of kernel set by users.

## 6.2 Brute-Force Matcher

Firstly creating an ORB (Oriented FAST and Rotated BRIEF) object is needed.

Brute-Force matcher is a simple matching algorithm because it only uses few distance calculation to take the descriptor of one feature in the first set and is matched with all other features in the second set. And only the closest one is returned.

In order to use BF matcher, a BFMatcher object needs to be created by using the `cv2.BFMatcher()` method. It takes two optional params. The first one is `normType`. It specifies the distance measurement to be used. Since the descriptor used in ORB is binary string based, `cv2.NORM_HAMMING` is supposed to be used as the first param. The second param is boolean variable, `crossCheck`, whose default value is false. The true means the two features in both sets should match each other and it can replace the ratio test in the SIFT method because it can provide consistent result.

After that, `BFMatcher.match()` will return only the best result, however `BFMatcher.knnMatch()` can return the k best matches where k is specified by the user. The result of `matches = bf.match(des1,des2)` line is a list of DMatch objects. This DMatch object has following attributes:

- `DMatch.distance` - Distance between descriptors. The lower, the better it is.
- DMatch.trainIdx - Index of the descriptor in train descriptors
- DMatch.queryIdx - Index of the descriptor in query descriptors
- DMatch.imgIdx - Index of the train image.

```
# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)
```

With this code the best matches (with low distance) would come to the front.

### 6.3 DrawMatches Function

The cv2.drawMatches module is not included in the python library of OpenCV 2.4.12. So a function named drawMatches had to be defined first and the code of this function is as follows:

```
def drawMatches(img1, kp1, img2, kp2, matches):
    # Create a new output image that concatenates the two images together
    rows1 = img1.shape[0]
cols1 = img1.shape[1]
rows2 = img2.shape[0]
cols2 = img2.shape[1]

    out = np.zeros((max([rows1,rows2]),cols1+cols2,3), dtype='uint8')

    # Place the first image to the left
    out[:rows1,:cols1] = np.dstack([img1, img1, img1])

    # Place the next image to the right of it
    out[:rows2,cols1:] = np.dstack([img2, img2, img2])
```
# For each pair of points we have between both images
# draw circles, then connect a line between them
for mat in matches:
    # Get the matching keypoints for each of the images
    img1_idx = mat.queryIdx
    img2_idx = mat.trainIdx

    # x - columns
    # y - rows
    (x1,y1) = kp1[img1_idx].pt
    (x2,y2) = kp2[img2_idx].pt

    # Draw a small circle at both co-ordinates
    # radius 4
    # color green
    # thickness = 1
    cv2.circle(out, (int(x1),int(y1)), 4, (0, 255, 0), 1)
    cv2.circle(out, (int(x2)+cols1,int(y2)), 4, (0, 255, 0), 1)

    # Draw a line in between the two points
    # thickness = 1
    # color green
    cv2.line(out, (int(x1),int(y1)), (int(x2)+cols1,int(y2)), (0, 255, 0), 1)

    # Show the image
    cv2.imshow('Matched Features', out)
cv2.waitKey(0)

return out

This function has three main steps:

- Combine the two images together in gray scale, one is on the left and the other one is on the right.
- Draw key points on both of the pictures.
- Draw green lines to connect all the matched features.

Here is the result of the two pictures after feature comparing:

![Matched Features](image)

**Figure 17.** Result of Feature Matching

These matched points are the detected feature points so these places are not the different parts, which can make sure that the places is right.
7 FUTURE RESEARCH

7.1 Improve the Algorithm of Image Comparison

There are some restrictions for this application and the core part is the algorithm in image comparing. In most cases, the difference value is much bigger than the error values created by perspective transformation. So that the robot can figure out the four different places. But in some specific cases, if the difference value is really small which is even smaller than the error value, then the robot would regard these error values as the real difference value of the pictures. So the algorithm used in the thesis is not perfect yet and needs to be improved in the future.

7.2 Implement Real Time Comparison From the Camera

In the current stage, the application can only retrieve images from the camera by taking photos. Future research could make it show the differences in real time, in other words, the results can be shown in the robot’s vision window constantly instead of taking photos, just like the human face detection which has been commonly used in photography.


8 CONCLUSION

This thesis introduces one possible way to make the NAO robot play the Find Differences Game autonomously and precisely. How this specific application works is presented at length as well.

To overcome the challenging problem of the thesis, it was necessary to divide it into several small parts. The main process consists of three steps: getting the images from the NAO’s camera, using some certain strategies for image comparing and the behavior designing.

Firstly, the most basic steps of configuring the Python environment to the PC and retrieving images in the NAO robot were finished as planned. But obstacles occurred in the second stage where to find a suitable algorithm for the image comparing procedure. The result of the image chop method was unsatisfactory. The research of different methods related to image processing and computer vision resulted in the use of the template matching method. Unfortunately the initial application of this method gave unsatisfactory results because template matching algorithm is applied to the entire data of the image.

To overcome this problem, the image was firstly divided into specific parts, then the template matching method was applied to each block. At last, the four different places were correctly detected and spotted out by the robot. But there are still some areas for the improvements to make it suitable for any case in the future.
REFERENCES


http://robotics.puv.fi/


http://doc.aldebaran.com/2-1/dev/naoqi/index.html

http://doc.aldebaran.com/2-1/software/choregraphe/choregraphe_overview.html

https://en.wikipedia.org/wiki/Transformation_matrix#Perspective_projection

/7/ Introduction of Brute-Force Matcher Accessed 05.03.2016

/8/ Perspective Transform Accessed 07.03.2016
## APPENDICES

### 1. List of Functions and Modules Used

<table>
<thead>
<tr>
<th>Modules And functions</th>
<th>The Information of Modules and Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>numpy</td>
<td>NumPy is the basic package for scientific computing with Python. It contains a N-dimensional array object, some broadcasting functions, couple of tools for integrating C/C++ and Fortran code and some useful linear algebras.</td>
</tr>
<tr>
<td>scipy</td>
<td>SciPy stands for Scientific Computing Tools for Python and it is a collection of core packages for scientific computing with python, providing several numerical routines.</td>
</tr>
<tr>
<td>cv2</td>
<td>A cross-platform computer vision library based on the open source.</td>
</tr>
<tr>
<td>time</td>
<td>Provides various kinds of time related functions.</td>
</tr>
<tr>
<td>matplotlib</td>
<td>Matplotlib is a 2D plotting library in python which can produce publication quality figures in plenty of hard copy formats and interactive environments across a variety of platforms.</td>
</tr>
<tr>
<td>ALProxy</td>
<td>An object which enable you to call the methods and modules on NAO robot.</td>
</tr>
</tbody>
</table>
PIL stands for Python Imaging Library, which can add image processing capabilities to the Python interpreter. This library supports many file formats and provides powerful image processing and graphics capabilities.

<table>
<thead>
<tr>
<th>getPerspectiveTransform</th>
<th>A function included in opencv2 to calculate the value M for the perspective transformation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>matchTemplate</td>
<td>A function included in opencv2 to find the most matching place of a block in another picture.</td>
</tr>
<tr>
<td>minMaxLoc</td>
<td>A function included in opencv2 to find the location of maximum and minimum values in the matrix.</td>
</tr>
</tbody>
</table>

2. User Guide

- Copy the whole application to the working space of your Python IDE.
- Change the IP address of the NAO robot with your NAO robot’s IP.
- Change the coordinates of the blocks with your values measured from the image.
- Run the application.