VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Khanh Mai Trung

Human gesture imitation using the whole body of a Nao robot and classification with neural networks

Information Technology

2018

ACKNOWLEDGEMENTS

Firstly, I want to give my special thank to my thesis supervisor, Mr. Yang Liu, are from the Department of Information of Technology for all his support and guide through this work. He gave me a huge knowledge and motivation not only to be able to do this thesis but also for many project works.

I would like to thank also to all professors: Mr. Seppo Makinen and Mr. Santiago Chavez for supporting me with all the difficulties with software, sensors, robots, etc. They helped me a lot whenever problems appeared and are my biggest encouragement.

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Degree Program of Information Technology

ABSTRACT

Author	Mai Trung Khanh
Title	Vaasa University of Applied Science
Year	2018
Language	English
Pages	52
Name of Supervisor	Yang Liu

This thesis completed an application for controlling a Nao robot to imitate human gestures and classification using the Kinect sensor. The main goal of the task was to manipulate the whole body of the robot while also keeping the balance and support changes needed. A project work was only about control Nao with only four upper joints of the body part and therefore there was no balancing problem. The Nao robot can become free to use both of its legs and change the support due to human movement. The second section of the thesis implemented convolutional neural network to build a system for training and testing human daily activities to observe the advantages of joint data collected by Kinect sensor.

The version of a Nao robot used was the 5th generation, with a Kinect sensor One (version 2). The main application was built with Python in Visual Studio 2015 providing the communication between Naoqi and Windows. The simulation runs on Webots software which can have a wide range of simulated figures. Using Choregraphe, a free-to-use IDE by Aldebaran, offers a friendly environment to start working with a Nao robot.

The thesis work converges both theoretical and practical experiments in human movement imitation and classification using different approaches and testing performance.

Keywords Kinect, Nao Robot, Deep Learning

CONTENTS

LIST OF FIGURES AND TABLES

LIST OF ABBREVIATIONS

SPL	Standard Platform League
SDK	Software Development Kit
API	Application Programming Interface
COM	Center of Mass
CNN	Convolutional Neural Network

1. INTRODUCTION

1.1 Nao Robot

Nao robot [1] is an autonomous programmable robot developed by a French Robotics company which located in Paris. The use of Nao robot is becoming worldwide because of its personalization and huge capability in education and research. The robot consists of 25 degrees of freedom, electric motors, actuators and sensors to show its flexibility.

Figure 1. Nao Robot /1/

Nao robots have been used in many different types of projects since the launch of Project Nao Robot in 2004. On 15 August 2007, Nao replaced Sony's robot dog Aibo as the robot used in the RoboCup Standard Platform League (SPL), a well-known worldwide robotics competition.

1.2 Microsoft Kinect

Kinect shown in Figure 2 is a motion sensing input device that is provided by Microsoft for Xbox 360. Xbox One and Microsoft Windows PCs. Today, more and more API support for other platforms such as Mac OS and Linux making it more dynamic for developers. Microsoft released the first version of Kinect sensor in

November 2010 and until June 16, 2011 the Kinect software development kit was provided as the first beta. This SDK allows developers to write applications on multiple programming languages such as C++/CLI, C#, or Visual Basic .Net.

Figure 2. Kinect for Xbox One /2/

The SDK for Kinect One (Kinect for Xbox One) using from Visual Studio 10 can provide multiples output using camera, IR sensors, microphone as shown in Figure 3 and Figure 4.

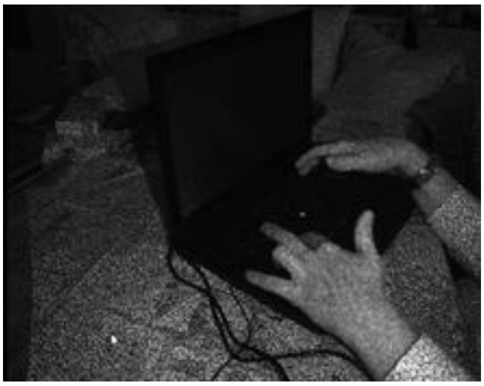


Figure 3. Infrared image shows the laser grid Kinect uses to calculate depth /2/

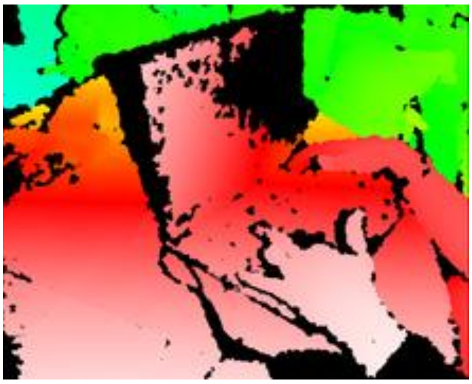


Figure 4. The depth map is visualized here using colors gradients from white (near) to blue (far) /2/

1.3 Overall Structure

This thesis provides a method for implementing a Nao robot and Kinect with human movement. To be specific, this report will explain how to control the robot using the input data from Kinect to imitate human arm gestures. The main programming language is Python using Microsoft Visual Studio.

Firstly, the background of this approach is introduced namely motivation and some information about the Nao robot and the Kinect sensor. The second topic is about all necessary knowledge and technologies. After that, all the solutions and the algorithm will be analyzed and the results captured using pictures and videos. The method of getting the data from Kinect to transfer to the Nao Robot for gestures imitation mothed we mentioned as shown in Figure 5.

After getting the data from the sensor, Nao robot will calculate the desired angle for each joint, also the support polygon and stability need to be obtained. Without the balancer and changing support, Nao Robot can fall down easily.

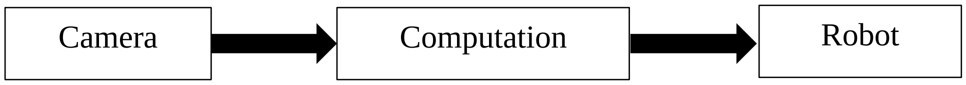


Figure 5. Application Structure

In the second application, the data from the Kinect sensor is sent to a neural network to train and classify with test data using Tensorflow open source framework. Building network steps and testing results are analyzed in detail to observe the performance and advantages of using deep learning as a study approach. Taking the advantage of image classification, a convolutional neural network system is created for a light weight application that still generates a descent result. The data collected by the Kinect sensor is transformed from sequence data in each frame to images for visualization purposes.

2. PROBLEM DEFENITION

2.1 Requirement Knowledge

Human movement imitation for a Nao robot is not an easy task because of the differences between human and robot coordination. The robot needs to get each joint rotation in order to actuate successfully. The challenge also comes from correct imitation and includes support changes for the Nao Robot to stand without any dangerous activity. The second challenge is to classify human movement using deep learning. It requires a lot of knowledge in data processing, building a neural network and tuning parameters to build a perfect system with no overfitting or underfitting.

2.2 Used Technology

2.2.1 Python

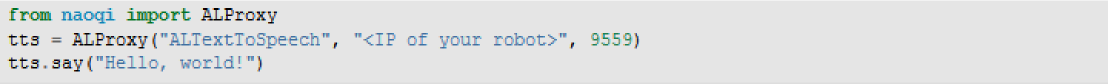
Python is an easy-to-use script programming language but it is still very efficient in matrix computing. In this project lots of dot and cross product will be calculated so using Python will provide a clear and fast matrix and array operation using Numpy (a Python scientific computing package).

2.2.2 Microsoft Visual Studio 2015 Commity

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop websites, web application, mobile application and normal application. Using Visual Studio makes application packages much easier to be controlled.

2.2.3 NAOqi

In this project, in order to communicate with the robot, I will use NAOqi which is an operating system on Nao robot allows controlling the memory and behavior of the robot as shown in Figure 6. NAOqi has C++ API and Python API and for this project, Python API is used for both the robot and the Kinect camera.



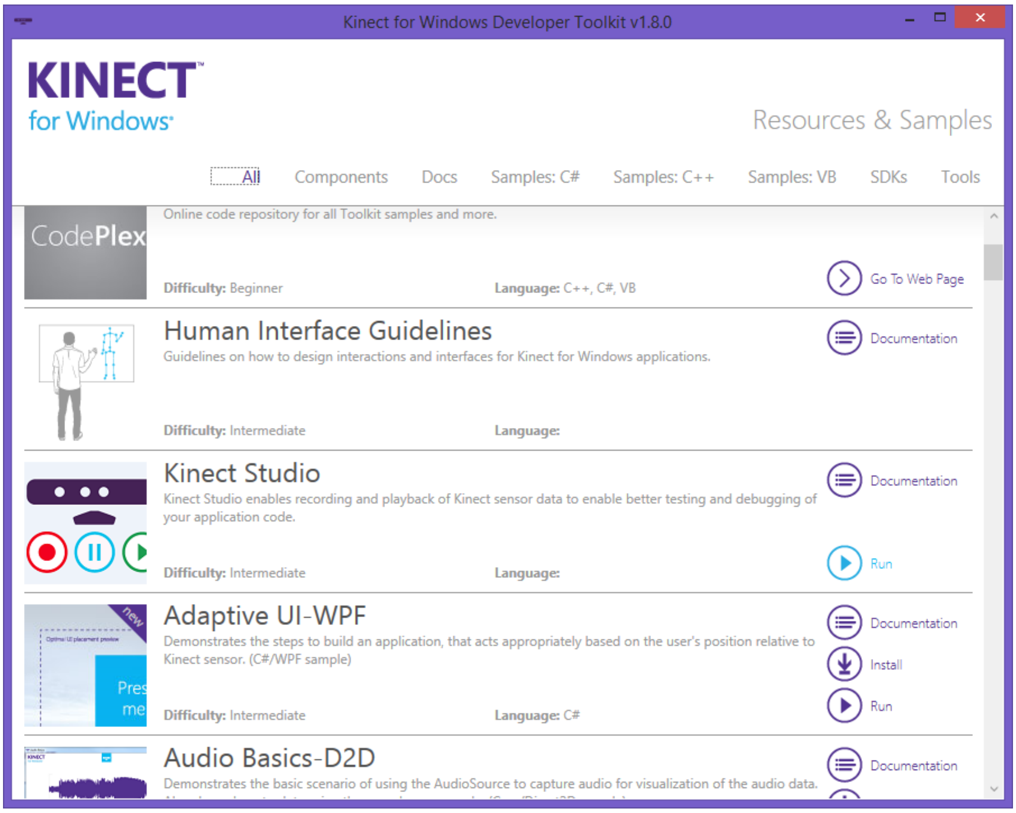
```
from naoqi import ALProxy
tts = ALProxy("ALTextToSpeech", "<IP of your robot>", 9559)
tts.say("Hello, world!")
```

Figure 6. Simple “Hello world” application using NAOqi

Motion with the Nao robot will be important in this project. When using NAOqi, it can be easy to control a joint angle with different angles and speed.

2.2.4 Kinect SDK

Microsoft provides Kinect for Windows SDK 2.0 as shown in Figure 7 which enables developers to write and create applications that support many features such as human gestures, video stream, and voice recognition on computers using Windows 8, Windows 8.1, Windows 10 and Windows Embedded Standard 8 [3,4].



Online code repository for all Toolkit samples and more.

Difficulty: Beginner Language: C++, C#, VB

Human Interface Guidelines

Guidelines on how to design interactions and interfaces for Kinect for Windows applications.

Difficulty: Intermediate Language:

Kinect Studio

Kinect Studio enables recording and playback of Kinect sensor data to enable better testing and debugging of your application code.

Difficulty: Intermediate Language:

Adaptive UI-WPF

Demonstrates the steps to build an application, that acts appropriately based on the user's position relative to Kinect sensor. (C#/WPF sample)

Difficulty: Intermediate Language: C#

Audio Basics-D2D

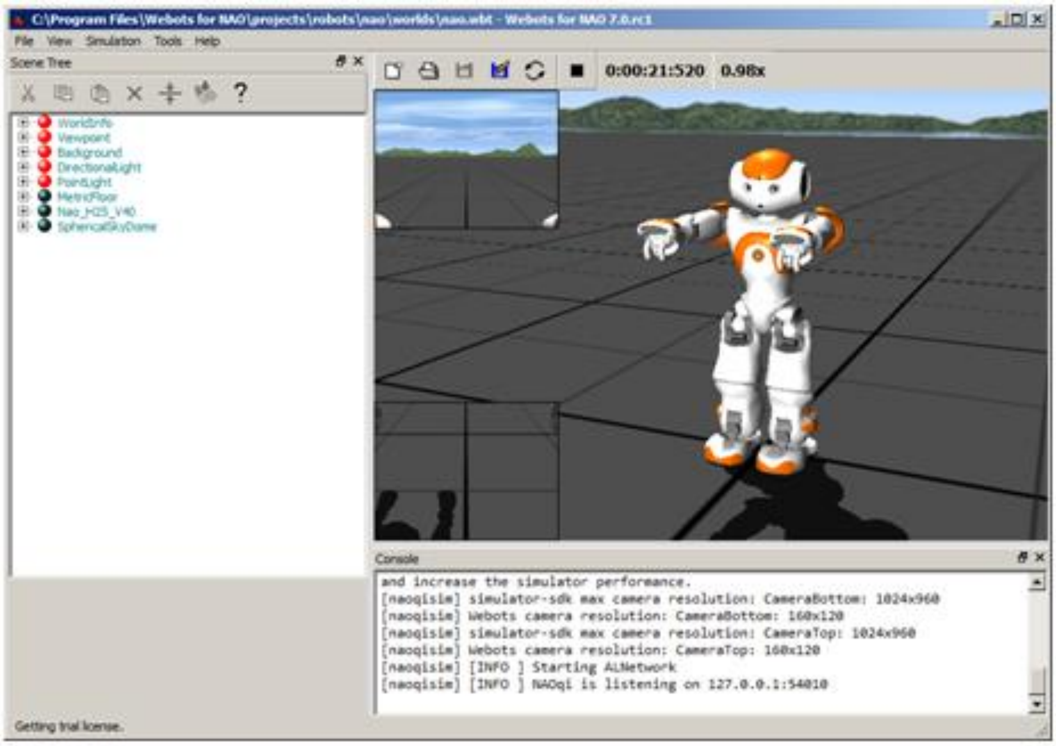
Demonstrates the basic scenario of using the AudioSource to capture audio for visualization of the audio data.

Figure 7. Windows for Kinect SDK /3/

2.2.5 WEBOTS Simulation

Webots is a development system environment for developing and simulating mobile robots. A large amount of sensor, actuators is available to manipulate for each Robot. This software is widely used by over 1329 universities for research and study. With Webots which it is possible to have a Nao robot setup with all features that are required andcan communicate using Naoqi.

In this particular thesis, it is a wise choice to use Webots since this software can detect the falling situation of the robot and can simulate as normal as shown in Figure 8.



```
and increase the simulator performance.
[naoqisim] simulator-sdk max camera resolution: CameraBottom: 1024x960
[naoqisim] Webots camera resolution: CameraBottom: 160x120
[naoqisim] simulator-sdk max camera resolution: CameraTop: 1024x960
[naoqisim] Webots camera resolution: CameraTop: 160x120
[naoqisim] [INFO ] Starting ALNetwork
[naoqisim] [INFO ] NAOqi is listening on 127.0.0.1:54010
```

Figure 8. Webots with Nao robot /4/

To open and program with Nao robot on Webots, the robot environment file is located in the following path: [Webots directory]\projects\robots\nao\worlds. The IP and port of the simulated robot can be used as follows:

IP = 127.0.0.1 PORT = 9559

2.2.6 Vector Operations

Using vector operation when adding two vector A and B creates a new vector with both unique magnitude and direction. Cross product and dot product for 2 vectors: $\vec{a}\ (x_a, y_a, z_a)\ and\ \vec{b}\ (x_b, y_b, z_b)$

$$\vec{a} \times \vec{b} = (y_a z_b - z_a y_b, z_a x_b - x_a z_b, x_a y_b - y_z x_b) \tag{1}$$

$$\vec{a} \,.\, \vec{b} = x_a x_b + y_a y_b + z_a z_b \tag{2}$$

Angle α between 2 vectors: $\vec{a}\ (x_a, y_a, z_a)\ and\ \vec{b}\ (x_b, y_b, z_b)$

$$\alpha = arccos\ \frac{\vec{a}\,.\vec{b}}{|\bar{a}| \times |\bar{b}|} \tag{3}$$

Normal vector of plane P contains 2 vectors : $\vec{a}\ (x_a, y_a, z_a)\ and\ \vec{b}\ (x_b, y_b, z_b)$ equals to their cross product: $\vec{a} \times \vec{b}$.So the angle β between vector $\vec{c}\ (x_c, y_c, z_c)$ and plane has an normal vector $\vec{P}\ (x_P, y_P, z_P)$:

$$\beta = \arcsin(\frac{|\vec{c}\,.\vec{P}|}{|\bar{c}| \times |\bar{P}|}) \tag{4}$$

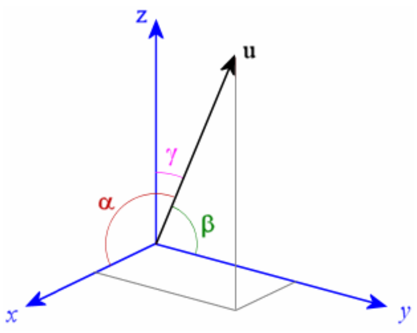


Figure 9. Angle between two vectors /5/

2.2.7 Center of Mass and Support Polygon

Center of mass is a position that can be defined relative to and object or and system of objects. It can be calculated as the average position of all part of the systems according to their mass. With a complex of shape, the center of mass can be understood as a position where the sum of all relatively weighted position vector equals to zero. According to /6/, if we have a system with several particles: $P_i, i = 0,1 ,..,n$ with each mass m_i located in position r_i, so the position R of COM follows the condition:

$$\sum_{i=1}^{n} m_i \,(r_i - R) = 0 \qquad (5)$$

Support polygon is a horizontal region that COM laid on to achieve static stability. For instance, when an object is located on a table which is a horizontal surface as shown in Figure 10.

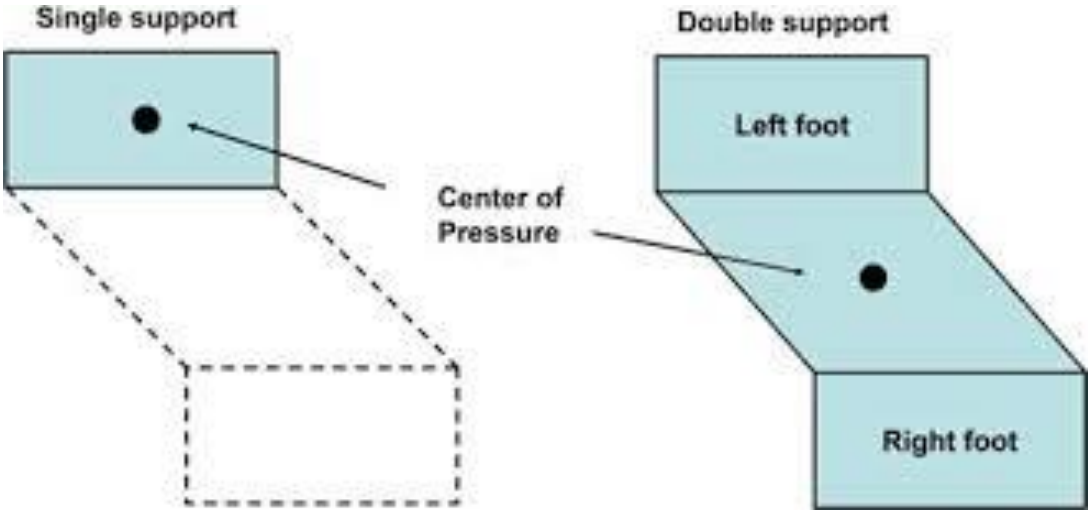


Figure 10. Relationship between COM and support polygon /6/

2.2.8 Convolutional Neural Network

Neural networks or artificial neural networks represent systems that can be inspired by human brains. Neural networks are created to learn and do tasks using many units or nodes called neurons and each neuron is related to each other to process data. Today, deep learning and artificial intelligence take a significant role in modern world development on many areas such as engineering, biology and chemistry, for example. Several kinds of neural networks are implemented in research depending on the types of input and output wanted.

In this thesis work Convolutional Neural Network /7/ is used as the network system for gestures classifications. CNN neural consists of learnable weights and biases, they receive some inputs and execute computations with activation function such as TanH, Sigmoid and ReLU. Finally, a fully connected layer is applied to find the predictions. Convolutional neural networks have a huge advantage of image processing and classification as the layers of ConvNet consist of three channels of width, height and depth as shown in Figure 11.

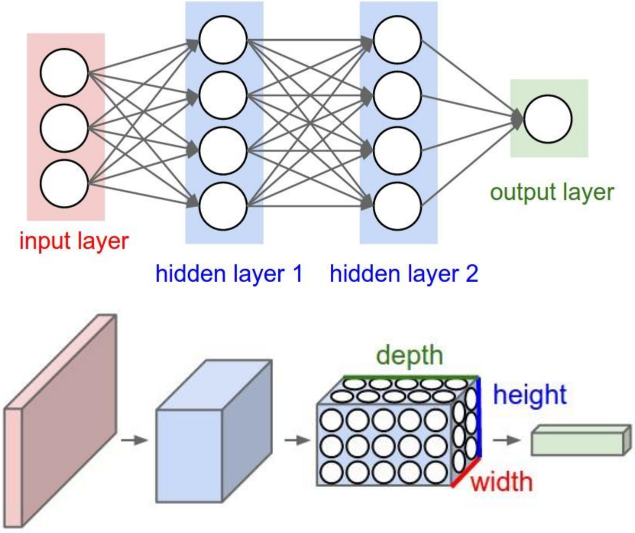


Figure 11. Example of Convolutional Neural Network. /7/

As said above, a CNN network is a sequence of processing data layers. Through each channel, data is transferred and computed with given inputs, weights and biases. There are two main parameters that configure the behavior of its layer. The way of choosing the padding and stride is a significantly important step of convolutional layers.

Stride takes the role of how many units should be shifted by the filter through the input at a time. Normally, the stride is used to prevent channel overlaps and get the smaller dimension for the output. For example, if the stride is set to 1, the output volume should be as shown in Figure 12.

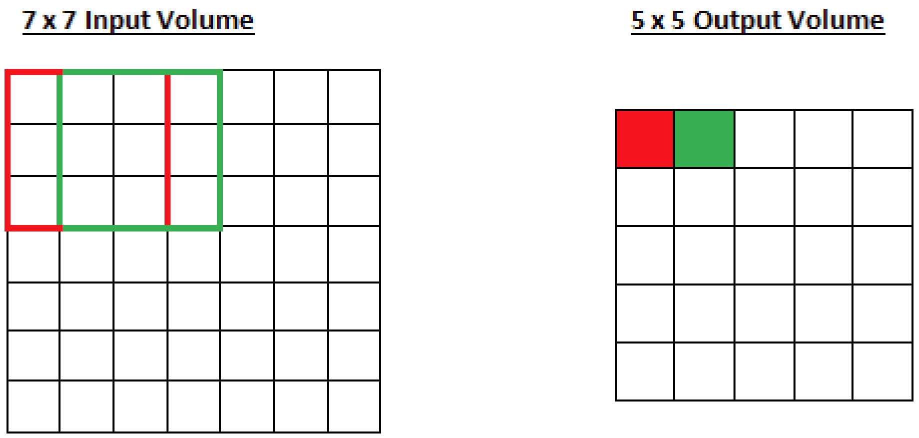


Figure 12. Stride in neural network. /8/

The second main parameter is the padding, as shown in Figure 13, which means creating the borders for the input before passing it to the channels as programmers do not want to have the size of the output decrease more then they want as it can makes the result underfitting.

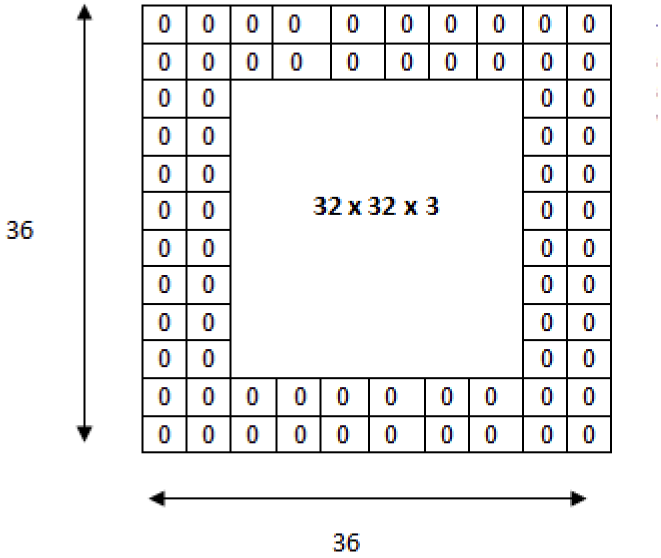


Figure 13. Padding in neural network./8/

When both of the padding and the stride are set, the output size can be calculated with the following formula:

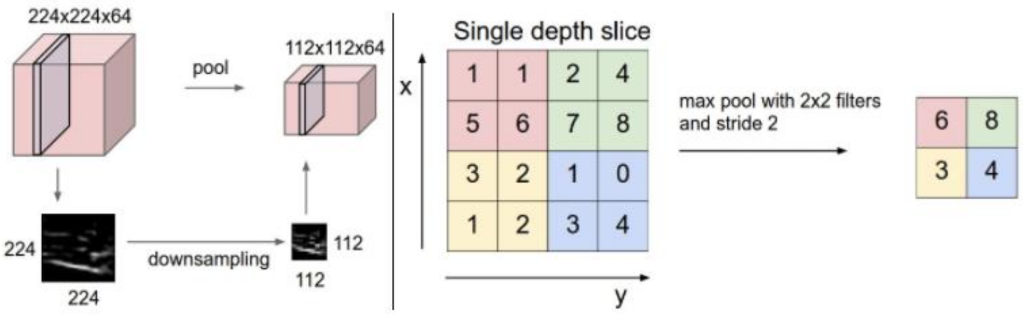
$$Output = \frac{(W-F+2P)}{S} + 1 \tag{6}$$

Where W is the input weight/height, F is the filter size, P is the padding and S is the stride.

In convolutional neural networks, the input needs to go through several layers to reduce the dimensions or size. Each layer has a different purpose to scale the data as required. Firstly, a ReLU layer is a non-linear activation function such as a rectified linear unit to calculate the output of each element where the result is zero then value is less than zero. Note that this layer will not change the size of the input.

$$f(x) = \begin{cases} x\,, x \geq 0 \\ 0\,, x < 0 \end{cases} \tag{7}$$

After the ReLU layers, applying pooling layers is necessary for droping out the size of input using padding and stride to control overfitting. The most used pooling layer is maxpooling as shown in Figure 14.



1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

6	8
3	4

Figure 14. Maxpooling layer. /7/

Next, using fully connected layers which connect all the input from the previous layer and reduce to the size after several convolutional and pooling layers.

On the training process, the main purpose is to minimize the cost by updating parameters for features. Generally neural network systems can have several kinds of optimizations such as Gradient Descent, Adagrad, Adam, for example. After long research and investigation, Adam (Adaptive Moment Estimation) optimizer as shown in Figure 16 is chosen for network optimization since it works well in practice and performs fasters compared to other technique as shown in Figure 15.

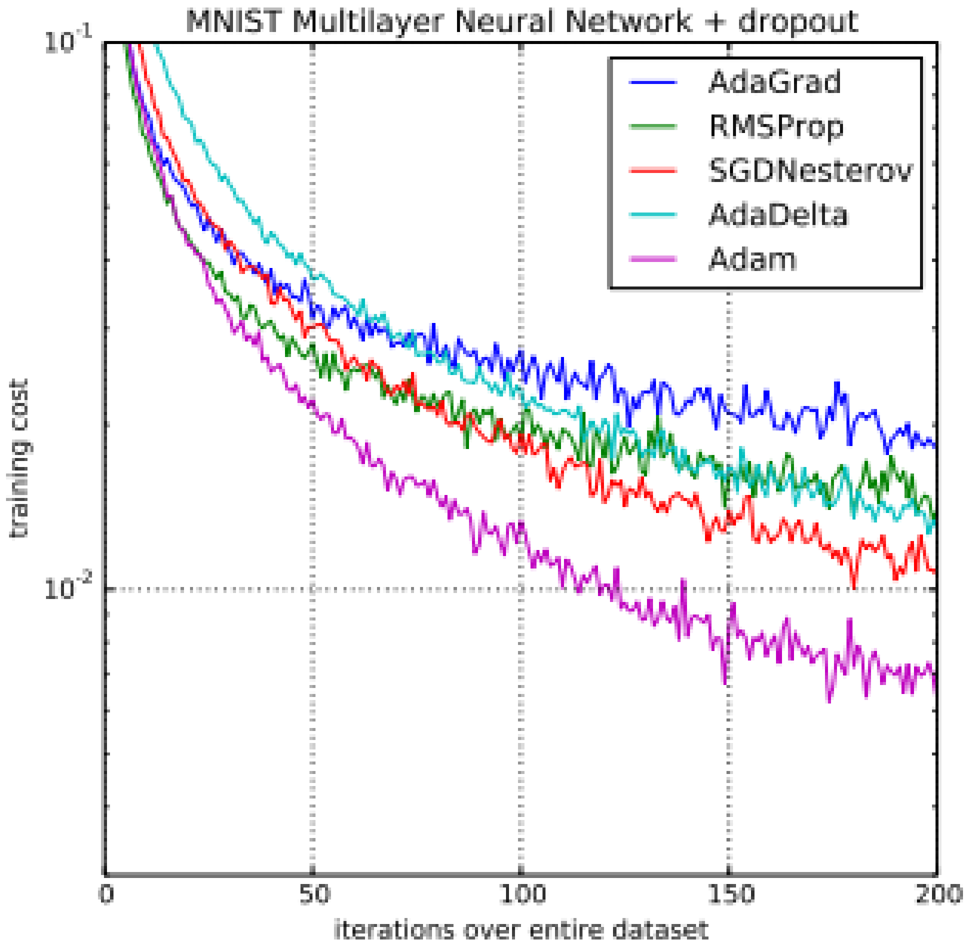
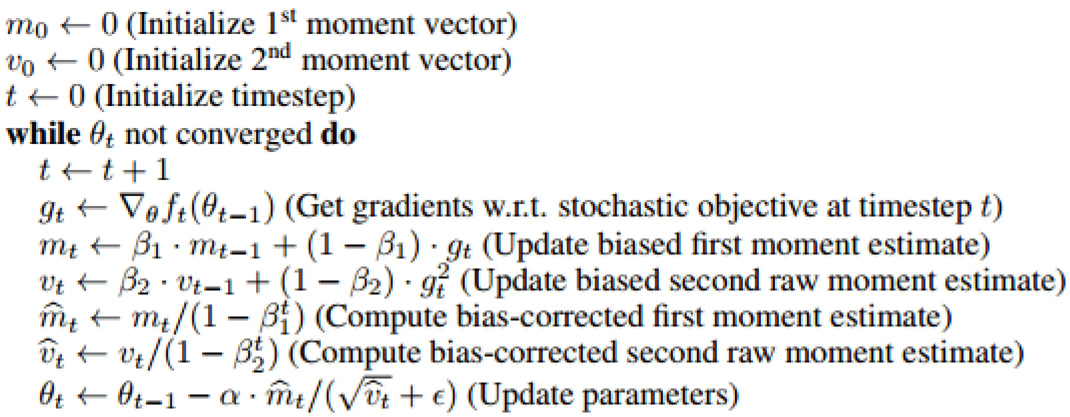


Figure 15. Comparison between Adam and other optimizers /9/



```
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while $\theta_t$ not converged do
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
  $\widehat{m}_t \leftarrow m_t/(1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
  $\widehat{v}_t \leftarrow v_t/(1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
  $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t/(\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
```

Figure 16. Adam Optimizer equations. /10/

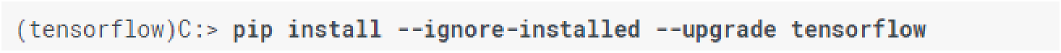
On the notations, α is the learning rate for each step, β_1 is the exponential decay for the first moment estimates while the β_2 is for the second estimates. We need to add ϵ as a very small number to prevent any division be zero in this formula. All the parameters will be initialized and configured as default.

2.2.9 Tensorflow API

Tenssorflow /15/ is an open source library that supports high-performance computation in artificial intelligence, allows developers to deploy developments in multiple platforms and from servers to mobile devices. Tensorflow was developed by engineers and researchers from the Google Brain team within Google's AI organization supporting machine learning and deep learning. This open source library is now being implemented by a variety of technology companies such as Airbnb, AMD, Google and Intel.

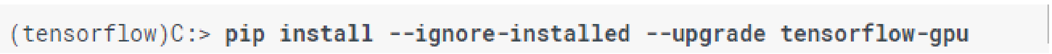
Tensorflow is built and tested in MacOS, Windows and Linux and also developed from source code. The API from Tensorflow now supports multiple programming languages make developers more dynamic. In the latest version 1.7, Tensorflow provides API for C/C++, Java, Python. Go, C# and Javascript for both software developments and now also with web applications.

In this particular thesis work, Tensorflow is installed in Python language using Anaconda Python Notebook for more dynamic practical training and testing. The installation with Anaconda can be shown as follow:



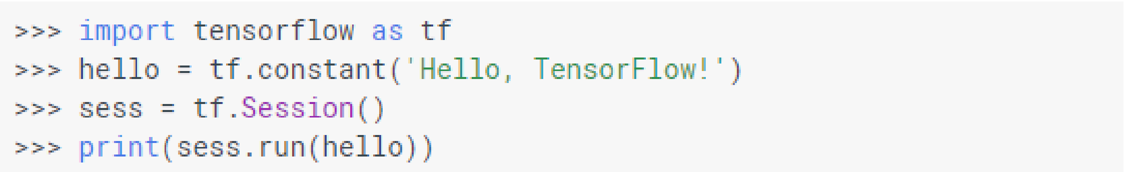
```
(tensorflow)C:> pip install --ignore-installed --upgrade tensorflow
```

This will install the CPU-only version of Tensorflow. When using a powerful GPU, the GPU version can be installed to improve the performance:



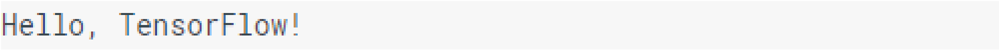
```
(tensorflow)C:> pip install --ignore-installed --upgrade tensorflow-gpu
```

After the installation, the activation can be done through Anaconda environment and do the testing a short program inside the python shell to observe the connection:



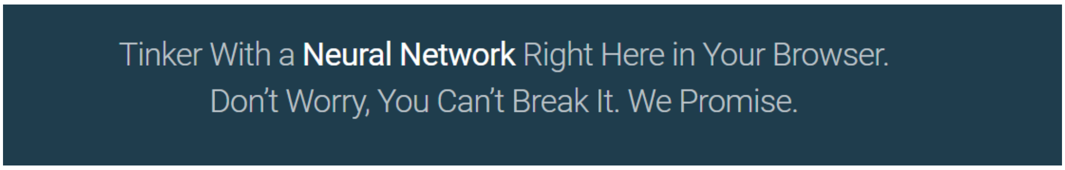
```
>>> import tensorflow as tf
>>> hello = tf.constant('Hello, TensorFlow!')
>>> sess = tf.Session()
>>> print(sess.run(hello))
```

The system will output the following:



```
Hello, TensorFlow!
```

For beginners, when building a simple neural network, programmers can use Tensorflow Playground /11/ as shown in Figure 17 which is developed by Tensorflow's developers and which allows you to tune parameters to see the output such as test loss and training loss.



Tinker With a **Neural Network** Right Here in Your Browser.
Don't Worry, You Can't Break It. We Promise.

Figure 17. Tensorflow Playground /11/

Both classification and regression can be tested, with dynamic learning rate, activation type, etc. It is possible to add more hidden layers with more neurons. The performance is real-time can be visualized during each epoch.

For example, in Figure 18, when implementing the classification neural network using learning rate as 3 and Tanh activation function, because the learning rate is too high, the cost became unstable and could not reach the local minimum.

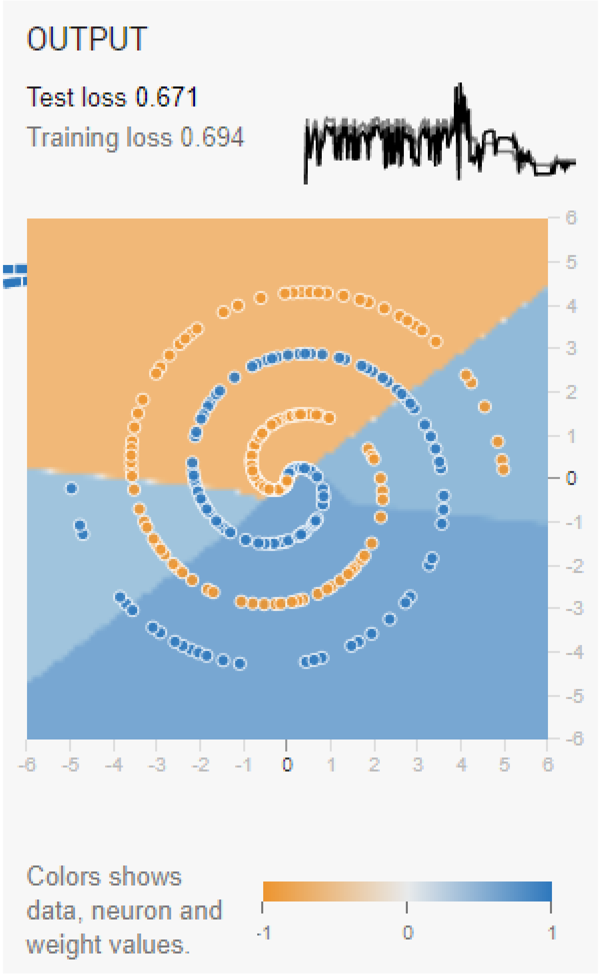


Figure 18. Large learning rate experience.

With the great work of Tensorflow community, training and testing from simple to complex networks becomes visualized and tuned.

3. SOLUTION ANALYSIS

3.1 Human gestures imitations with Nao robot

By using the Kinect sensor human skeleton data is collected which consists of 25 3D position joints according to human camera coordinate as shown in Figure 19 and table 1.

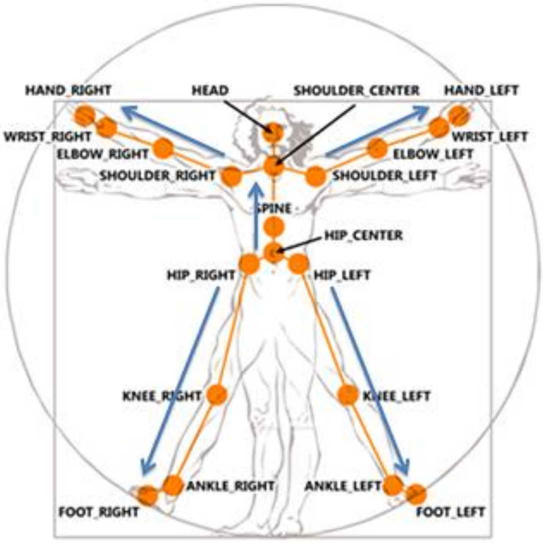


Figure 19. Joints data collected from Kinect sensor /12/

For this project all joints except neck, both feet and both hands are going to be used for human gestures imitation purpose. Since the Nao robot has fewer joints than a normal human only the necessary joints /13/ are included as shown in Figure 20.

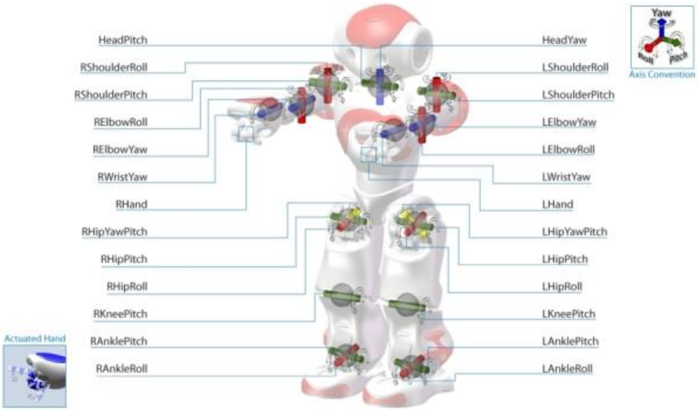
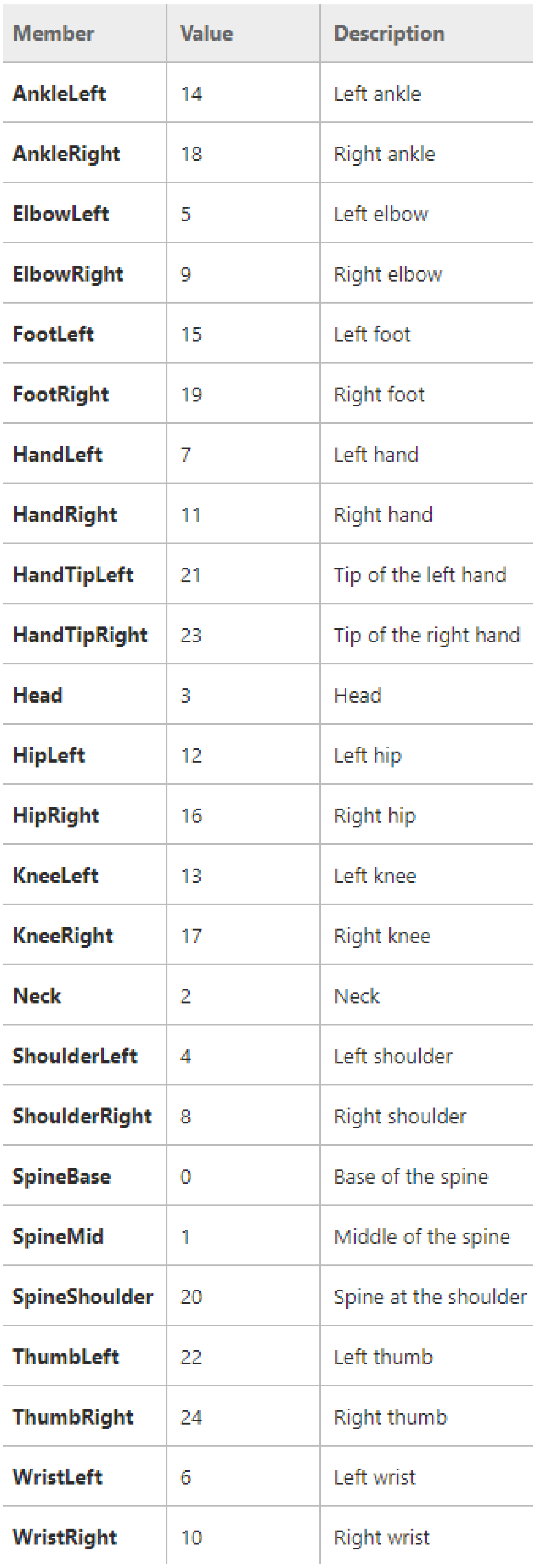


Figure 20. Joints data provided from Nao robot /13/

Table 1. Data collected from Kinect sensor /12/



Member	Value	Description
AnkleLeft	14	Left ankle
AnkleRight	18	Right ankle
ElbowLeft	5	Left elbow
ElbowRight	9	Right elbow
FootLeft	15	Left foot
FootRight	19	Right foot
HandLeft	7	Left hand
HandRight	11	Right hand
HandTipLeft	21	Tip of the left hand
HandTipRight	23	Tip of the right hand
Head	3	Head
HipLeft	12	Left hip
HipRight	16	Right hip
KneeLeft	13	Left knee
KneeRight	17	Right knee
Neck	2	Neck
ShoulderLeft	4	Left shoulder
ShoulderRight	8	Right shoulder
SpineBase	0	Base of the spine
SpineMid	1	Middle of the spine
SpineShoulder	20	Spine at the shoulder
ThumbLeft	22	Left thumb
ThumbRight	24	Right thumb
WristLeft	6	Left wrist
WristRight	10	Right wrist

```
def run(self):
    # -------- Main Program Loop -----------
    while not self._done:
        # --- Main event loop
        if self._kinect.has_new_color_frame():
            frame = self._kinect.get_last_color_frame()
            frame = None

        if self._kinect.has_new_body_frame():
            self._bodies = self._kinect.get_last_body_frame()

        # --- Collect Joints data to a global array
        if self._bodies is not None:
            for i in range(0, self._kinect.max_body_count):
                body = self._bodies.bodies[i]
                if body.is_tracked:
                    global skeleton_data
```

By using this joint data, seven angles for each arm of the robot can be calculated using the principle of cross and dot product. Each joint of the Nao robot has the range from -180 degree to 180 degree but the angle calculated from the operations is from 0 to 180 degree, so those angles which are not in the range are considered.

Table 2. Nao’s joint angles and ranges /13/

Joint name	**Motion**	**Range (degrees)**	**Range *(radians)***
HeadYaw	Head joint twist (Z)	-119.5 to 119.5	-2.0857 to 2.0857
HeadPitch	Head joint front and back (Y)	-38.5 to 29.5	-0.6720 to 0.5149
LShoulderPitch	Left shoulder joint front and back (Y)	-119.5 to 119.5	-2.0857 to 2.0857
LShoulderRoll	Left shoulder joint right and left (Z)	-18 to 76	-0.3142 to 1.3265
LElbowYaw	Left shoulder joint twist (X)	-119.5 to 119.5	-2.0857 to 2.0857
LElbowRoll	Left elbow joint (Z)	-88.5 to -2	-1.5446 to -0.0349
LWristYaw	Left wrist joint (X)	-104.5 to 104.5	-1.8238 to 1.8238
LHand	Left hand	Open and Close	Open and Close
RShoulderPitch	ight shoulder joint front and back (Y)	-119.5 to 119.5	-2.0857 to 2.0857

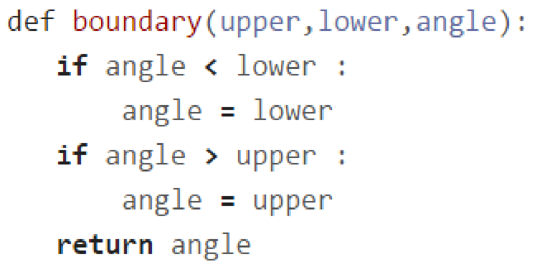
RShoulderRoll	Right shoulder joint right and left (Z)	-76 to 18	-1.3265 to 0.3142
RElbowYaw	Right shoulder joint twist (X)	-119.5 to 119.5	-2.0857 to 2.0857
RElbowYaw	Right shoulder joint twist (X)	-119.5 to 119.5	-2.0857 to 2.0857
RElbowRoll	Right elbow joint (Z)	2 to 88.5	0.0349 to 1.5446
RWristYaw	Right wrist joint (X)	-104.5 to 104.5	-1.8238 to 1.8238
RHand	Right hand	Open and Close	Open and Close
LHipRoll	Left hip joint right and left (X)	-21.74 to 45.29	-0.379472 to 0.790477
LHipPitch	Left hip joint front and back (Y)	-88.00 to 27.73	-1.535889 to 0.484090
LKneePitch	Left knee joint (Y)	-5.29 to 121.04	-0.092346 to 2.112528
LAnklePitch	Left ankle joint front and back (Y)	-68.15 to 52.86	-1.189516 to 0.922747
LAnkleRoll	Left ankle joint right and left (X)	-22.79 to 44.06	-0.397880 to 0.769001
RHipRoll	Right hip joint right and left (X)	-45.29 to 21.74	-0.790477 to 0.379472

RHipPitch	Right hip joint front and back (Y)	-88.00 to 27.73	-1.535889 to 0.484090
RKneePitch	Right knee joint (Y)	-5.90 to 121.47	-0.103083 to 2.120198
RAnklePitch	Right ankle joint front and back (Y)	-67.97 to 53.40	-1.186448 to 0.932056
RAnkleRoll	Right ankle joint right and left (X)	-44.06 to 22.80	-0.768992 to 0.397935

After the calculation, the captured motion data of each frame will be sent to the robot using NAOqi with a 50 percent of maximum speed and 100 millisecond delays.

3.1.1 Upper body control

All orientations can be captured and got using the vector operation mentioned above to transform from human coordinates to robot coordinates and be ranged before sending to the robot using a simple function:



```
def boundary(upper,lower,angle):
    if angle < lower :
        angle = lower
    if angle > upper :
        angle = upper
    return angle
```

The angle θ_1, left shoulder pitch is the supplement of the angle between vector $\overrightarrow{H\imath p\ Shoulder\ Left}$ (shoulder left, hip left) and vector $\overrightarrow{Left\ Upper\ Arm}$ (shoulder left, elbow left):

$$\theta_1 = \frac{\pi}{2} - \ cos^{-1}\ \frac{\overrightarrow{Left\ Upper\ Arm}\,.\overrightarrow{H\imath p\ Shoulder\ Left}}{\|\overrightarrow{H\imath p\ Shoulder\ Left}\| \times \|\ \overrightarrow{Left\ Upper\ Arm}\ \|} \qquad (8)$$

Angle θ_2, left shoulder roll is the angle between vector $\overrightarrow{Shoulder}$ (shoulder left, shoulder right) and vector $\overrightarrow{Left\ Upper\ Arm}$ (shoulder left, elbow left):

$$\theta_2 = cos^{-1}\ \frac{\overrightarrow{Left\ Upper\ Arm}\,.\overrightarrow{Shoulder}}{\|\overrightarrow{Shoulder}\| \times \|\ \overrightarrow{Left\ Upper\ Arm}\ \|} - \frac{\pi}{2} \qquad (9)$$

Left elbow roll, θ_3 is calculated using the angles between vector $\overrightarrow{Left\ lower\ arm}$ (elbow left, wrist left) and vector $\overrightarrow{Left\ Upper\ Arm}$ (shoulder left, elbow left):

$$\theta_3 = cos^{-1}\ \frac{\overrightarrow{Left\ Upper\ Arm}\,.\overrightarrow{Left\ Lower\ Arm}}{\|\overrightarrow{Left\ lower\ Arm}\| \times \|\ \overrightarrow{Left\ Upper\ Arm}\ \|} - \ \pi \qquad (10)$$

To calculate the left elbow yaw angle, θ_4, is the angle between body plane and left lower arm to create the body plane which contains two vector $\overrightarrow{H\imath p\ Shoulder\ Left}$ (shoulder left, hip left) and vector $\overrightarrow{Shoulder}$ (shoulder left, shoulder right), the normal vector of body plane is calculated using cross product equation :

$$\vec{P}_{body}\ =\ \overrightarrow{H\imath p\ Shoulder\ Left} \times \overrightarrow{Shoulder} \qquad (11)$$

$$\theta_4 = cos^{-1}\ \frac{\vec{P}_{body}\,.\overrightarrow{Let\ Lower\ Arm}}{\|\overrightarrow{Left\ lower\ Arm}\| \times \|\ \vec{P}_{body}\ \|} - \ \pi \qquad (12)$$

Finally, the left wrist angle, θ_5, is the supplement of the angle between body plane and vector $\overrightarrow{Wr\imath st\ Hand\ left}$ (Wrist left, Hand left):

$$\theta_4 = cos^{-1} \frac{\vec{P}_{body} . \overrightarrow{Wr\imath st\ Hand\ left}}{\left\|\overrightarrow{Wr\imath st\ Hand\ left}\right\| \times \left\| \vec{P}_{body} \right\|} - \pi/2 \qquad (13)$$

For the right hand, the same formula is implemented so that all the angles are calculated.

Left knee pitch, $\theta_5$is the angle of the $\overrightarrow{H\imath p\ Knee\ Left}$(knee left, hip left) and vector $\overrightarrow{Knee\ Ankle\ left}$ (ankle left, knee left). Left hip roll, $\theta_6$is the angle of the $\overrightarrow{H\imath p\ Knee\ Left}$(knee left, hip left) and vector $\overrightarrow{H\imath p\ Left\ R\imath ght}$ (hip left, Hip Right) and the last angle for the left foot is left hip pitch, which is the combination of hip knee left vector and hip shoulder left vector. After also providing the angles to right foot, all the desired angles for the Nao robot are ready to actuate. The thesis is heading to the biggest challenge of making the robot to keep its balance and having the support changes for certain gestures.

3.1.2 Double support with the whole body

Using the method for upper body control, the lower part of the body needs to be imitated using forward kinematics of human gestures. Also, the limits and balance need to be checked every time providing the joints angles to the Nao Robot.

This section of the thesis is considered to be a significantly important since the task is to control all the possible joints while keeping the center of mass of the robot to lie on the support polygon. The support change in real-time can be found in the next section. The stabilization of Nao while receiving noisy human gestures and some of the movements can be dangerous for Nao to imitate is analyzed.

The function used to transfer the joint angles of the robot is: proxy.angleInterpolationWithSpeed(names, angleLists, 0.2) which means 20% of the maximum speed for Nao Robot. The speed should be lower than normally to make the imitation safer and have more time to calculate. Also, the feet are fixed to the plane when the robot is on double support mode using the following function: proxy.wbFoot-State("Fixed", "Legs") and proxy.wbGoToBalance("Legs",0.5).

The balance of the Nao robot needs to be constrained using inversed kinematics to modify the desired joint angles, the Cartesian coordinates and also providing task priority jobs. We need to minimize the quadratic function of the error between the corrected Nao Joint angle - θ and the desired Nao joint angle - θ_d to be as small as possible with the quadric formula:

$$min\frac{1}{2}||\theta - \theta_d||^2 \tag{14}$$

And this can be transformed following /14/ as below with the aspect of time:

$$\min\frac{1}{2}(\Delta\theta - \Delta\theta_d)^T W(\Delta\theta - \Delta\theta_d) \tag{15}$$

With W is the weighting matrix.

In general, each frame got from Kinect and calculation have been achieved, 20 joints will be the desired angles for the Nao Robot. The COM of the Nao robot is calculated using getCOM() provided from the API to get the center of mass of the Nao robot during imitation. The center of mass must lie within the support polygon with keeps the balance of the robot. If the COM is out of the support polygon, Nao is going to lose its balance. A Jacobian matrix can present the relationship between the center of mass and the current angles which be obtained follow /14/. In each support leg mode, it is fixed to the plane or even when it becomes double-support so the center of mass is the center of the support polygon of the robot. The Jacobian is shown as below with the time aspect:

$$\Delta P_{COM} = J \times \Delta\theta_d \tag{16}$$

Following /14/, combining equations (1) and (2), the solution to a quadratic problem is solved:

$$minimize\ \frac{1}{2}(\Delta\theta - \Delta\theta_d)^T W(\Delta\theta - \Delta\theta_d)$$

$$subject\ to\ \Delta P_{COM} = J \times \Delta\theta_d$$

As a result, the linear system in (17) is the solution to the quadratic problem obtained above:

$$\begin{bmatrix} W & J^T \\ J & 0 \end{bmatrix} \times \begin{bmatrix} \Delta\theta_d \\ \varphi \end{bmatrix} = \begin{bmatrix} W \\ \Delta P_{COM} \end{bmatrix} \quad (17)$$

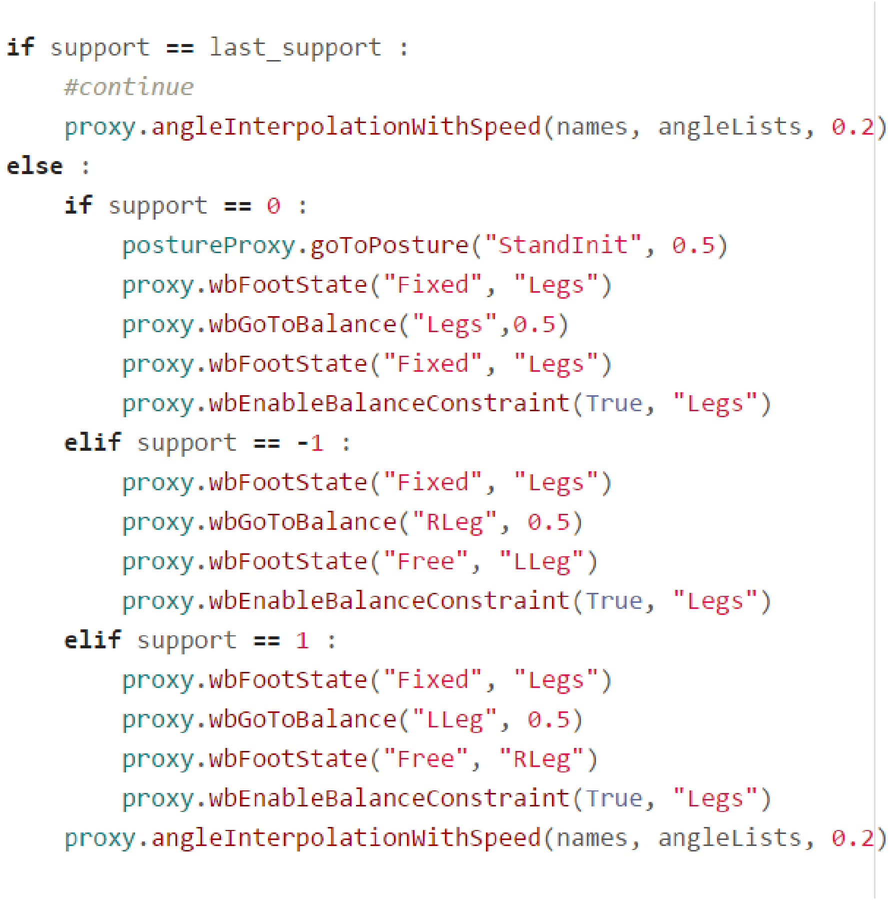
Where φ is a set of Lagrange multipliers of $\Delta\theta_d$ and after solving the mathematics problem, the result of $\Delta\theta_d$ provides us to modify the safe gestures to the Nao robot that avoid falling using equation (18):

$$\Delta\theta_d = \Delta\theta + W^{-1} \times J^T \times (J \times W^{-1} \times J^T)^{-1} \times (J \times \Delta\theta - \Delta P_{COM}) (18)$$

3.1.3 Single support with whole body

One of the objectives of this research thesis is to make the Nao robot be able to stand on one of its own feet. When the Nao robot tries to imitate the gestures with one foot, that foot needs to be fixed on the plane and free the other foot. Also, the center of mass is on support polygon to make sure there is no corruption issue.

After defining the support leg by fixing it on the floor, the other leg of the robot is free to actuate and then all the other joints are set using the following conditions:



```
if support == last_support :
    #continue
    proxy.angleInterpolationWithSpeed(names, angleLists, 0.2)
else :
    if support == 0 :
        postureProxy.goToPosture("StandInit", 0.5)
        proxy.wbFootState("Fixed", "Legs")
        proxy.wbGoToBalance("Legs",0.5)
        proxy.wbFootState("Fixed", "Legs")
        proxy.wbEnableBalanceConstraint(True, "Legs")
    elif support == -1 :
        proxy.wbFootState("Fixed", "Legs")
        proxy.wbGoToBalance("RLeg", 0.5)
        proxy.wbFootState("Free", "LLeg")
        proxy.wbEnableBalanceConstraint(True, "Legs")
    elif support == 1 :
        proxy.wbFootState("Fixed", "Legs")
        proxy.wbGoToBalance("LLeg", 0.5)
        proxy.wbFootState("Free", "RLeg")
        proxy.wbEnableBalanceConstraint(True, "Legs")
    proxy.angleInterpolationWithSpeed(names, angleLists, 0.2)
```

3.1.4 Support change method

The problem of the support change system is that the robot cannot move directly from the right leg support to the left leg support so a middle move needs to be set between the two. In this application, when Nao want to change the single support from the left leg to the right and vice versa, both legs will be fixed and after that the unsupported leg is free for imitation. After testing and choosing the pose for each leg support, the initial gestures are described as follows:

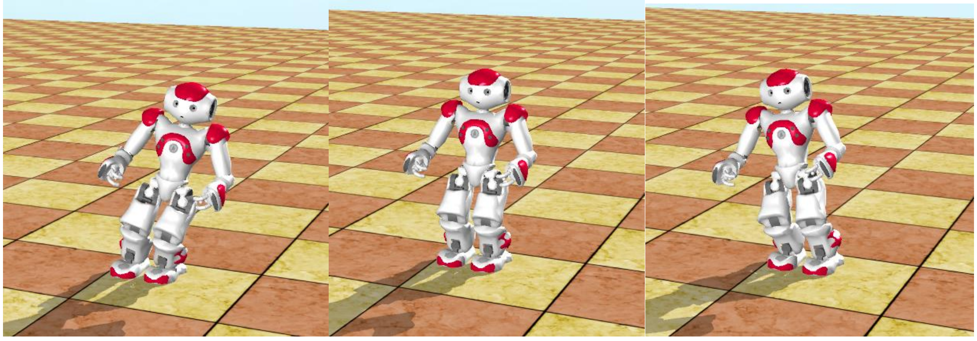
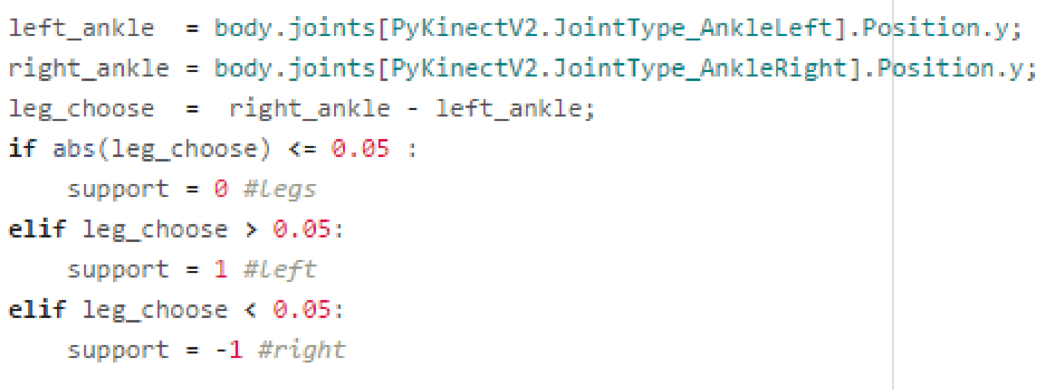


Figure 21. Left Leg – Right Leg – Both Legs initial pose

After having the one feet standing balancer, a system to make Nao robot change its support during imitation needs to be created. For this part, the position of each foot from the human skeleton was used to compare to each other, the difference is used to define the support for the Nao robot from the right to left leg to both. The threshold is selected to be 0.05m but with the noise of the Kinect sensor, a fixed threshold might not be a dynamic idea however in this case, it is safe to use. In the future studies, a learning process to obtain a range of threshold can be used, which would be useful for the robot to choose depending on the location between the camera and human position. Support selection is described as follows:



```
left_ankle  = body.joints[PyKinectV2.JointType_AnkleLeft].Position.y;
right_ankle = body.joints[PyKinectV2.JointType_AnkleRight].Position.y;
leg_choose  =  right_ankle - left_ankle;
if abs(leg_choose) <= 0.05 :
    support = 0 #Legs
elif leg_choose > 0.05:
    support = 1 #Left
elif leg_choose < 0.05:
    support = -1 #right
```

The model for supporting the change method can be described in the following state machine. The system will check the threshold every time when Nao receives a new desired angle to complete. If the value stays unchanged in a fixed range, support mode will be kept. But when Nao needs to change the support from one leg to another, a process of changing to the middle double support is executed as shown in Figure 22. The period for changing support takes about three seconds.

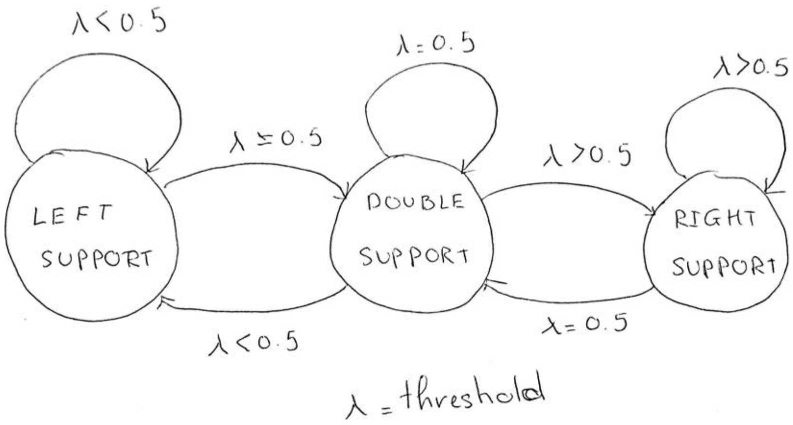


Figure 22. State Machine for application.

So finally, the combination between all the tasks can be simplified into the workflow of the main application to control Nao Robot with support change as shown in Figures 23.

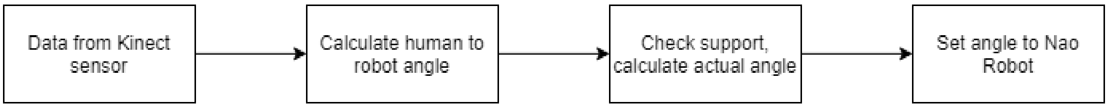

Data from Kinect sensor
Calculate human to robot angle
Check support, calculate actual angle
Set angle to Nao Robot

Figure 23. Application System.

3.2 Human movement classification

When taking the advantage of using the Kinect sensor to collect human joint data, one idea is to classify human gestures using a neural network. A system is built in a convolutional neural network. All the data is captured, trained and tested to see if the use of CNN can be implemented in human movement classifications.

3.2.1 Preprocessing Data

Human actions and images have the same construction when joints consist of three orthogonal planes X, Y, Z, and images have three channels R, G, B. Data of joints can be concatenated as an array of numbers with three channels and can be calculated and produced as an image. The following equation calculates the RGB channels of the image using skeleton data:

$$X = 255 * \left|\frac{P-P_{min}}{P_{max}P_{min}}\right| \tag{19}$$

While X is the value of corresponding red or green or blue channel, $P, P_{max}\ and\ P_{min}$ are the current coordinate values, the maximum value and the minimum value of Cartesian coordinate respectively. After applying the transformation function, the following image shown in Figure 24 is generated:

Figure 24. Image generated by joints data.

3.2.2 Network System

All the data is preprocessed with the transformation and then be transferred into the network system. The workflow of this convolutional neural network consists of forward-propagation, computing the cost and then backpropagation using AdamOptimizer to minimize the cost with a fixed learning rate.

In the forward propagation for the model, there are eight layers that data pass on through. They are the convolutional 2D layer, rectifier layer (can be considered as an activation function), Maxpool layer, Flatten layer and Fully connected layers. The number of neurons of the output of forward-propagation is equal to the number of classes, the percentage of predicted activity is classified and presented. In the next step, the output of forward-propagation is passed to the cost computation function to observe the difference between the corrected label and the predicted label because the main task is to keep the cost as minimum as possible. Finally, by using optimizers, all the parameters for our network should be configured to be able to minimize the cost of feeding to the forward propagation again /7/.

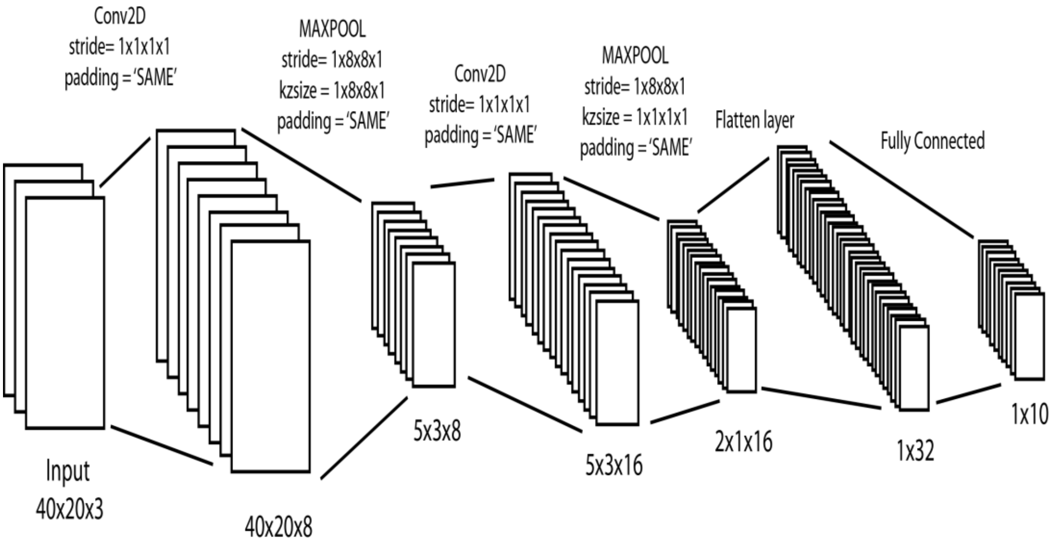
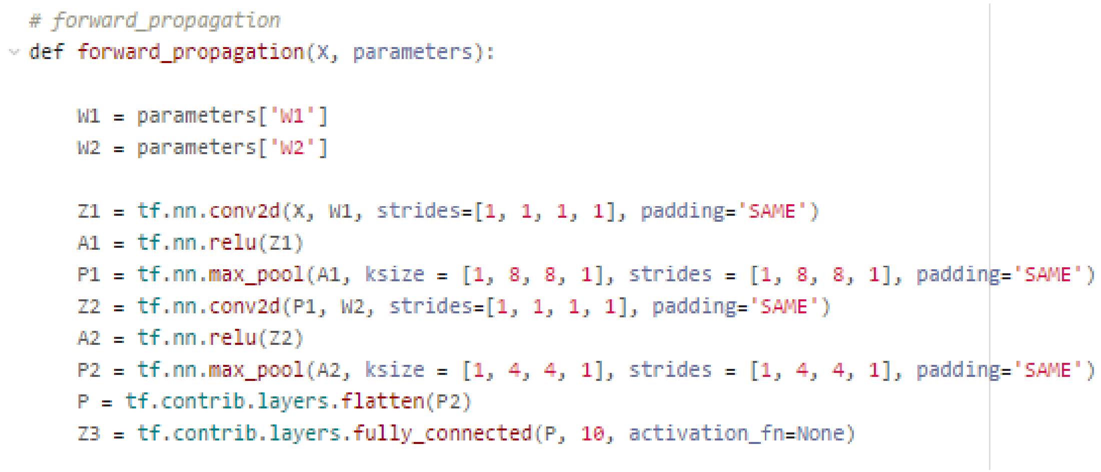


Figure 25. Network system.

The following function contains the propagation layers for forwarding steps where input data are transformed and resized:



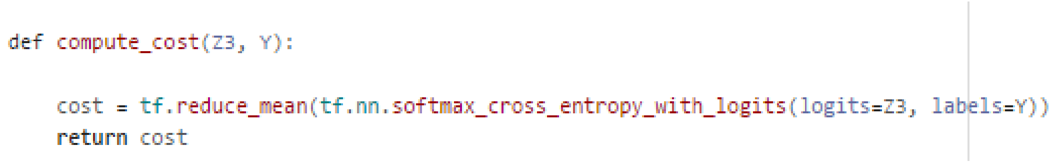
```
# forward_propagation
def forward_propagation(X, parameters):

    W1 = parameters['W1']
    W2 = parameters['W2']

    Z1 = tf.nn.conv2d(X, W1, strides=[1, 1, 1, 1], padding='SAME')
    A1 = tf.nn.relu(Z1)
    P1 = tf.nn.max_pool(A1, ksize = [1, 8, 8, 1], strides = [1, 8, 8, 1], padding='SAME')
    Z2 = tf.nn.conv2d(P1, W2, strides=[1, 1, 1, 1], padding='SAME')
    A2 = tf.nn.relu(Z2)
    P2 = tf.nn.max_pool(A2, ksize = [1, 4, 4, 1], strides = [1, 4, 4, 1], padding='SAME')
    P = tf.contrib.layers.flatten(P2)
    Z3 = tf.contrib.layers.fully_connected(P, 10, activation_fn=None)
```

After the forward propagation layer, the input data are transformed and have the following output size (note that the final output should have the size equal to the number of class in the network) as shown in table 3:

Table 3. Output data size

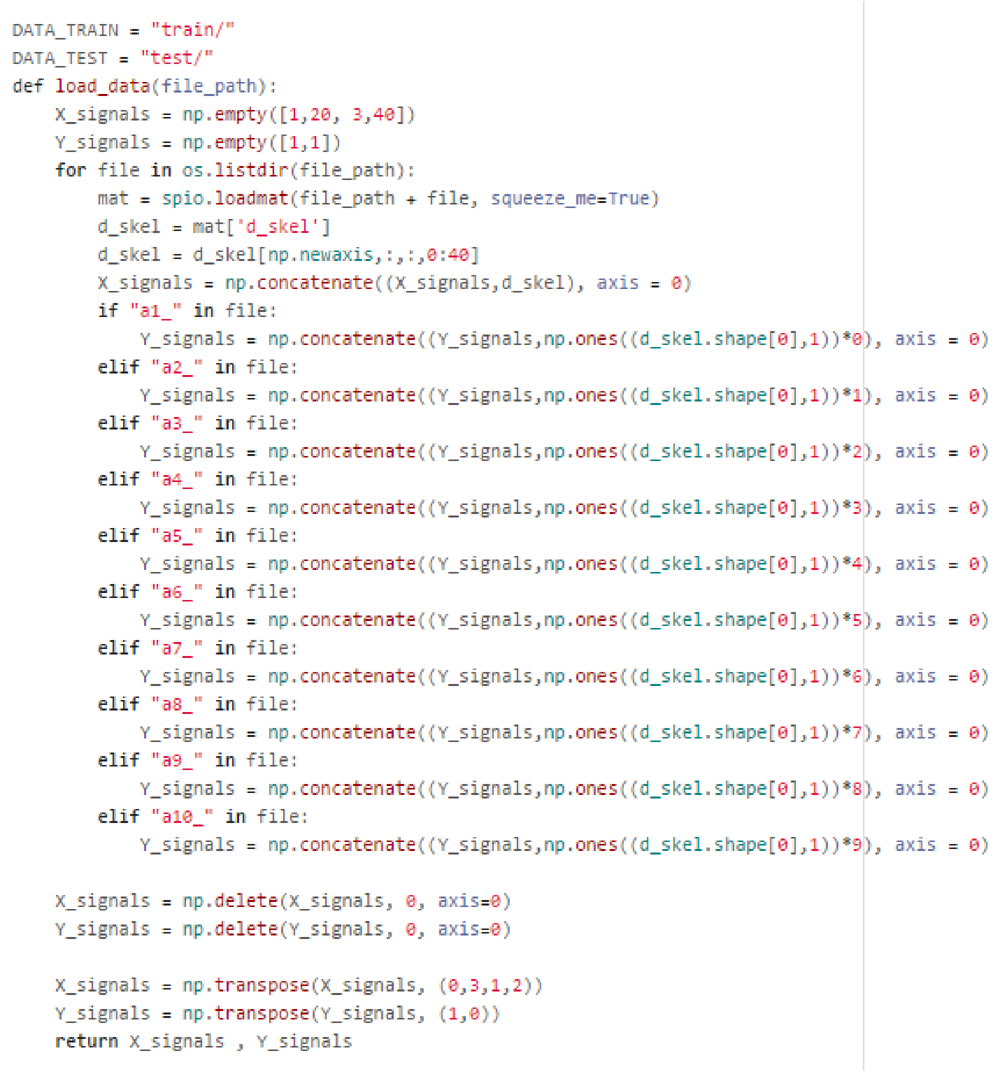
Layers	**Data size**
Conv2d	[?, 40, 20, 3]
Max_pool	[?, 40, 20, 8]
Conv2d	[?, 5, 3, 8]
Max_pool	[?, 5, 3, 16]
Flatten	[?, 1, 32]
Fully_connected	[?, 1, 10]

The cost function computes the difference between the predicted value and the actual value using softmax function and returns a cost for minimization. After each epoch, the cost is reduced to be as small as possible



```
def compute_cost(Z3, Y):

    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=Z3, labels=Y))
    return cost
```

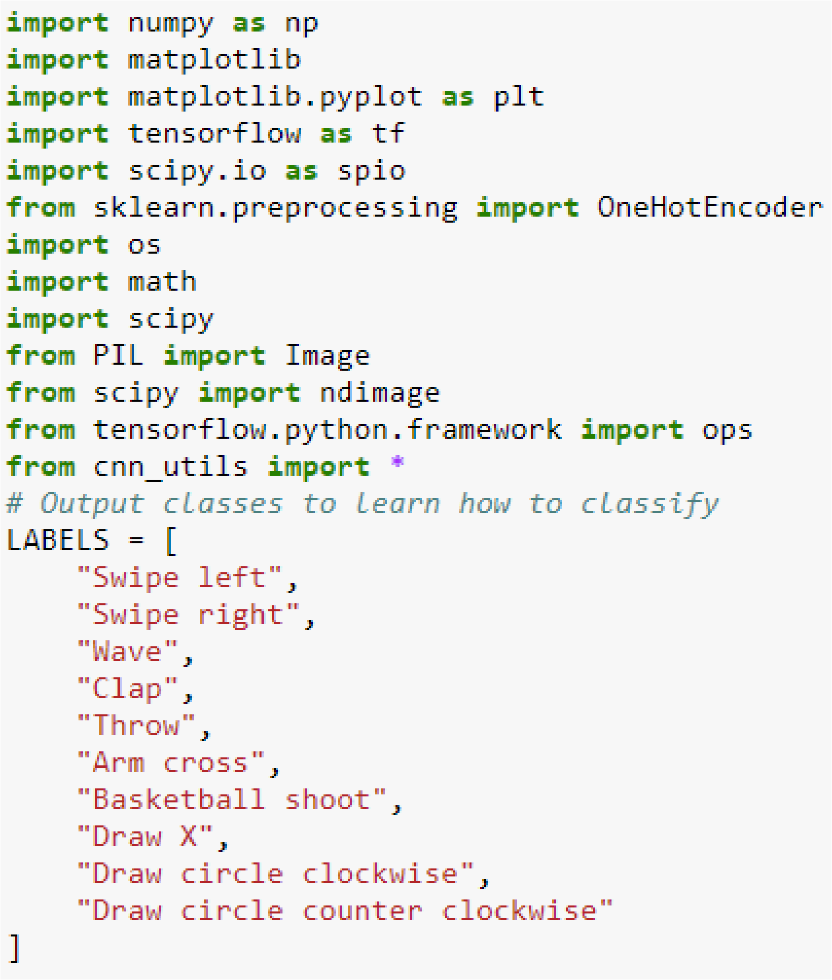
There are ten actions in Figure 28 that are classified using this network with 20 different collected joints from the Kinect sensor: swipe left, swipe right, wave, clap, throw, arms cross basketball shoot, draw “x” and draw a circle (clockwise and counterclockwise). The test data is 10% of the training data for the best performance but not either overfitting nor underfitting.



```
DATA_TRAIN = "train/"
DATA_TEST = "test/"
def load_data(file_path):
    X_signals = np.empty([1,20, 3,40])
    Y_signals = np.empty([1,1])
    for file in os.listdir(file_path):
        mat = spio.loadmat(file_path + file, squeeze_me=True)
        d_skel = mat['d_skel']
        d_skel = d_skel[np.newaxis,:,:,0:40]
        X_signals = np.concatenate((X_signals,d_skel), axis = 0)
        if "a1_" in file:
            Y_signals = np.concatenate((Y_signals,np.ones((d_skel.shape[0],1))*0), axis = 0)
        elif "a2_" in file:
            Y_signals = np.concatenate((Y_signals,np.ones((d_skel.shape[0],1))*1), axis = 0)
        elif "a3_" in file:
            Y_signals = np.concatenate((Y_signals,np.ones((d_skel.shape[0],1))*2), axis = 0)
        elif "a4_" in file:
            Y_signals = np.concatenate((Y_signals,np.ones((d_skel.shape[0],1))*3), axis = 0)
        elif "a5_" in file:
            Y_signals = np.concatenate((Y_signals,np.ones((d_skel.shape[0],1))*4), axis = 0)
        elif "a6_" in file:
            Y_signals = np.concatenate((Y_signals,np.ones((d_skel.shape[0],1))*5), axis = 0)
        elif "a7_" in file:
            Y_signals = np.concatenate((Y_signals,np.ones((d_skel.shape[0],1))*6), axis = 0)
        elif "a8_" in file:
            Y_signals = np.concatenate((Y_signals,np.ones((d_skel.shape[0],1))*7), axis = 0)
        elif "a9_" in file:
            Y_signals = np.concatenate((Y_signals,np.ones((d_skel.shape[0],1))*8), axis = 0)
        elif "a10_" in file:
            Y_signals = np.concatenate((Y_signals,np.ones((d_skel.shape[0],1))*9), axis = 0)

    X_signals = np.delete(X_signals, 0, axis=0)
    Y_signals = np.delete(Y_signals, 0, axis=0)

    X_signals = np.transpose(X_signals, (0,3,1,2))
    Y_signals = np.transpose(Y_signals, (1,0))
    return X_signals , Y_signals
```

There is a way to speed up the network without underfitting the result is to separate the data into small mini-batches. In the experiment result, the different batch numbers are applied to observe the performance of each situation. The epoch, number of single pass through the whole training set, is also applied to for us to be able to train the data by many iterations to take the best output of the classifier.

Finally, all the important libraries need to be preinstalled and imported before implementing the network:



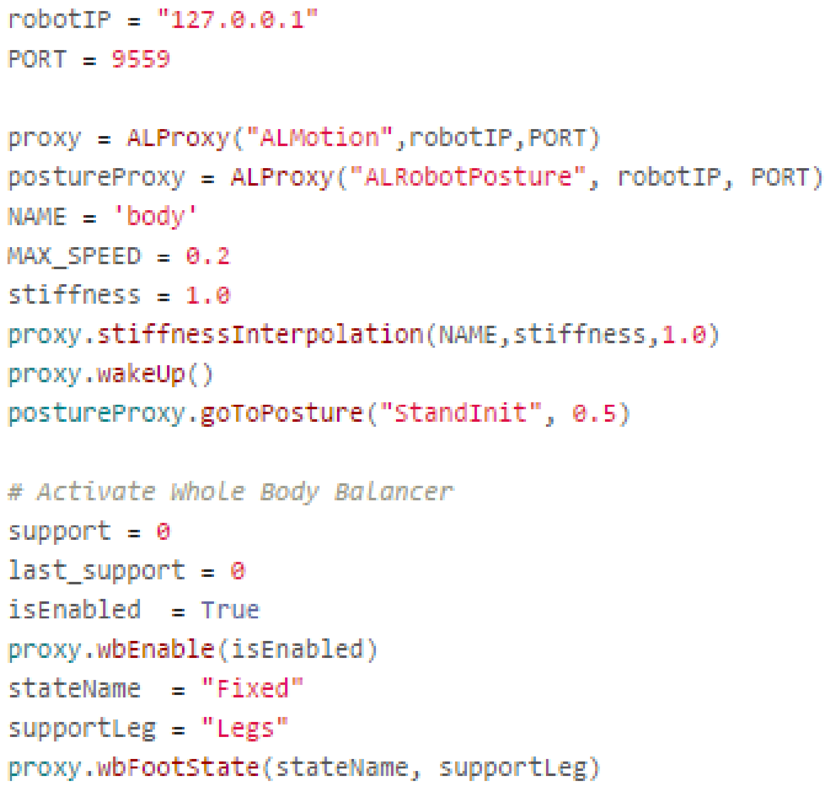
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import tensorflow as tf
import scipy.io as spio
from sklearn.preprocessing import OneHotEncoder
import os
import math
import scipy
from PIL import Image
from scipy import ndimage
from tensorflow.python.framework import ops
from cnn_utils import *
# Output classes to learn how to classify
LABELS = [
    "Swipe left",
    "Swipe right",
    "Wave",
    "Clap",
    "Throw",
    "Arm cross",
    "Basketball shoot",
    "Draw X",
    "Draw circle clockwise",
    "Draw circle counter clockwise"
]
```

4. EXPERIMENT RESULT

4.1 Whole body imitation with Nao robot

The whole experiment result was collected with the virtual robot using a Webots simulator. After connecting to the correct IP and Port, a Nao robot can imitate the motion with balance and without balance to see the effects of using the balancer in motion transferring. In practice, it takes about 10 seconds to start the first connection and 5 seconds for the next one. The distance between the human body and the Kinect sensor should be at least 2 meters so that Kinect can capture all the joints data needed.

When the imitation started, the Nao Robot took 5 seconds to capture the first movement. The maximum speed and the stiffness are configured in the setup on top of the program:



```
robotIP = "127.0.0.1"
PORT = 9559

proxy = ALProxy("ALMotion",robotIP,PORT)
postureProxy = ALProxy("ALRobotPosture", robotIP, PORT)
NAME = 'body'
MAX_SPEED = 0.2
stiffness = 1.0
proxy.stiffnessInterpolation(NAME,stiffness,1.0)
proxy.wakeUp()
postureProxy.goToPosture("StandInit", 0.5)

# Activate Whole Body Balancer
support = 0
last_support = 0
isEnabled  = True
proxy.wbEnable(isEnabled)
stateName  = "Fixed"
supportLeg = "Legs"
proxy.wbFootState(stateName, supportLeg)
```

In Figure 26, the imitation without balance support presents how the Nao robot behaves when the support is not set. After receiving the skeleton data and calculating the actual angles, the robot falls at the first frame. This issue may cause a significant problem when the real robot is brought into real testing.

As the result of virtual robot testing in Figure 27 and Figure 28 show, the gestures imitations are now successful with the support. When changing the support from the left foot to the right foot, an extra step of double support takes 5 seconds to complete.

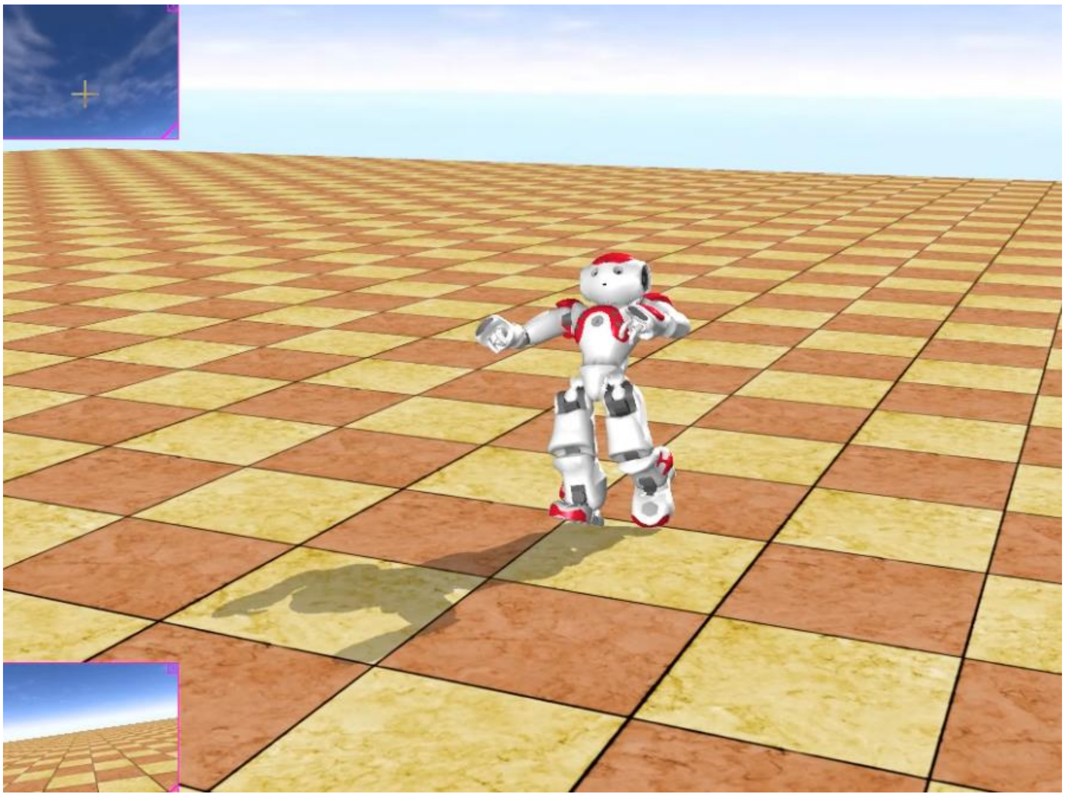
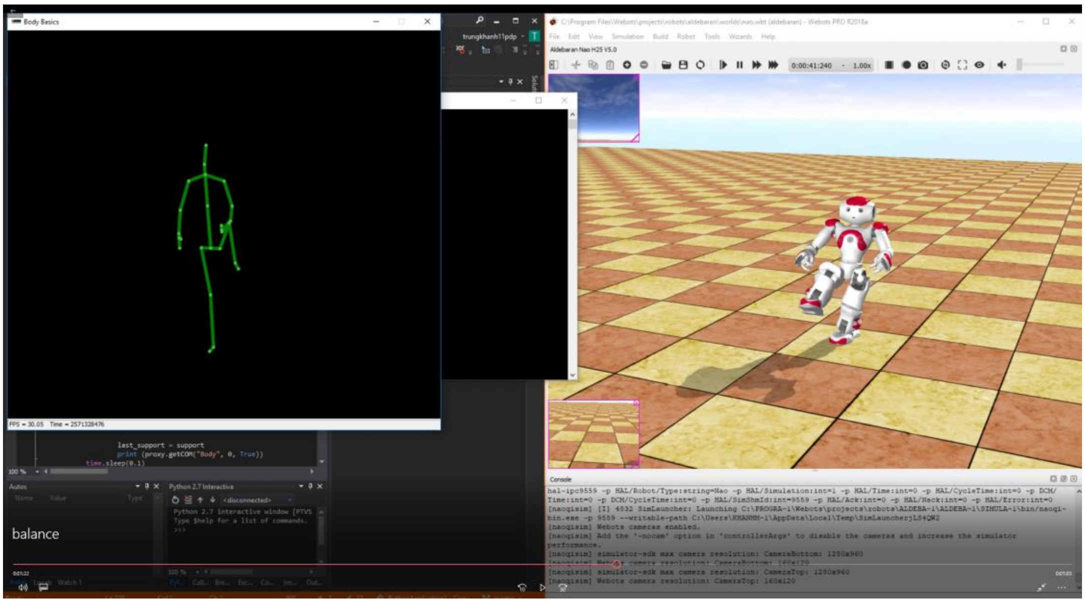


Figure 26. Nao robot's imitation without balance support.

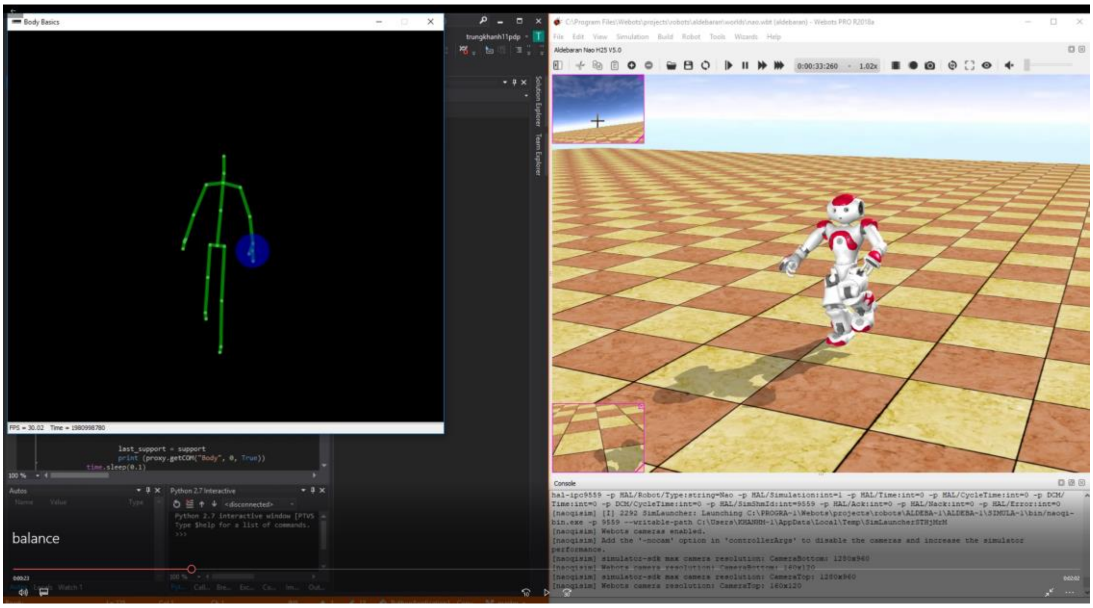

Body Basics
FPS = 30.05 Time = 2571328476
trungkhanh11pdp
C:\Program Files\Webots\projects\robots\aldebaran\worlds\nao.wbt (aldebaran) - Webots PRO R2018a
File Edit View Simulation Build Robot Tools Wizards Help
Aldebaran Nao H25 V5.0
0:00:41:240
1.00x
100 %
Autos
Name Value Type
Python 2.7 Interactive
<disconnected>
balance
Console

```
                last_support = support
                print (proxy.getCOM("Body", 0, True))
        time.sleep(0.1)
```

```
Python 2.7 interactive window [PTVS
Type $help for a list of commands.
>>>
```

```
hal-ipc9559 -p HAL/Robot/Type:string=Nao -p HAL/Simulation:int=1 -p HAL/Time:int=0 -p HAL/CycleTime:int=0 -p DCM/
Time:int=0 -p DCM/CycleTime:int=0 -p HAL/SimShmId:int=9559 -p HAL/Ack:int=0 -p HAL/Nack:int=0 -p HAL/Error:int=0
[naoqisim] [I] 4632 SimLauncher: Launching C:\PROGRA~1\Webots\projects\robots\ALDEBA~1\ALDEBA~1\SIMULA~1\bin/naoqi-
bin.exe -p 9559 --writable-path C:\Users\KHANHM~1\AppData\Local\Temp\SimLauncherjLS4QM2
[naoqisim] Webots cameras enabled.
[naoqisim] Add the '-nocam' option in 'controllerArgs' to disable the cameras and increase the simulator
performance.
[naoqisim] simulator-sdk max camera resolution: CameraBottom: 1280x960
[naoqisim] Webots camera resolution: CameraBottom: 160x120
[naoqisim] simulator-sdk max camera resolution: CameraTop: 1280x960
[naoqisim] Webots camera resolution: CameraTop: 160x120
```

Figure 27. Nao robot’s right leg imitation.

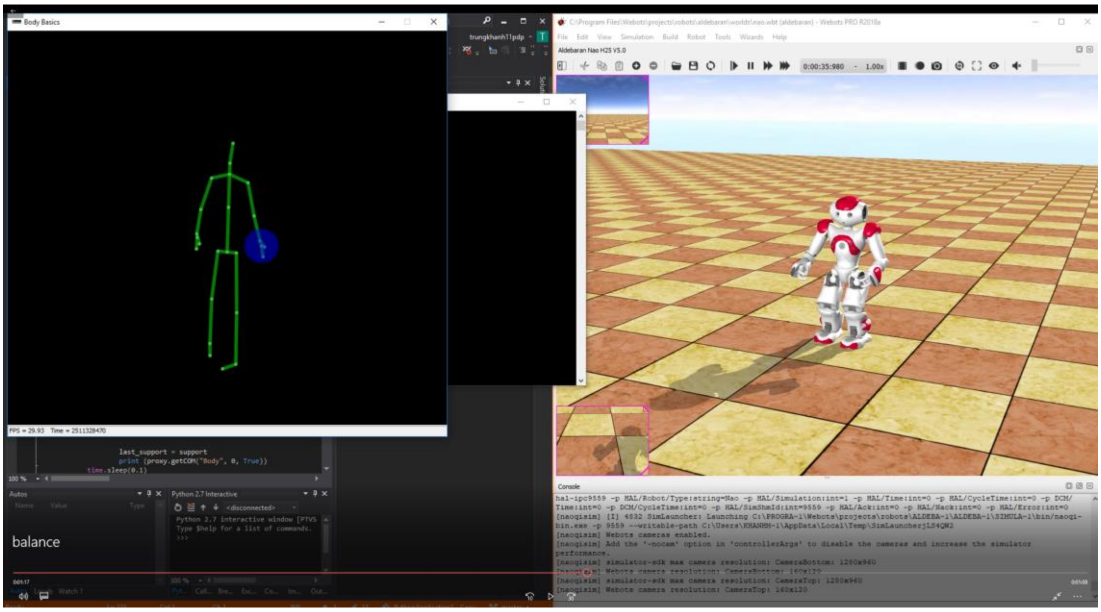

Body Basics
FPS = 30.02 Time = 1980998780
trungkhanh11pdp
Solution Explorer
Team Explorer
C:\Program Files\Webots\projects\robots\aldebaran\worlds\nao.wbt (aldebaran) - Webots PRO R2018a
File Edit View Simulation Build Robot Tools Wizards Help
Aldebaran Nao H25 V5.0
0:00:33:260
1.02x
100 %
Autos
Name Value Type
Python 2.7 Interactive
<disconnected>
balance
Console

```
                last_support = support
                print (proxy.getCOM("Body", 0, True))
        time.sleep(0.1)
```

```
Python 2.7 interactive window [PTVS
Type $help for a list of commands.
>>>
```

```
hal-ipc9559 -p HAL/Robot/Type:string=Nao -p HAL/Simulation:int=1 -p HAL/Time:int=0 -p HAL/CycleTime:int=0 -p DCM/
Time:int=0 -p DCM/CycleTime:int=0 -p HAL/SimShmId:int=9559 -p HAL/Ack:int=0 -p HAL/Nack:int=0 -p HAL/Error:int=0
[naoqisim] [I] 2292 SimLauncher: Launching C:\PROGRA~1\Webots\projects\robots\ALDEBA~1\ALDEBA~1\SIMULA~1\bin/naoqi-
bin.exe -p 9559 --writable-path C:\Users\KHANHM~1\AppData\Local\Temp\SimLauncherSTHjMrM
[naoqisim] Webots cameras enabled.
[naoqisim] Add the '-nocam' option in 'controllerArgs' to disable the cameras and increase the simulator
performance.
[naoqisim] simulator-sdk max camera resolution: CameraBottom: 1280x960
[naoqisim] Webots camera resolution: CameraBottom: 160x120
[naoqisim] simulator-sdk max camera resolution: CameraTop: 1280x960
[naoqisim] Webots camera resolution: CameraTop: 160x120
```

Figure 28. Nao robot’s left leg imitation.



```
                last_support = support
                print (proxy.getCOM("Body", 0, True))
            time.sleep(0.1)
```

```
Python 2.7 interactive window [PTVS
Type $help for a list of commands.
>>>
```

```
hal-ipc9559 -p HAL/Robot/Type:string=Nao -p HAL/Simulation:int=1 -p HAL/Time:int=0 -p HAL/CycleTime:int=0 -p DCM/
Time:int=0 -p DCM/CycleTime:int=0 -p HAL/SimShmId:int=9559 -p HAL/Ack:int=0 -p HAL/Nack:int=0 -p HAL/Error:int=0
[naoqisim] [I] 4832 SimLauncher: Launching C:\PROGRA~1\Webots\projects\robots\ALDEBA~1\ALDEBA~1\SIMULA~1\bin/naoqi-
bin.exe -p 9559 --writable-path C:\Users\KHANHM~1\AppData\Local\Temp\SimLauncherjLS4QW2
[naoqisim] Webots cameras enabled.
[naoqisim] Add the '-nocam' option in 'controllerArgs' to disable the cameras and increase the simulator
performance.
[naoqisim] simulator-sdk max camera resolution: CameraBottom: 1280x960
[naoqisim] Webots camera resolution: CameraBottom: 160x120
[naoqisim] simulator-sdk max camera resolution: CameraTop: 1280x960
[naoqisim] Webots camera resolution: CameraTop: 160x120
```

Figure 29. Nao robot's double support.

Although the imitation is not smooth and in real-time, the main scope of this is to make the Nao robot can stay in balance with both sing and double support using joints data which can be covered. The maximum speed of the robot can be increased despite of the fact that it is not recommended.

4.2 Human movement classification

In both of the network testing results, the training and testing accuracy are both 1.0 which means that is really good for this network. The CPU system I used for running this network is AMD Ryzen 3 1300X but if GPU is used as the main processor, the speed can be significantly faster. The result graphs contain the cost number during the training part and the predicted result compare to the actual result.

Firstly, the batch size is 16 and the number of the epoch is 200, the following graph shown in Figure 30 is the result of the training.

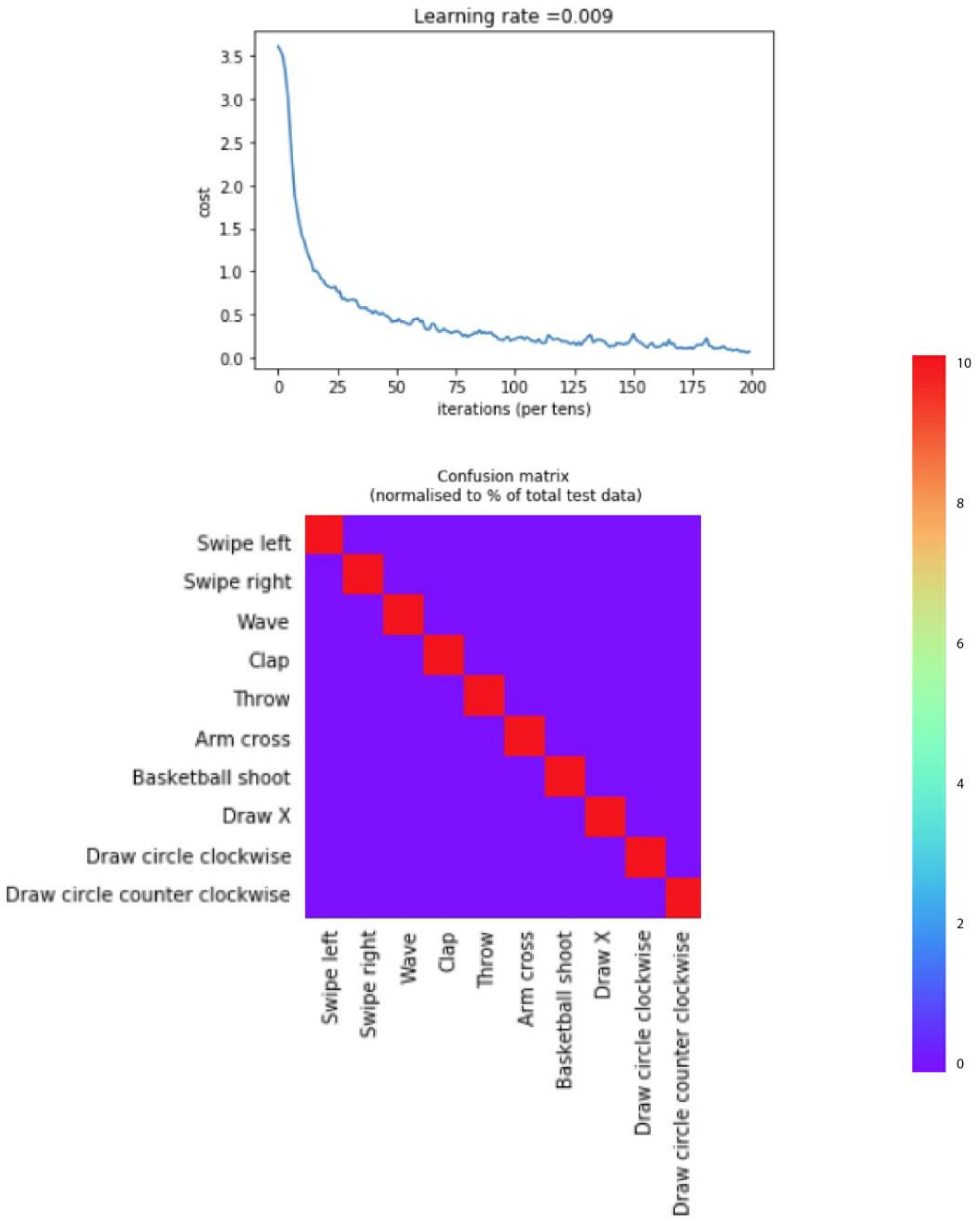


Figure 30. The first network training and testing.

After the first network run, the result in Figure 30 shows the number of the epoch can be smaller than 200 because the cost remains stable after a half of the training. In the next try, the hyperparameters are configurated: 100 as the epoch number and 8 as the batch size to see if the performance can be improved.

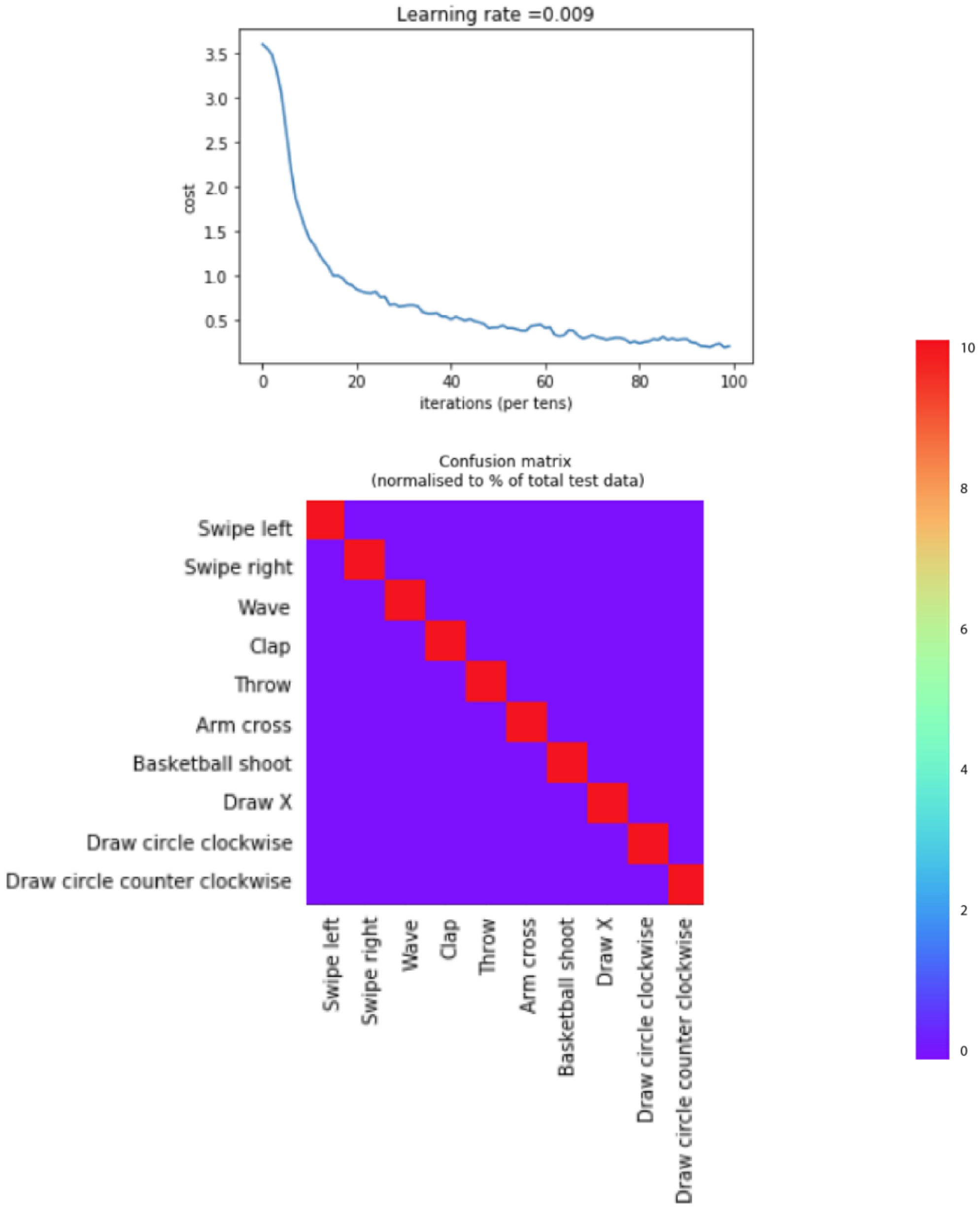


Figure 31. The second network training and testing.

In the second training in Figure 31, the network took only 3 minutes to train the data (less than 20% compared to the first training) and the best result for the output was still maintained. In both situations, the same learning rate, 0.009, id recommended and used for the network. All of the hyperparameters (epoch, batch size, learning rate, etc.) are free from tuning and configuring according to the network to improve the best performance.

5. CONCLUSION

The goal of the thesis consists of two areas: gestures imitation and classification using the Nao robot, the Kinect sensor and deep learning. The first target is to implement and transfer human movements to the Nao robot using a Kinect sensor, and the second part is classifying human actions using convolutional neural networks with a built system.

In the first target, using forward and inverse kinematics, the skeleton data collected from the Kinect sensor is transferred to the Nao robot using wireless transmission and a simulation software to prevent fall detection. The task faced many problems of changing leg support, transferring from the human angle joints to the robot angle joints and tracking stabilization. A lot of knowledge about mathematics and physics need to be archived beforehand to be able to create the solution for the task. The imitation simulation can be extended with supporting the Nao robot with more dynamics movements in the future. This approach can be used in the future for more developments on the Nao robot to become a personal assistant, after studying human behaviors and robot capability aspect.

The second task details the application using neural networks with human gestures for classifications and predictions. By using convolutional neural networks, the combination between image processing and sequence data processing was successfully obtained. The skeletons data from the Kinect sensor was transformed into RGB images for visualization, then fed to the network for training and testing. The biggest problem was to tune the hyperparameters for smallest training time and the best test result. Although this target has many limitations, it is still meaningful to apply neural network in different data processing methods.

After all the analysis and experiment results, all the suspects of the thesis were done correctly and had a good performance and the right technology and method were used to match the requirements. The biggest challenge in this work was to configure the parameters for the best result and learn the new technology widely used today. Although the lack of training data and testing data, the result in human movement classification can be improved by capturing more actions.

Because of the limitations of this thesis work, several improvements can be done in the future using the base work of human gestures imitation and classifications. For example, instead of using an image processing method, recurrent neural networks or hierarchical recurrent neural networks can be implemented for more dynamic performance and applications. The actions can be classified in real-time and output the prediction without taking certainly fixed frames. On the other hand, using more cameras make the networks better in classification whether machines can use multiple dimensions instead of one. With the imitations with the Nao robot implementation, programmers can consider making the robot perform actions without fixing at least one leg on the ground such as jumping or lying without losing its balance.

In the conclusion, this thesis can be a huge approach not only for future study but also for industrial implementations with human services and productions.

For any student who might be interested in robotics and artificial intelligence, this article contains the information of applying kinematics and deep learning on robots and carrying out for more ideas, researchs and future work.

REFERENCE

/1/ Nao robot from Softbank Robotics

https://www.ald.softbankrobotics.com/en/robots/nao

/2/ Kinect sensor by Microsoft

https://en.wikipedia.org/wiki/Kinect

/3/ Kinect SDK

https://www.microsoft.com/en-us/download/details.aspx?id=44561

/4/ Webots software

http://doc.aldebaran.com/1-14/software/webots/webots_index.html

/5/ Vectors documentation

https://www.intmath.com/vectors/7-vectors-in-3d-space.php

/6/ Centre of pressure and support polygon

https://www.researchgate.net/figure/Center-of-pressure-and-support-polygon_fig5_235676820

/7/ Convolutional neural networks

http://cs231n.github.io/convolutional-networks/

/8/ Convolutional neural network prameters

https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

/9/ Adam optimizer for deep learning

https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

/10/ Adam optimizer research

https://medium.com/@nishantnikhil/adam-optimizer-notes-ddac4fd7218

/11/ Tensorflow playground

https://playground.tensorflow.org/

/12/ Kinect sensor documentation

https://msdn.microsoft.com/en-us/library/jj131025.aspx

/13/ Nao robot version H2.5

http://doc.aldebaran.com/1-14/family/nao_h25/joints_h25.html

/14/ Quadric program

https://en.wikipedia.org/wiki/Quadratic_programming

/15/ Tensorflow

https://www.tensorflow.org/