Constructing a Ball-Grasping algorithm enhanced with Deep Convolutional Neural Network to control a Robotic Arm

Department of Computer Science
Brunel University London

Parth Patel

A dissertation submitted in partial fulfilment of the requirements for the degree of Bachelor of Science

"Intelligence is the ability to adapt to change."
-Stephen Hawking

Abstract

The problem that encapsulates my project is Object Recognition and Object Detection. Additionally, these two problems are combined with Robotics such that the Robotic Arm is to follow the movement of specific objects that are thrown to it and attempt to successfully catch the object using trained neural networks which process consistent frames through the use of a webcam.

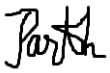
The project gives a starting point to allow the construction of systems that would be used to measure the performance of humans in various scenarios i.e. sports. One scenario is through the game of 'pool' where players can be tested against a gradually improving AI using Deep Learning to a point where a player can no longer win.

Furthermore, applications for this type of interception of objects with a robotic arm range from military to assistive operations in the home. The robotic arm can track and intercept moving objects therefore, becoming a valuable asset to ground troops by stopping grenades and other harmful projectiles. Furthermore, the ability to catch in-motion objects would be useful in the field of Socially Assistive Robotics (SAR) which focusses on assisting users through *social* rather than *physical* interaction [1]. The mission of SAR is to develop the computational techniques that will enable the design, implementation and evaluation of robots that stimulate social, emotional and cognitive growth. The use of robotic arms could encourage rehabilitated patients to lift objects as a method of training and perform tool handling exercises.

Acknowledgements

I would like to thank some people that helped me through writing this report and updating through feedback. First of all, I would like to express my sincere gratitude to my supervisor, Dr. Stasha Lauria, for reviewing and guiding me throughout the project. He was always available for quick feedback meetings and through other ways such as e-mail. I would further like to thank the Image-net database from which I was able to gather a variety of data needed to complete this project. Next, I thank my friends who encouraged me from start to end of this project and provided feedback on writing of this dissertation. Last but not least, I want to thank my family, especially parents for the continuous support and love. They helped me to build the robotic arm which took several days.

I certify that the work presented in the dissertation is my own unless referenced.

Signature 

Date _____05/04/2018_____

Total Words: 9570 (excluding References and Appendix)

Table of Contents

List of Tables

List of Figures

Nomenclature

Acronyms / Abbreviations

DNN	Deep Neural Network
CNN	Convolutional Neural Network
PASCAL	Pattern Analysis, Statistical Modelling and Computational Learning
VOC	Visual Object Classes
CAFFE	Convolutional Architecture for Fast Feature Embedding

1 Introduction

This project is based on Deep Learning, which is the sub-field of Machine Learning. In today's world, Deep Learning has taken a trend to solve many problems. The problem that encapsulates my project is Object Recognition and Object Detection with the use of a Robotic Arm such that the arm is to track the location of a ping-pong ball in 2-D space that is in a range specific to the limit of its reach. The arm will then attempt to successfully grasp the ball using a trained convolutional neural network which processes consistent frames using a web-cam.

My inspiration was drawn from a recent project which is currently being developed by 'The **Learning Algorithms** and **Systems Laboratory**' (LASA) at EPFL research institution. The researchers have programmed a robot capable of reacting on the spot and grasping objects with complex shapes and trajectories in less than five hundredths of a second [2]; more information to follow regarding their process in the Background section.

1.1 Aims and Objectives

Aim:

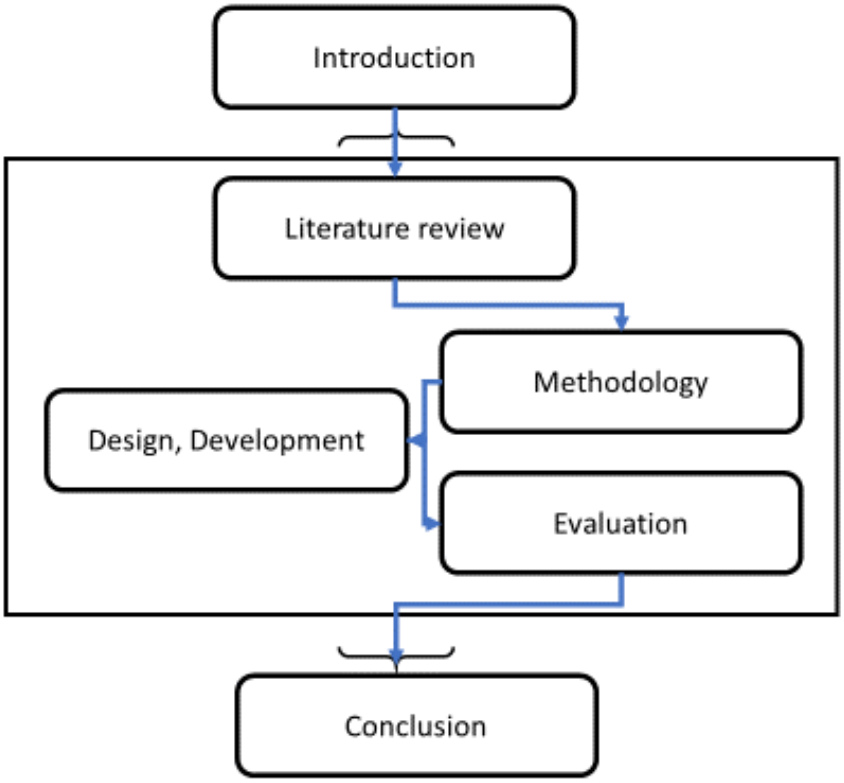
- To create a system that would train a neural network to classify, detect and track ping-pong ball(s) in order to grasp them using a robotic arm.

The aim of this project was to implement a system to grasp ping-pong balls with the use of web-cam and a robotic arm. This system would include a specifically created 'Ball Grasping Algorithm' to work in synchronously with the ball detection feature and several other features (mentioned later in this report) to grasp the ball effectively.

Objectives:

1. Gathering and creating a training and test dataset of ping-pong balls which are to be used for the neural network.
2. To train the neural network to classify and detect the ping-pong balls.
3. To process the neural network and display the detect results in real-time using a web-cam.
4. To track the ball(s) in its relative positions in 2-D space.
5. Finding the maximum movement time and angle of each joint of the arm through trial and error.
6. Calculating the angle between the ball and the robotic arm and to move *'x'* angle.
7. To implement an algorithm to get the final/current position of the arm and returning it to the original position.

1.1 Dissertation Outline



2 Background

The problem of grasping objects in flight or on a planar surface by predicting the object's trajectory and intercepting it with robotic arm has been researched deeply in the fields of Robotics, Machine Learning and Computer Vision. Currently, the problem of grasping through Deep Learning is being focussed at an in-depth level which would allow the robotic arm to grasp any type of object by learning the optimum grasping area for each object and maximising the chance of successfully grasping it. This project involves Deep Learning to learn the features of ping-pong balls and detect them using a processing script. Furthermore, the script would be directly linked to the robotic arm however, due to the robotic arm not containing any sensors and the arm being slow in movement, it is difficult to apply the Deep Learning grasping problem. Therefore, this project involves the implementation of a system that uses a trained model on ping-pong balls and calculates the angle between the ball and the robotic arm (with the support of the OpenCV library). This allows the robotic arm to move the specified angle to grasp the ball. As there have been several attempts in the past to catch moving and stationary objects, I shall briefly provide the key and most common techniques used in those attempts.

2.1 Deep Convolutional Neural Network

Deep Learning is a sub-field of machine learning which uses highly effective network architecture to learn specific features of objects. Convolutional neural networks are a special kind of neural network for processing data that contains a grid-like topology, e.g. image data, which can be interpreted as a 2-D grid of pixels. CNN's have achieved state-of-the-art performance in various applications such as image segmentation, object detection, image recognition etc. [34].

The inspiration of convolutional networks comes from biological processes in which the connectivity relationship between neurons is inspired by the anatomy structure of the animal visual cortex. Convolutional neural networks are very similar to ordinary neural networks and are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product operation and optionally follows it with a non-linearity activation function. The entire neural network still represents a single differentiable function, i.e., from the raw input image pixels on one end to class scores at the other. For example, when processing an image, the input image might have thousands or millions of pixels, but we can still extract meaningful features, e.g., edges features, with kernels that only occupy a small and restricted area [34].

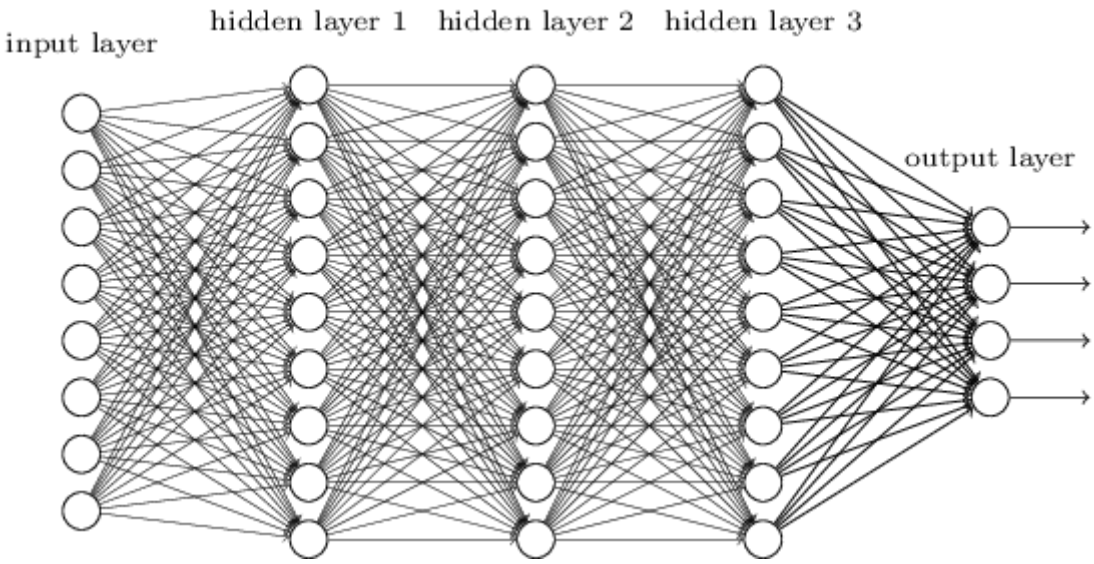


Figure 1: Example of fully connected neural networks.

The network shown in Figure 1, shows adjacent networks that are fully connected to one another. This involves the features learnt in each layer to be passed on until it reaches the output layer which then stores the classification of the object. In addition to convolutional layer, CNN also contain **pooling layers**. The task for the pooling layer is to simplify the information in the output from the convolutional layer. The overall goal of the architecture is to use the training data to train the network's weights and biases so that the network does a good job classifying input images/data [33].

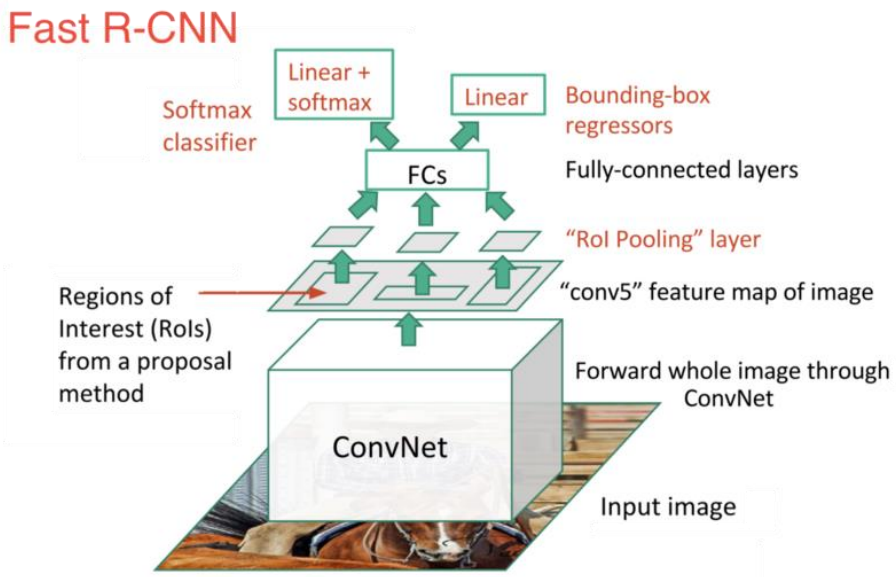


Figure 2: A type of CNN known as Fast-R-CNN [23].

Figure 2 illustrates how a CNN performs its tasks in different layers to output the classification in the image using the '**softmax'** classifier.

2.2 Research

A group of researchers at LASA [3], devoted their work to the autonomous control of fast movements such as catching moving objects at a quick rate. The researchers were inspired by how humans themselves learn by imitation and trial and error. This technique is known as, programming by demonstration, which shows examples of possible trajectories to catch the object. The researchers would guide the robotic arm to the projected target and repeat this process several times in order for the robot to learn. To catch objects which are statically unbalanced such as a hammer and dynamically unbalanced such as a half-full water bottle, the robot creates a model of the object through the cameras whilst the object is in flight. A motion capture system, an OptiTrack system by NaturalPoint, with the use of reflective markers stuck to the objects help the camera to model the object in mid-air [4].

Although their research doesn't consist of Deep Learning as they use sensory equipment to track the movement and rotation of the object. During the training phase, they would use a human entity to move the robotic arm to the thrown object's trajectory. The angular joint movement of the arm would be calculated in this process and hence, the joint movement values would be used for training. The overview of their full framework is shown in Figure 3.

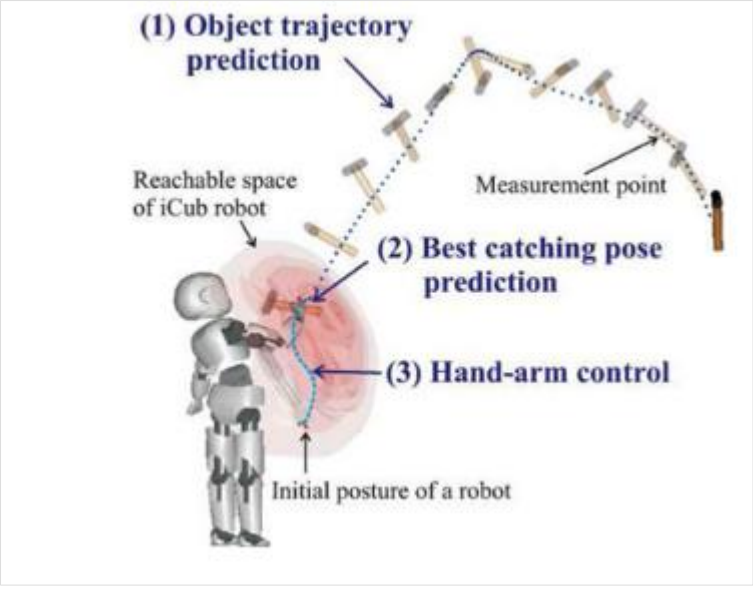


Figure 3: A schematic overview of the system developed by LASA.

The problem definition for this project consists of grasping specific objects (in this instance, a ping-pong ball) that are in the graspable area of the robotic arm. The main aim was to predict the trajectory of the moving ball and grasp it however, due to some technical reasons regarding inefficient robotic arm functions, the aim was directed towards a stationary ball.

Applications for this type of interception of objects with a robotic arm range from military to assistive operations in the home. The robotic arm can track and intercept moving objects therefore, becoming a valuable asset to ground troops by stopping grenades and other harmful projectiles. Furthermore, the ability to catch in-motion objects would be useful in the field of Socially Assistive Robotics (SAR) which focusses on assisting users through *social* rather than *physical* interaction [5]. The mission of SAR is to develop the computational techniques that will enable the design, implementation and evaluation of robots that stimulate social, emotional and cognitive growth. The use of robotic arms could encourage rehabilitated patients to lift objects as a method of training and perform tool handling exercises.

A similar topic in the field of Deep Learning was utilised by Google who want to solve robotic grasping by letting the robotic arms learn by themselves. In a normal human scenario, it is possible to pick up objects without using eyes however, a link between eyes and an arm is much better. In robotics, this is known as **Visual Servoing**, and in addition to improving grasping accuracy, it makes grasping possible when objects are moving around or changing orientation during the grasping process. A team of researchers at Google Research, tasked a 7-DoF robot arm with grasping objects in clutter using monocular visual servoing and used a deep CNN to predict the outcome of the grasp. The CNN would continuously re-train itself where the start contained a lot of fail however, it gradually became better. Google increased the training time by using 14 robots at this problem in parallel. The method for Google's Research consists of two components: a grasp success predictor, which uses a deep convolutional neural network to determine how likely a given motion is to produce a successful grasp, and a continuous servoing mechanism that uses the CNN to constantly update the robot's motor commands [7]. The result of their approach led to intelligent reactive behaviours after over 800,000 grasp attempts which is equivalent to approximately 3,000 robot-hours of practice. The robot would observe its own gripper and correct its motions in real-time. Furthermore, it would separate certain objects to get a better grasp at the object [38].

[35] shows a system built by 'Shadow Robotics' that allows one to create and test robot programs through simulations. It also allows to make a robot to grasp something without having to learn everything related to Machine Learning and being available on the ROS Development Studio. Lastly, the video provided on their site shows how it is possible to grasp a ball by adjusting the position of the robot arm.

3 Methodology

In this section, I shall discuss and compare the available methodologies that are supported for this project. Secondly, I describe the selected methodology and explain the reasons on how it may affect the project in terms of improvement. Finally, I investigate the effects of using the testing methodology and how it was able to balance the system to work synchronously.

3.1 Comparison of Methodologies

The research in this particular field of Artificial Intelligence is expanding at an enormous rate hence, the popular methodologies such as Waterfall would not be considered as appropriate. The optimal approach that is followed throughout my project is the Agile methodology. A change of requirements is a key factor in this project as opposed to the Waterfall methodology, which should only be used when requirements are clearly understood and are unlikely to change during system development. Furthermore, this project involves recurrent background research which potentially disregards this model as new factors (restrictions/challenges) in the project start to arise.

Additional methodologies were researched to meet the requirements for this project and the most suitable found was in the form of an iterative process. This would involve several builds to be completed after each stage which include rigorous testing and bug fixing. More variants of Agile methodology were explored such as Scrum and Extreme Programming (XP) however, Scrum requires working closely with a team and "delivering potentially shippable increments of software during successive Sprints, typically lasting 30 days. Once a Sprint has been delivered, the Product Backlog is analysed and re-prioritised, if necessary, and the next set of functionality is selected for the next Sprint" [5]. The similar scenario is with the XP variant which "promotes high customer involvement, rapid feedback loops, continuous testing, continuous planning and close teamwork to deliver working software at very frequent intervals" [5]. Although, XP might seem a viable approach due to having an iterative-by-iteration testing, the customer works closely with the development team. Hence, these approaches were discarded due to the project not having a specific user group to coordinate with throughout the entirety of this project.

3.2 Selected Methodology

The variant of Agile approach known as FDD (Feature-Driven Development) originally developed and articulated by Jeff De Luca, would be most suitable for this project. Some of the best practices for FDD are 'Developing by Feature', 'Inspections', 'Configuration Management',

'Regular Builds' and 'Visibility of progress and results'. These practices directly relate to this project and the only ethics required would be at the completion of the project.

Finally, one of the most important project trait would be 'Feature Prioritisation' which is related to the FDD approach. "Prioritisation by value ensures the most valuable features are implemented first, thus reducing the risk of having an unusable product once funding runs out. This approach decreases risk of complete failure by allowing "partial" success" [6].

3.3 Testing Methodology

The testing approach that was utilised for this project is to consider the main 'Aim' of the project and choose a scope. In this case, the scope was to test the amount of times the arm successfully catches the ball out of 100 times. The reason why the value '100' is chosen is because of the time it takes to constantly run the project and to consider the battery life of the robot arm. For a GPU, it would be faster in terms of detection however, the robot arm would still follow its slow speed. Therefore, it was essential to select a suitable testing range to gather results.

As mentioned in the selected methodology, the FDD agile approach would be suitable for this project. So, each main 'features' of the project were tested to synchronously work with other features including the robot arm. For e.g. if the ball detection feature was tested to find that the ball would not be detected at certain distances, the threshold value would be changed to fit that requirement. This would allow for a better detection based on the accuracy of the model. For the ball tracking method, FPS rate was highly beneficial however, due to technical reasons, GPU could not be used for this project. A CPU would perform slow tracking as the model is being processed each frame to ensure the object is not missed. To get higher FPS rate in CPU, a method called '**Threading**' was attempted however, it did not provide the rates needed for a faster object tracking method. The angle calculation feature was tested using OpenCV as it was essential to find correct angles in the testing phase. There were certain scenarios where the angle was not accurate, so the angle formula was changed to improve that. Additionally, the angle of the web-cam helped to solve this issue. Due to the angle using 'tan' quadrants, it would give negative angles. To fix this, a method was developed for using absolute values for angles.

Lastly, the whole system was tested after all the features were successfully in sync. The last testing stage was to adjust the environment lighting to get the best detection possible which would allow for the whole system to work effectively.

4 System Design

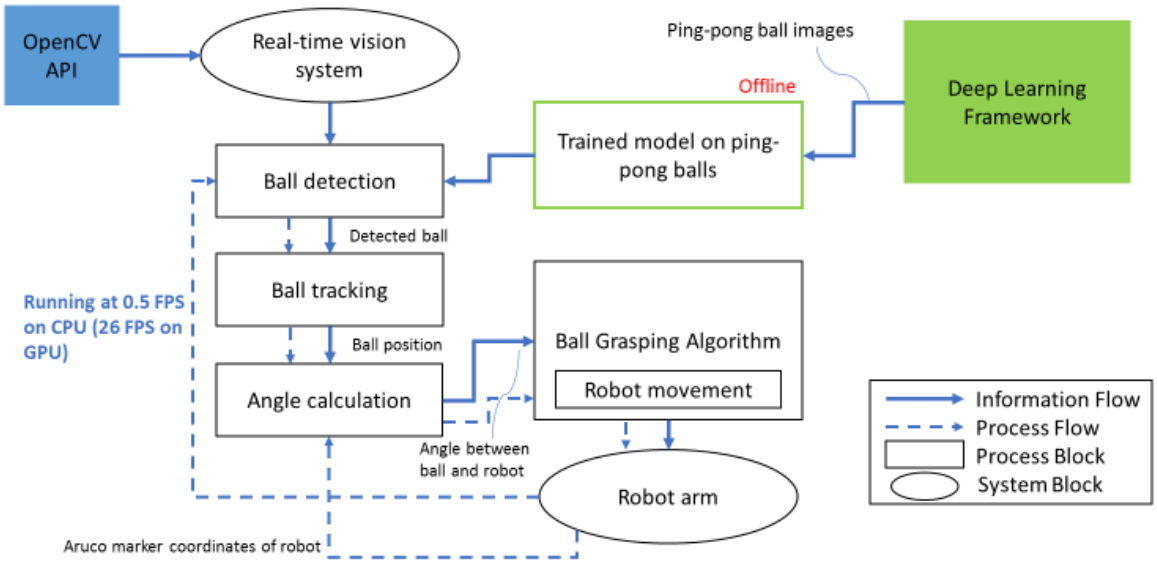


Figure 4: Block diagram for robot grasping system.

Designing the system was a tough task for this project as the frameworks had not been decided and there was no clear destination of the end goal of the project. Deep Learning is a field with many uncertainties which makes it difficult to predict the outcome of the system. Figure 4 illustrates the entire system flow in term of blocks which is known as a '**Block diagram**'. Each of the process blocks represent a feature of the system which have specific tasks to complete. Additionally, there are 'Offline' process blocks which are tasks that were done beforehand such as training a deep neural network. System blocks play a major role as without them, the project would not commence. The trained model is then used for ball detection using the web-cam. The information is flowed throughout the system between each process hence, the ball is tracked after detection. Next, the robot arm would be constantly updating the Aruco marker attached to it and sending the coordinates of it to the angle calculation block. Once the angle is calculated, it is passed onto the 'Ball Grasping Algorithm' to start the movement and grasp the ball at the given angle. This process is iterated on a loop and so, the system will try to detect a ball in the environment at every stage. Finally, the robot arm will stay idle until a **stationary ball** is detected, and the coordinates are steady for the angle to be precise.

5 Implementation

The immense work that was implemented in this project range from new Deep Learning methods to novel algorithm for controlling a robotic arm. The below sub-sections are the building blocks for this project in the order of the implementation. The main objective of this section is to portray the 'lifeline' of this project from start to end. The section commences with gathering the training and test dataset needed for the neural network. This is followed by training different DNNs (Deep Neural Network) and comparing the FPS rates. Lastly, the Ball Grasping algorithm is explored featuring the robotic arm which is scripted in Python (programming language).

5.1 Dataset Creation

This stage of the project consisted of finding the dataset that would be used to train the neural network. This is a crucial process and time consuming due to the amount of data needed for certain networks to generate a decent accuracy. Object Detection requires a dataset with a specific folder structure which varies according to the architecture. The DetectNet architecture by NVIDIA, uses a structure as shown in Figure 5 [10], where each image inside the image folder must have an associated text file in the label folder.

Figure 5: The input folder structure for DetectNet consisting of images and labels.

The label (text) file is encoded with 15 columns which represent certain values that are needed for the object to be detected at a higher rate. The label (as show in Figure 6) is of 'Tennis Ball' as this was the first category to be used for training purposes. The second label, 'DontCare', denotes regions of the images that would be ignored whilst the network is being trained. The 'DontCare' label can also be used as the negative images of the dataset. For example, in Object Classification, it is usually the case where there are two separate folders, 'positive' for images that are considered interesting (or the object to be classified) and 'negative' for images that are not to be classified. This creates a balance between the two classes and avoids '**Overfitting**'.

Additionally, the 'DontCare' label is used in this manner to prevent objects being counted as false positives and hence, the network will ignore these specified regions. For more information, please visit [11].

```
TennisBall 0.00 0 -1.59 523.41 194.53 550.65 219.39 1.44 1.66 4.53 -4.82 2.90 47.02 -1.69
TennisBall 0.00 0 -1.68 572.15 185.81 606.45 219.24 2.02 1.70 4.60 -1.42 2.89 47.11 -1.71
TennisBall 0.00 2 1.27 387.26 166.06 473.51 213.36 3.46 2.57 14.66 -15.22 3.07 59.75 1.02
DontCare -1 -1 -10 176.23 181.27 229.40 218.81 -1 -1 -1 -1000 -1000 -1000 -10
```

Figure 6: The bounding box format required by DetectNet.

There have been several problems regarding the dataset such as the size and format. The size of the dataset is a key aspect when training a Deep Neural Network where the amount of data required for machine learning relies on many factors, such as [8]:

- **The complexity of the problem** – nominally the unknown underlying function that best relates your input variables to the output variable.
- **The complexity of the learning algorithm** – nominally the algorithm used to inductively learn the unknown underlying mapping function from specific examples.

The 'Tennis Ball' category was chosen because of the spherical shape which can be grasped from any angle with ease. Initially, the dataset was gathered from the ILSVRC 2012+ (Imagenet Large Scale Visual Recognition Challenge) database. Approximately **1162 images** were available for this category however, the associated label files which contains the bounding box locations of the object(s) inside the images were of 'xml' format. This resulted in a dataset conversion from XML to KITTI format. The KITTI [9] format is used by the DetectNet framework which is shown in Figure 6.

5.1.1 PASCAL VOC to KITTI Dataset Conversion

The Imagenet Dataset [12] gathered contained a PASCAL VOC [13] format which had specific XML tags in an order. KITTI data used a text file format with a different layout therefore, a Python script was created to get the main XML tags (bounding box coordinates and the label) and create text files in the KITTI format. The rest of the values are not necessary for KITTI as they can be null or '0.0'.

5.1.2 Tensorflow Dataset Conversion

Tensorflow's Object Detection API, which was released to the public in June 2017, offers one of the best performance and usability in the Machine Learning field. For more information on Tensorflow, please go to section 5.4.2. An adjustment had to be made during this stage of the project, which was to use 'Ping-Pong Ball' as the category instead of the 'Tennis Ball'. This was due to the robotic arm's gripper could not withstand the size of the tennis ball hence, unable

to grasp it at a later stage. A new dataset size of 477 images was created with their corresponding label files; 377 images from the ILSVRC dataset and 100 manually labelled images by using "LabelImg" [19], a label tool, which outputs XML files with the bounding box coordinates. The dataset was split between training and validation images on a 70:30 ratio respectively. The validation images were used to validate the model's performance against unknown images.

Tensorflow's Object Detection API required annotation files in the XML format however, some label files from a certain year had to be tweaked due to inconsistency. The ILSVRC dataset had mixed images from the years, 2012, 2013 and 2014. The problem was regarding the bounding box coordinate tags in the XML files (from the year 2014), which followed an inconsistent order compared to the other years.

5.1.3 YOLOv2 Dataset Conversion

YOLOv2 [17] is a real-time object detection framework originally written in a Deep Learning framework called 'darknet' [25] (written in programming language, 'C') however, due to 'C' not being user-friendly, this project uses the Python-based version of 'darknet' which is called 'darkflow' [18]. For more information on YOLOv2, please go to section 5.4.3.

YOLOv2 is combined with Tensorflow which share the same data format (XML), hence no dataset conversion was needed from the dataset used for Tensorflow.

5.2 Data Augmentation

Augmentation of data involves the data to be changed in various ways to increase the performance of the model. Data was changed in terms of colour and shuffling train/validation images. Other techniques for data augmentation may include [27]:

- Scaling
- Translation
- Rotation (at 90 degrees)
- Rotation (at finer angles)
- Flipping
- Adding Salt and Pepper noise
- Lighting condition
- Perspective transform

Augmentation is needed is because it can generate additional data if there is a lack of quantity issues with the current dataset. For this project's case, there were fewer data however, more images were not needed since YOLOv2 and Tensorflow outputted great performance for the

trained models. Below, Figure 7 displays an image retrieved from the dataset with the change of lighting that occurred during the augmentation process. This change of lighting was not a great method as the neural network considered colour as a more meaningful feature than grayscale. As grayscale turned any coloured ball to white, it did not recognise the ping-pong balls efficiently in different lighting environments. Therefore, coloured dataset was further used during the project.

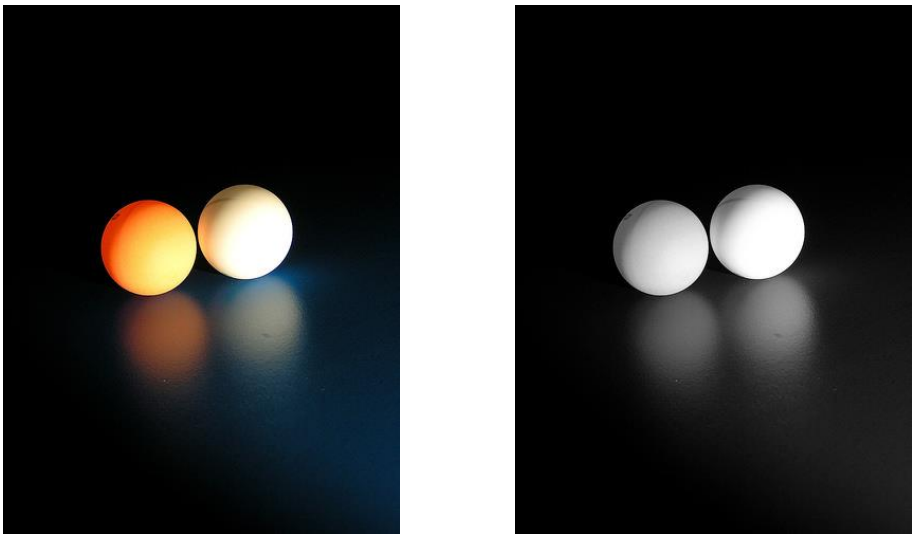


Figure 7: The lighting condition on the image changed from original (left) to grayscale (right).

5.3 Conversion Code Snippets

5.3.1 Pascal Conversion

pascal_voc_convert.py

```
import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET

def convert_to_pascal_voc(path):
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        value = root.find('filename').text + ".jpg"
        tree.find('.//filename').text = value
        tree.write(xml_file)

def main():
    for directory in ['all']:
        image_path = os.path.join(os.getcwd(), 'dataset/pascal_dataset/{}'.format(directory))
        xml_df = convert_to_pascal_voc(image_path)

main()
```

Figure 8: Conversion code from new ILSVRC labels to old PASCAL VOC label style.

: Conversion code from new ILSVRC labels to old PASCAL VOC label style.Figure 37 shows the conversion code that was created to fix the error or new ILSVRC label files not containing the ".jpg" extension of the image in the XML files. Therefore, this fixed the labels into default (PASCAL) format.

5.3.2 Tensorflow – XML to CSV

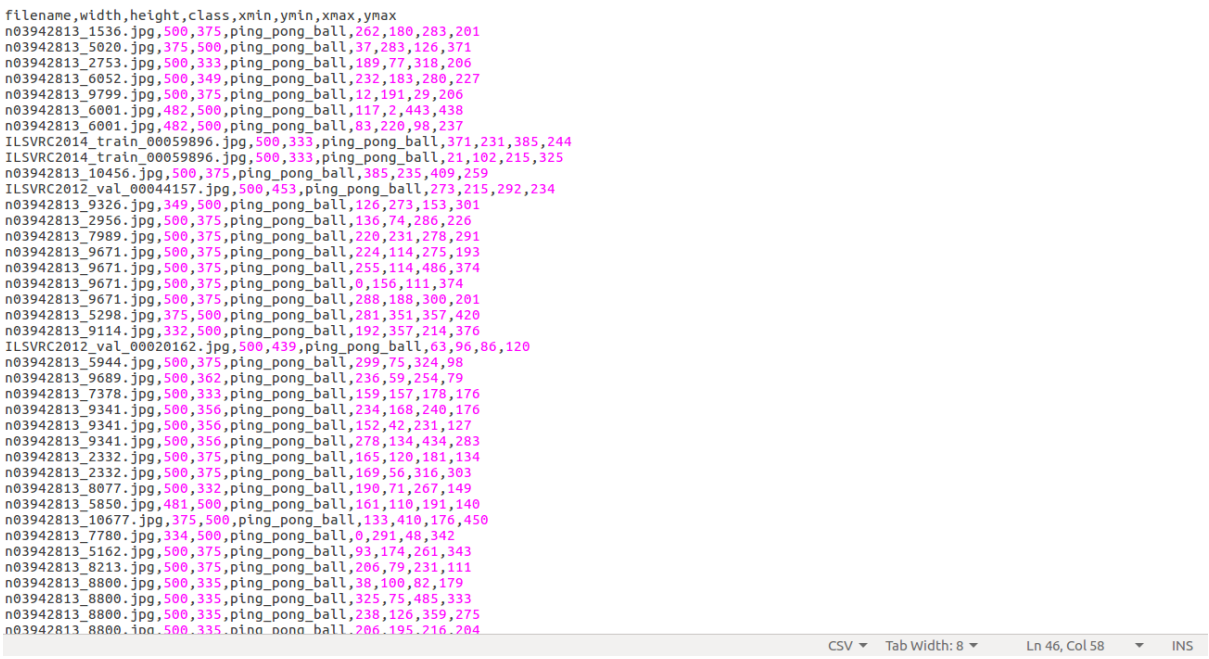
xml_to_csv.py

```python
import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET


def xml_to_csv(path):
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        xml_filename_ext = os.path.basename(xml_file)
        xml_filename = os.path.splitext(xml_filename_ext)[0]
        if ("ILSVRC" in xml_filename and "val" in xml_filename):
            for member in root.findall('object'):
                ping_pong_ball_str = "ping_pong_ball"
                value = (root.find('filename').text + ".jpg",
                         int(root.find('size')[0].text),
                         int(root.find('size')[1].text),
                         ping_pong_ball_str,
                         int(member[4][0].text),
                         int(member[4][1].text),
                         int(member[4][2].text),
                         int(member[4][3].text)
                         )
                xml_list.append(value)
        elif ("n03942813" in xml_filename) or ("ILSVRC" in xml_filename and "train" in xml_filename):
            for member in root.findall('object'):
                ping_pong_ball_str = "ping_pong_ball"
                value = (root.find('filename').text + ".jpg",
                         int(root.find('size')[0].text),
                         int(root.find('size')[1].text),
                         ping_pong_ball_str,
                         int(member[1][0].text),
                         int(member[1][2].text),
                         int(member[1][1].text),
                         int(member[1][3].text)
                         )
                xml_list.append(value)
    column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df


def main():
    for directory in ['train','val']:
        image_path = os.path.join(os.getcwd(), 'images/{}'.format(directory))
        xml_df = xml_to_csv(image_path)
        xml_df.to_csv('data/{}_labels.csv'.format(directory), index=None)
        print('Successfully converted xml to csv.')


main()
```

~/fyp_deeplearning_project/experiments/tensorflow_object_detection_api/ping_pong_ball/xml_to_csv.py 1:1 LF UTF-8 Python 0 files

Figure 9: Conversion code from XML to CSV format required by Tensorflow.

The above code (Figure 9) contains the steps needed to convert from XML (PASCAL) format to CSV. Tensorflow requires CSV format (as shown in Figure 10) in further steps when converting to TFRecord files as these are standard input format and an easier way to maintain a scalable architecture. For more information regarding TFRecords and the difference between “Naïve vs TFRecord” method, please visit [28].



```
filename,width,height,class,xmin,ymin,xmax,ymax
n03942813_1536.jpg,500,375,ping_pong_ball,262,180,283,201
n03942813_5020.jpg,375,500,ping_pong_ball,37,283,126,371
n03942813_2753.jpg,500,333,ping_pong_ball,189,77,318,206
n03942813_6052.jpg,500,349,ping_pong_ball,232,183,280,227
n03942813_9799.jpg,500,375,ping_pong_ball,12,191,29,206
n03942813_6001.jpg,482,500,ping_pong_ball,117,2,443,438
n03942813_6001.jpg,482,500,ping_pong_ball,83,220,98,237
ILSVRC2014_train_00059896.jpg,500,333,ping_pong_ball,371,231,385,244
ILSVRC2014_train_00059896.jpg,500,333,ping_pong_ball,21,102,215,325
n03942813_10456.jpg,500,375,ping_pong_ball,385,235,409,259
ILSVRC2012_val_00044157.jpg,500,453,ping_pong_ball,273,215,292,234
n03942813_9326.jpg,349,500,ping_pong_ball,126,273,153,301
n03942813_2956.jpg,500,375,ping_pong_ball,136,74,286,226
n03942813_7989.jpg,500,375,ping_pong_ball,220,231,278,291
n03942813_9671.jpg,500,375,ping_pong_ball,224,114,275,193
n03942813_9671.jpg,500,375,ping_pong_ball,255,114,486,374
n03942813_9671.jpg,500,375,ping_pong_ball,0,156,111,374
n03942813_9671.jpg,500,375,ping_pong_ball,288,188,300,201
n03942813_5298.jpg,375,500,ping_pong_ball,281,351,357,420
n03942813_9114.jpg,332,500,ping_pong_ball,192,357,214,376
ILSVRC2012_val_00020162.jpg,500,439,ping_pong_ball,63,96,86,120
n03942813_5944.jpg,500,375,ping_pong_ball,299,75,324,98
n03942813_9689.jpg,500,362,ping_pong_ball,236,59,254,79
n03942813_7378.jpg,500,333,ping_pong_ball,159,157,178,176
n03942813_9341.jpg,500,356,ping_pong_ball,234,168,240,176
n03942813_9341.jpg,500,356,ping_pong_ball,152,42,231,127
n03942813_9341.jpg,500,356,ping_pong_ball,278,134,434,283
n03942813_2332.jpg,500,375,ping_pong_ball,165,120,181,134
n03942813_2332.jpg,500,375,ping_pong_ball,169,56,316,303
n03942813_8077.jpg,500,332,ping_pong_ball,190,71,267,149
n03942813_5850.jpg,481,500,ping_pong_ball,161,110,191,140
n03942813_10677.jpg,375,500,ping_pong_ball,133,410,176,450
n03942813_7780.jpg,334,500,ping_pong_ball,0,291,48,342
n03942813_5162.jpg,500,375,ping_pong_ball,93,174,261,343
n03942813_8213.jpg,500,375,ping_pong_ball,206,79,231,111
n03942813_8800.jpg,500,335,ping_pong_ball,38,100,82,179
n03942813_8800.jpg,500,335,ping_pong_ball,325,75,485,333
n03942813_8800.jpg,500,335,ping_pong_ball,238,126,359,275
n03942813_8800.jpg,500,335,ping_pong_ball,206,195,216,204
```

CSV ▾ Tab Width: 8 ▾ Ln 46, Col 58 ▾ INS

Figure 10: A successfully converted CSV file.

5.4 Training a Deep Neural Network for Object Detection

There were three different Deep Learning frameworks used throughout the entirety of this project:

- DetectNet (DIGITS by NVIDIA)
- Tensorflow (by Google)
- YOLOv2

5.4.1 DetectNet

The DetectNet architecture is used by the DIGITS platform which performs common deep learning tasks while displaying real-time training performance. Figure 11 shows the process of how DIGITS represent the labelled images to train DetectNet. Each grid square is labelled with two key pieces of information: the class of object present in the grid square and the pixel coordinates of the corners of the bounding box of that object relative to the center of the grid square. In the case where no object is present in the grid square, a special “dontcare” class is used so that the data representation maintains a fixed size. A coverage value of 0 or 1 is also provided to indicate whether an object is present within the grid square. In the case where multiple objects are present in the same grid square, DetectNet selects the object that occupies the most pixels within the grid square. In the case of a tie, the object with the bounding box with the lowest y-value is used [16].

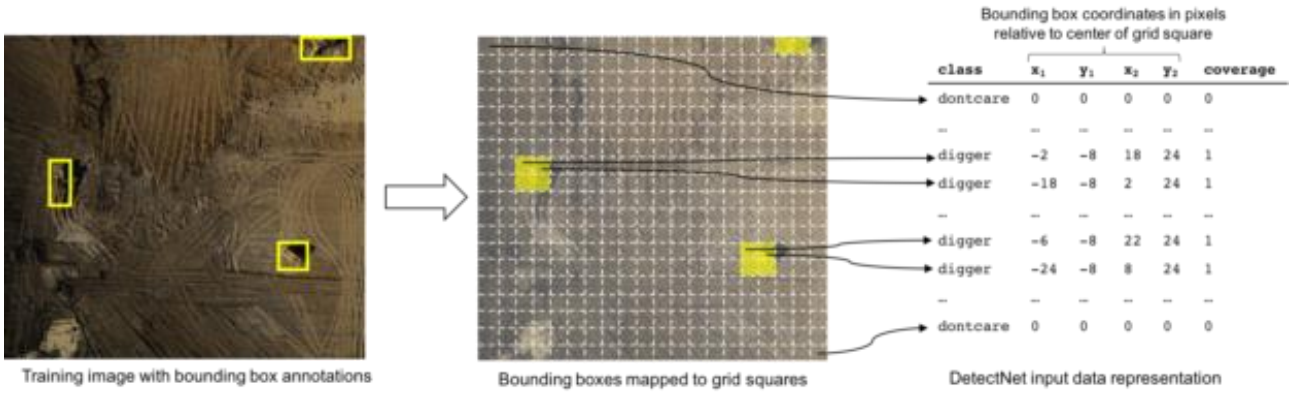


Figure 11: An example representation of DetectNet input data.

The DetectNet Architecture

[16] The DetectNet architecture has **five tasks** that are specified in the Caffe model definition file. Figure 7 displays the **training process** throughout the architecture highlighting three key stages:

1) Data layers ingest the training images and labels and a transformer layer applies online data augmentation.
2) A fully-convolutional network (FCN) performs feature extraction and prediction of object classes and bounding boxes per grid square.
3) Loss functions simultaneously measure the error in the two tasks of predicting the object coverage and object bounding box corners per grid square.

Figure 378 shows the **validation process** within the architecture with its two main stages:

4) A clustering function produces the final set of predicted bounding boxes during validation.
5) A simplified version of the mean Average Precision (mAP) metric is computed to measure model performance against the validation dataset.

DetectNet Training Process

To train a neural network, it is essential to use a GPU if the dataset size is large. In the initial process, there were problems that occurred while installing several libraries required for DIGITS such as NVIDIA Caffe (preferred version 0.15.1+), DIGITS v5 and other prerequisites for the framework. These libraries were not supporting specific versions which led to large amount of time spent to successfully install DIGITS with NVIDIA Caffe [15].

There were two major setbacks which led to migrating the training framework to Tensorflow:

1. The GPU contained problems with installations of libraries which led to enormous time being spent.
2. After training the model, Digits only showed inference on the platform for each image. There was no other way (at the time) to inference through webcam except by embedding NVIDIA Jetson modules [21] into the GPU which are costly and used C++ for inferencing which was not possible to be integrated with the robotic arm.

5.4.2 Tensorflow

Tensorflow is an open source library for numerical computation developed by the Google Brain team. It was primarily created for machine learning and deep learning network research. Detecting objects in images has been around for a long time however, detecting images in a video or a stream with a decent accuracy was a challenge. However, Tensorflow has tackled this problem and released their framework to the public while making it easy to train a model.

The API has been trained on the COCO (Common Objects in Context) dataset consisting of 90 most common found objects with a total of 300K images. It also provides a great selection of models from its 'Model Zoo' including:

- Single Shot Multibox Detector (SSD) with MobileNet,
- SSD with Inception V2,
- Region-Based Fully Convolutional Networks (R-FCN) with Resnet 101,
- Faster RCNN with Resnet 101,
- Faster RCNN with Inception Resnet v2

Figure 5 displays the sample range of models that are available to be tested in a real environment. To compare the accuracy and speed of the pre-trained models, I selected the **'ssd_mobilenet_v1_coco'** and **'faster_rcnn_resnet101_coco'** model to be inferenced through the web-cam. The results were very similar to the ones shown in Figure 6 [22]. Consequently, it was decided to train the custom object of 'Ping-Pong Ball' on the MobileNet pre-trained model as it was currently the fastest with a decent mAP.

Furthermore, this comparison improved my knowledge with **two key points**:

1. Good performance on small objects correlates with performance on bigger objects.
2. Input resolution affects detection accuracy of small objects. (as shown in Figure 25)

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		Boxes

Figure 12: Sample selection of models trained on the COCO dataset.

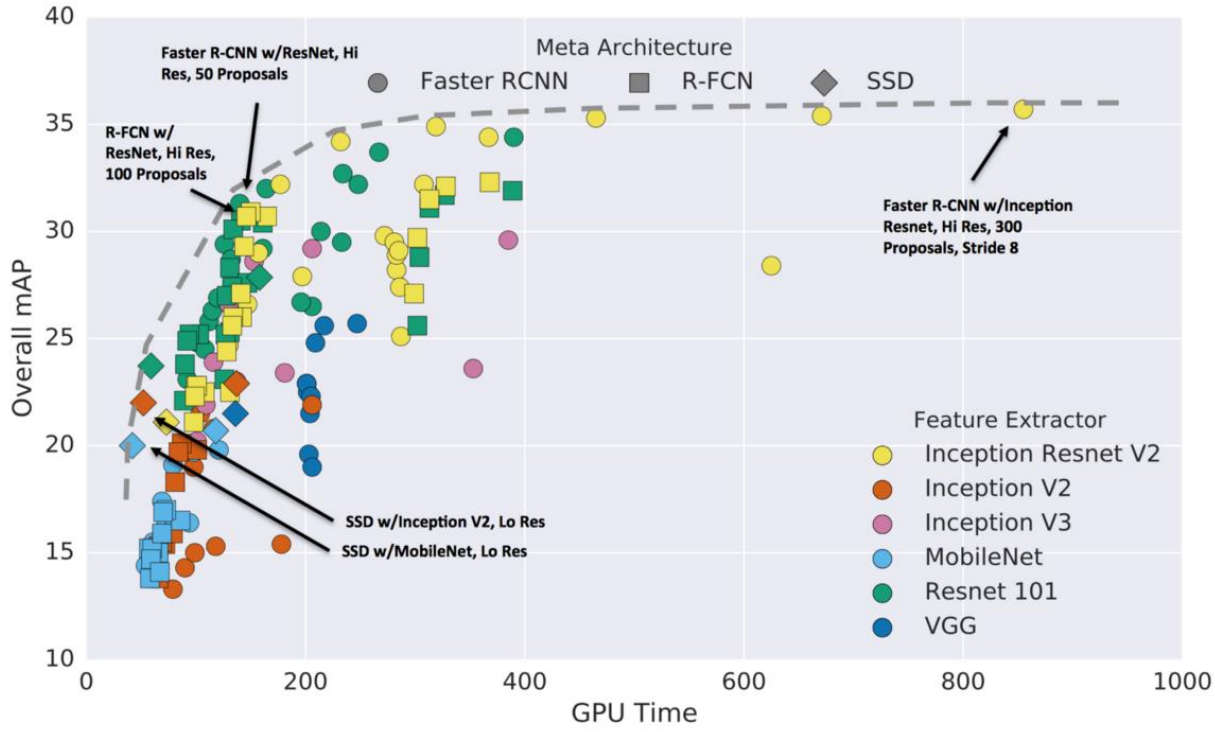


Figure 13: Speed-accuracy trade-offs for modern convolutional object detectors.

Tensorflow Object Detection Classifier Training Steps

The following steps were followed to train a model in Tensorflow:

1. Install pre-requisite libraries including Tensorflow-GPU.

2. Create the Object Detection directory structure (by cloning the 'models' directory from Github).
3. Gather and label images.
4. Generate training data.
    a. Convert all the XML to CSV (Comma-Separated Values) file for both 'train' and 'validation' set of images.
    b. Generate each '**TFRecord'** [14] from the previously created CSV files.
5. Create label map (Protobuf text (pbtxt) file) and configure training by choosing the MobileNet model.
6. Train the object detector.
7. Export inference graph (See further about inference in section 5.5)

Tensorflow has several advantages over DetectNet giving it the ability to be more efficient for training CNNs. Firstly, Tensorflow has a feature called **'checkpoint'** which is triggered when a model is in the training phase. Checkpoints provide an easy-to-use and an automatic mechanism for saving and restoring models. Therefore, it is possible to stop a model being trained after certain number of steps and retraining that same model from the same step. This provides time around the user as some may not want to leave the model running overnight. However, if DetectNet training is stopped, one would have to re-train the model from start.

Another advantage of Tensorflow is that it is simple to inference as it contains sample source code ready to inference a pre-trained model. Finally, Tensorflow contains a wide variety of pre-trained models compared to DetectNet and there are more users dwelling on this framework providing help to beginner users in comparison to DetectNet (Digits).

The graph named, **Total Loss**, indicates the loss of the model which is displayed through 'Tensorboard'; a real-time graph visualisation tool. A key ideology for loss graph is that, 'the lower the loss, the better the model'. It is clearly visible in Figure 14 that the loss is gradually being minimized after certain number of epochs/iterations. The loss of the model is calculated by interpreting how well the model is performing for the training and validation datasets. However, due to the low amount of data in each set, the loss value fluctuated frequently until its learning rate (Figure 15) was at minimum.

For this model, graphs such as '**Loss of No. of Positives**' and '**Loss of No. of Negatives**' were also visualised to test the suitability of the model on new data. These graphs can be viewed in the

Appendices as Figure 38 and Figure 39 respectively.

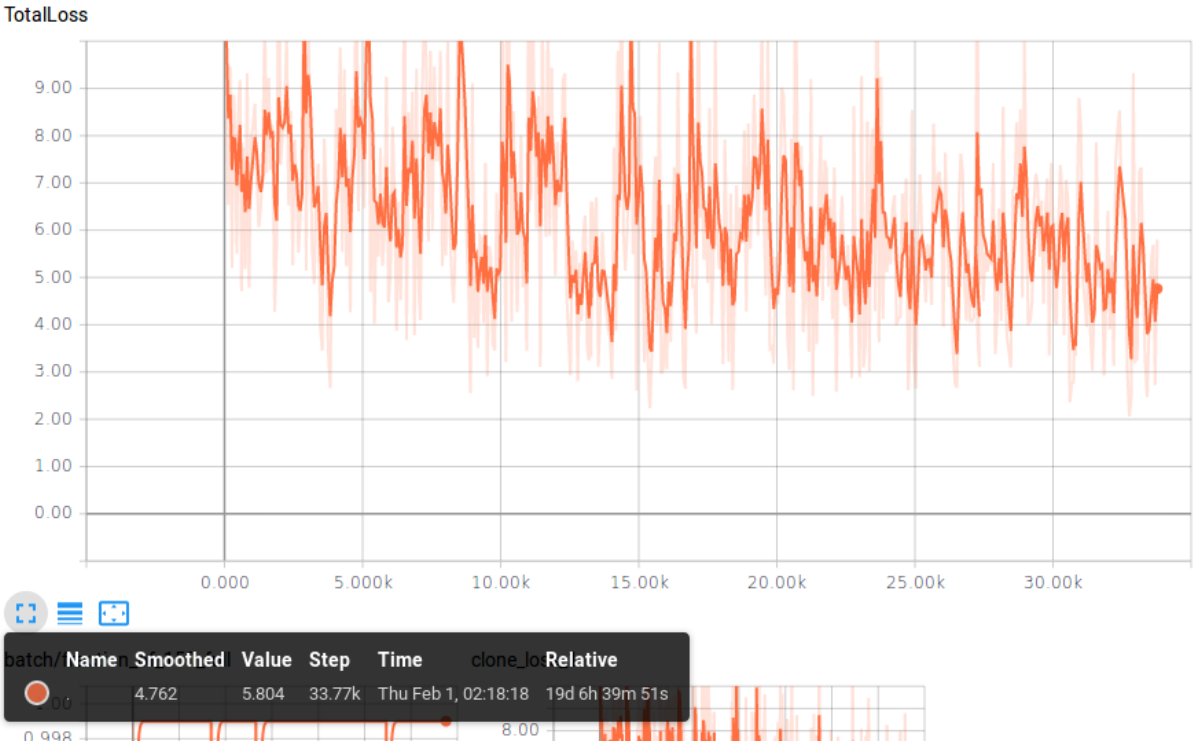


Figure 14: Total Loss graph of the trained model on Tensorflow.

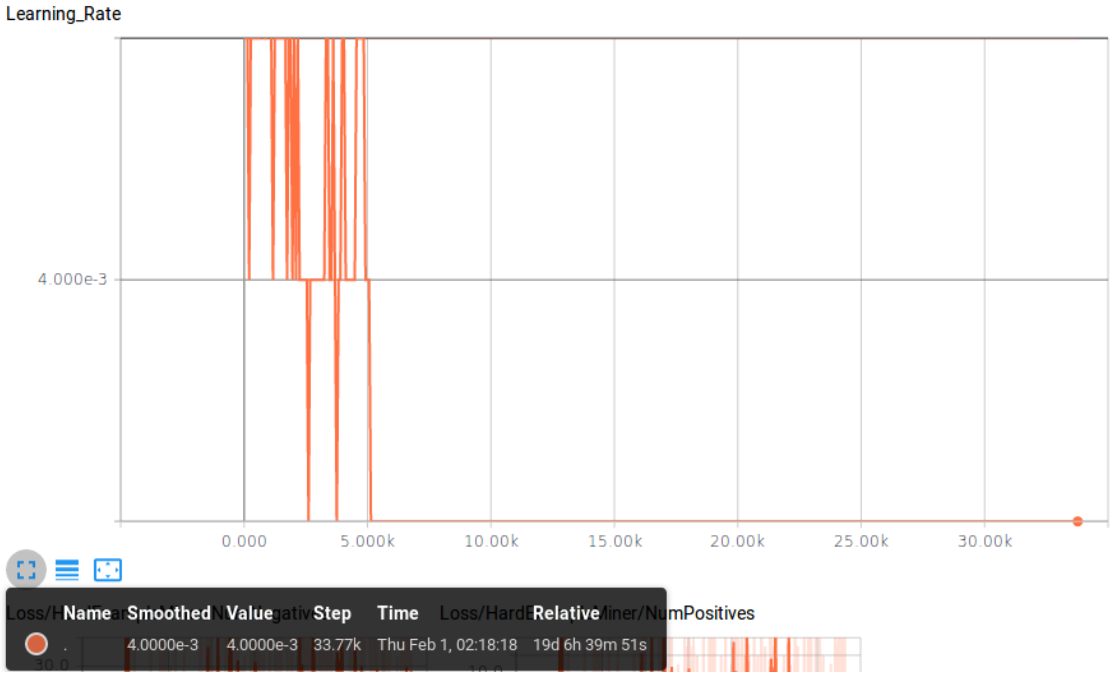


Figure 15: Learning Rate graph of the trained model on Tensorflow.

5.4.3 YOLOv2

YOLO (You Only Look Once) is a state-of-the-art real-time Object Detection which uses a completely different approach to modern classifiers. Modern approaches to object detection are currently extensions of image classification models. Some of the examples of current models are R-CNN, Fast R-CNN, Faster R-CNN, R-FCN and SSD [23].

However, YOLOv2 has currently the best approach compared to the other models. The rationale behind calling the system YOLO is that rather than pass in multiple sub-images of potential objects, you only pass in the whole image to the deep learning system once. Then, you would get all the bounding boxes as well as the object category classifications in one go. This is the fundamental design decision of YOLO and is what makes it a refreshing new perspective on the task of object detection [24]. YOLO subdivides the image into a 13x13 grid where each cell (anchor) is responsible for predicting 5 bounding boxes. A bounding box describes the rectangle that encloses an object. Additionally, YOLO outputs a confidence score that tells us how certain it is that the predicted bounding box encloses an object. The full process is shown in Figure 7 [26].

During the PASCAL VOC detection challenge dataset, YOLOv2 achieved a mAP of 63.4 (out of 100) at 45 FPS rate compared to the Faster R-CNN model which generated a mAP of 73.2 but only at a maximum of 7 FPS rate. YOLOv2 outperformed R-CNN and all its variants. The comparison of YOLO to other object detection frameworks is show in Figure 8.

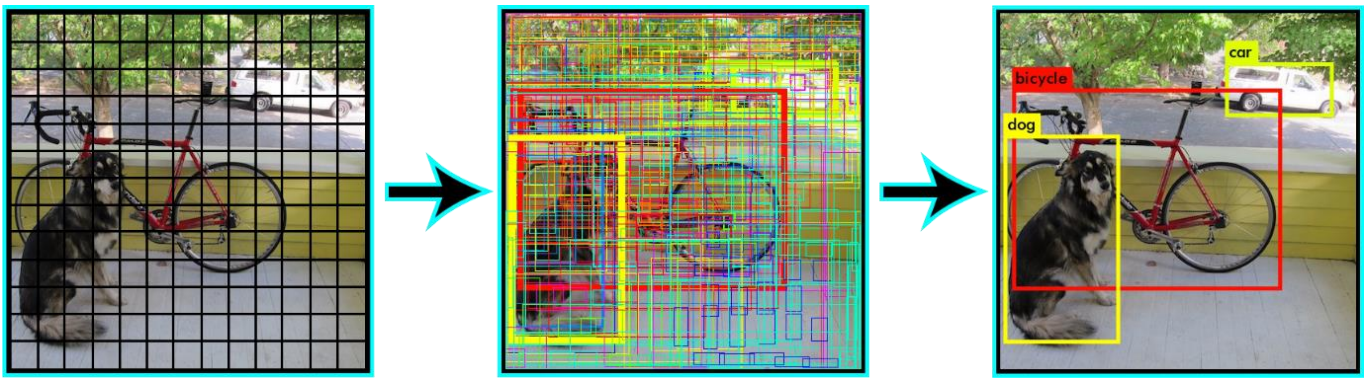


Figure 16: The YOLO object detection model.

Detection Framework	mAP	FPS
Faster RCNN - VGG16	73.2	7
Faster RCNN - ResNet	76.4	5
YOLO	63.4	45
SSD 500	76.8	19
YOLO v2 (416x416 image size)	76.8	67
YOLO v2 (480x480 image size)	77.8	59

Figure 17: Comparison of object detection frameworks with respect to mAP and FPS rate.

YOLO Training Process

The training for custom object (ping-pong ball) was simple on this framework as there were not many changes needed to be made to the pre-trained model. The training process was mentioned on the Github repository [26] of the framework where the only changes made were in the configuration file by changing the number of 'classes' in the 'region' layer. Furthermore, inside the 'convolutional' layer, the 'filters' variable was changed to match for the number of classes used in the 'region' layer. Finally, the 'labels.txt' was updated to only contain the ping-pong ball label. However, the label was changed to "sports ball" due to the length of the label which did not fit into the frame window hence, blocking the confidence score in the frame. The pre-trained weights were loaded from the original YOLO website [26].

Once the training commenced, the number of epochs will show the 'loss' and 'average loss' for each step. To get a desired outcome from the model, it was essential to leave the training running until the loss did not change and below the value 1.0.

5.5 Inference Visualisation

Inferencing in AI is similar to human inferencing as it is the process of inferring new images based on previously trained dataset. The different types of frameworks used during this project produced a variety of inferences where some were faulty (also known as false positive/negative). These are measured in the form of a Confusion Matrix which consists of four outcomes:

1. True Positive – Correct positive prediction
2. False Positive – Incorrect positive prediction
3. True Negative – Correct negative prediction

4. False Negative – Incorrect negative prediction

It is essential to minimise both FP and FN however, FN should be drastically reduced as it holds the chance to skip actual classified object. Table 1 below shows the confusion matrix which was deduced from this project. It contains data from **144 validation images** and inferenced within Tensorflow.

		Predicted Class	
		Ping-Pong Ball	**Not Ping-Pong Ball**
Actual Class	**Ping-Pong Ball**	TP = 96	FP = 15
	Not Ping-Pong Ball	FN = 3	TN = 30

Table 1: Confusion matrix for measuring ping-pong ball detections.

Tensorflow Inference

Tensorflow provides sample detection code to be used for images however, the code had to be tweaked to be used for real-time video inferencing. As mentioned previously, tennis ball was the first category to be trained for this project which had a **final test accuracy of 94.1%** (as shown below in Figure 18).

```
INFO:tensorflow:Looking for images in 'tennis_ball'
INFO:tensorflow:Looking for images in 'not_tennis_ball'
2018-01-08 17:25:22.012685: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU supports instructions that this TensorFlow binary was
not compiled to use: SSE4.1 SSE4.2 AVX
INFO:tensorflow:2018-01-08 17:25:25.767205: Step 0: Train accuracy = 71.9%
INFO:tensorflow:2018-01-08 17:25:25.767402: Step 0: Cross entropy = 0.643731
INFO:tensorflow:2018-01-08 17:25:26.104434: Step 0: Validation accuracy = 63.9% (N=866)
INFO:tensorflow:2018-01-08 17:31:04.133633: Step 100: Train accuracy = 87.5%
INFO:tensorflow:2018-01-08 17:31:04.133799: Step 100: Cross entropy = 0.380650
INFO:tensorflow:2018-01-08 17:31:04.427618: Step 100: Validation accuracy = 90.1% (N=866)
INFO:tensorflow:2018-01-08 17:36:50.514907: Step 200: Train accuracy = 90.6%
INFO:tensorflow:2018-01-08 17:36:50.515084: Step 200: Cross entropy = 0.265949
INFO:tensorflow:2018-01-08 17:36:50.875688: Step 200: Validation accuracy = 93.2% (N=866)
INFO:tensorflow:2018-01-08 17:42:33.396050: Step 300: Train accuracy = 90.6%
INFO:tensorflow:2018-01-08 17:42:33.396225: Step 300: Cross entropy = 0.229619
INFO:tensorflow:2018-01-08 17:42:33.715875: Step 300: Validation accuracy = 94.2% (N=866)
INFO:tensorflow:2018-01-08 17:48:19.087266: Step 400: Train accuracy = 90.6%
INFO:tensorflow:2018-01-08 17:48:19.087499: Step 400: Cross entropy = 0.202267
INFO:tensorflow:2018-01-08 17:48:19.597522: Step 400: Validation accuracy = 94.2% (N=866)
INFO:tensorflow:2018-01-08 17:54:04.797022: Step 500: Train accuracy = 96.9%
INFO:tensorflow:2018-01-08 17:54:04.797191: Step 500: Cross entropy = 0.157078
INFO:tensorflow:2018-01-08 17:54:05.092401: Step 500: Validation accuracy = 94.8% (N=866)
INFO:tensorflow:2018-01-08 17:59:48.038660: Step 599: Train accuracy = 96.9%
INFO:tensorflow:2018-01-08 17:59:48.038820: Step 599: Cross entropy = 0.120474
INFO:tensorflow:2018-01-08 17:59:48.329501: Step 599: Validation accuracy = 94.8% (N=866)
INFO:tensorflow:Final test accuracy = 94.1% (N=810)
INFO:tensorflow:Froze 2 variables.

Not extracting or downloading files, model already present in disk
Model path:  /tmp/imagenet/mobilenet_v1_1.0_224/frozen_graph.pb
Converted 2 variables to const ops.
```

Figure 18: Tensorflow – MobileNet accuracy of 'Tennis Ball'.

Moreover, to test the accuracy of the model, it was essential to use a test image from both categories and calculate the accuracy. The commands and its associated accuracy of the image was outputted into the terminal as shown in Figure 19 and Figure 20. The full comparison of accuracies against the FPS rate between all frameworks used can be found in section 5.5.2.

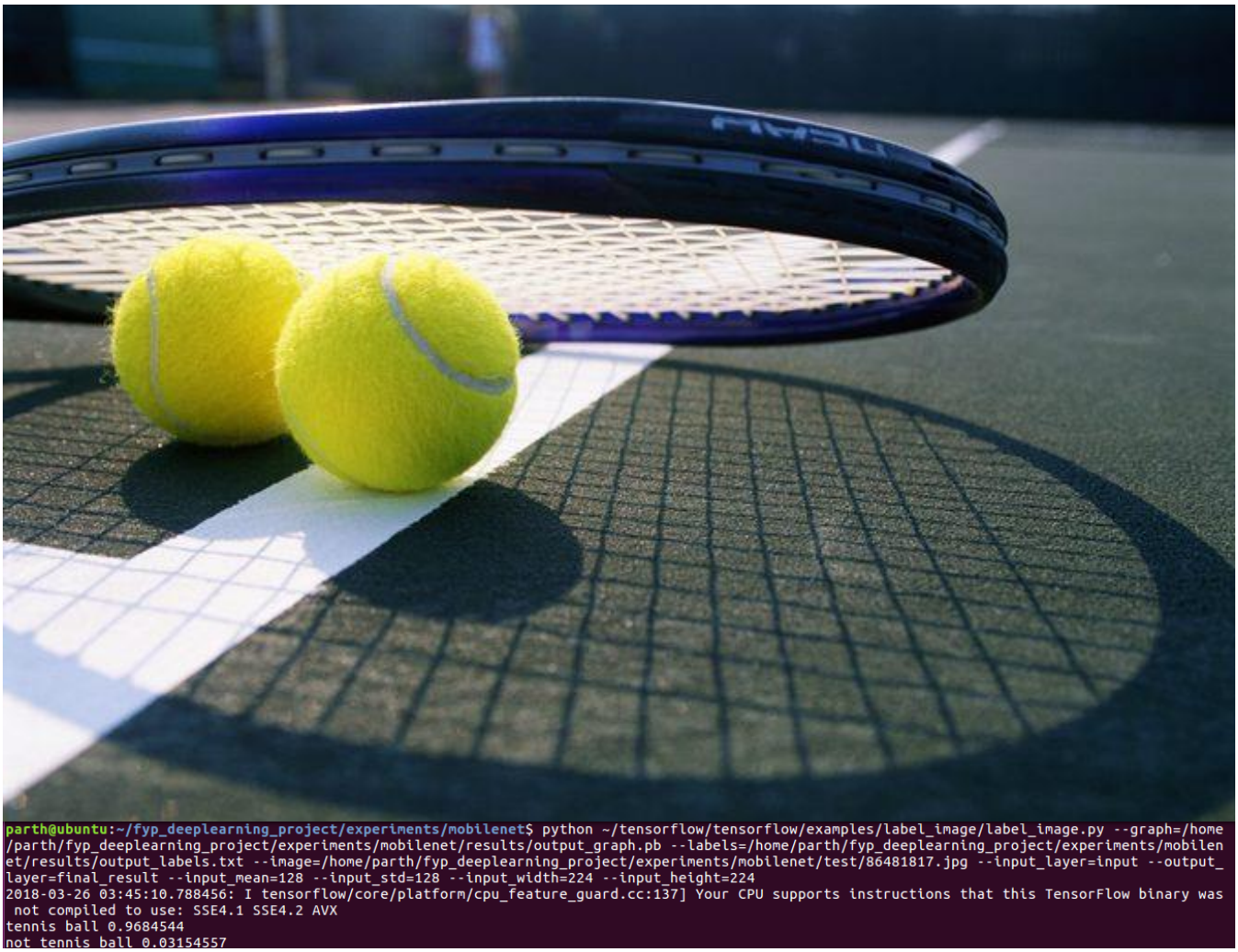


Figure 19: Trained MobileNet model processed on a test image of the 'tennis ball' category.

Figure 20: Trained MobileNet model processed on a test image of a different category.

As ping-pong ball was later introduced to the project, the model was separately trained in this category. However, due to less images for the training set, the final test accuracy generated was at 85%. The detections were still outstanding in certain images (

Figure 21 and

Figure 22). Due to the lack of adequate lighting and occluded ping-pong balls because of the network not being capable enough to extract certain features of the ball, some images were not detected (

Figure 23).

Figure 21: Ping-Pong ball detected using the Tensorflow trained model (1/2).

Figure 22: Ping-Pong ball detected using the Tensorflow trained model (2/2).

Figure 23: Sample images where detections did not occur with Tensorflow trained model.

YOLOv2 Inference

Inferencing in the YOLO framework was challenging due to the lack of sample code at the time. Therefore, it was difficult to script for inferencing through a webcam. The accuracy produced by the model was **72%** from the total of 477 images that were used. Figure 24 shows the successful detections from the model even though, the model was unsuccessful in detecting from certain images (as shown in Figure 25). The model was unsuccessful because YOLOv2's approach for detection is different compared to other frameworks. The neural network is applied to the full image compared to other detection systems which repurpose classifiers by applying the model to regions of the image and considering high scoring regions as detections. For further information, visit section 5.4.3.

Figure 24: Group of images that were tested using the trained YOLOv2 model.

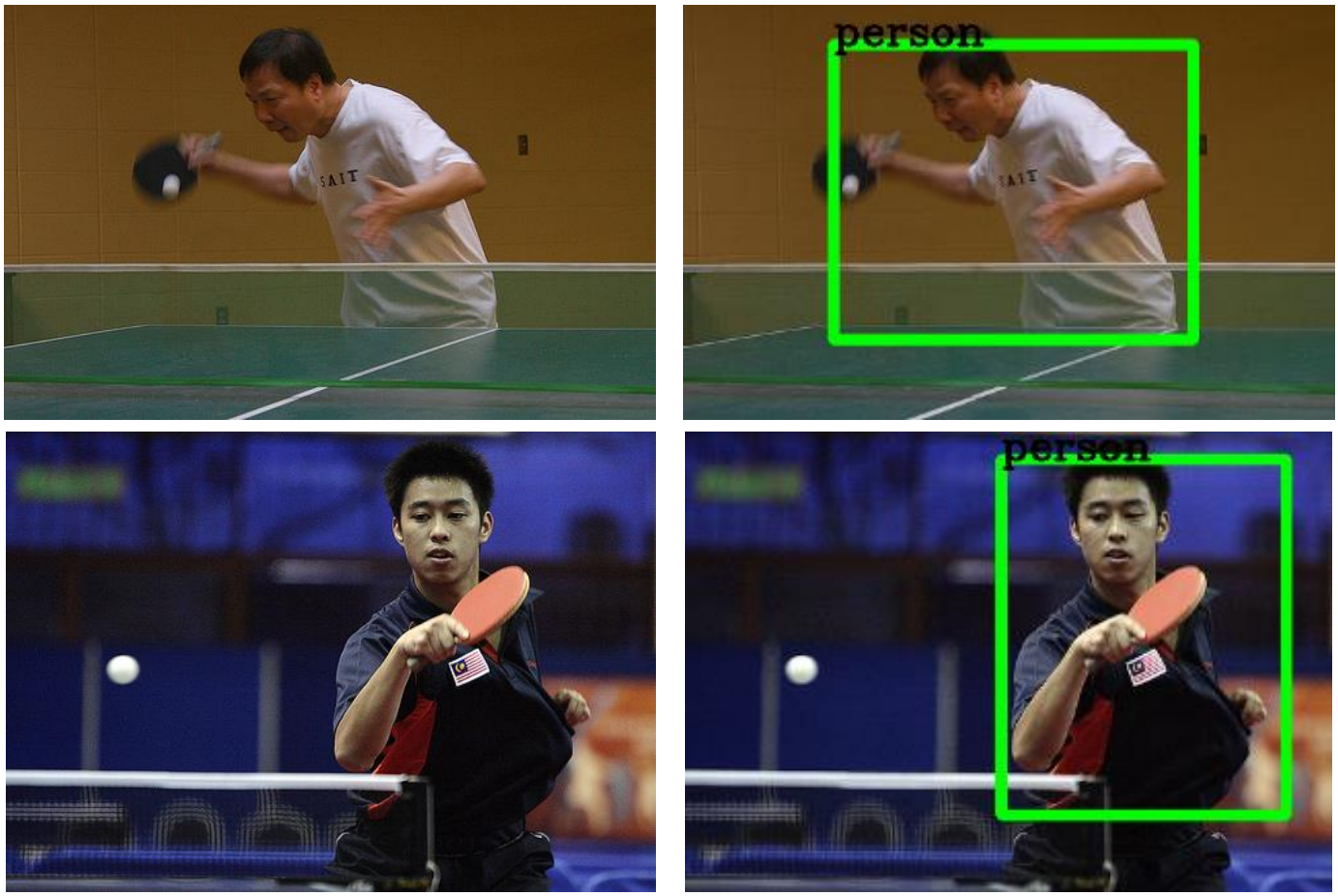


Figure 25: Original images and processed images showing incorrect detections of ball.

As shown in the images above, **persons** were detected rather than the ping-pong balls. The neural network did not learn the features for the ball with more blurred images. Although, the balls stand out in the background, the detections failed. In comparison, Tensorflow's detection produced accurate results at a lower FPS rate. The GPU could not be used during processing due to the lack of compute capability (as shown in Figure 26). Inferencing on a GPU would be an optimum choice for the YOLO framework as it is by-far the fastest framework detection system yet. To see the YOLOv2 inferencing in action, please visit the following links:

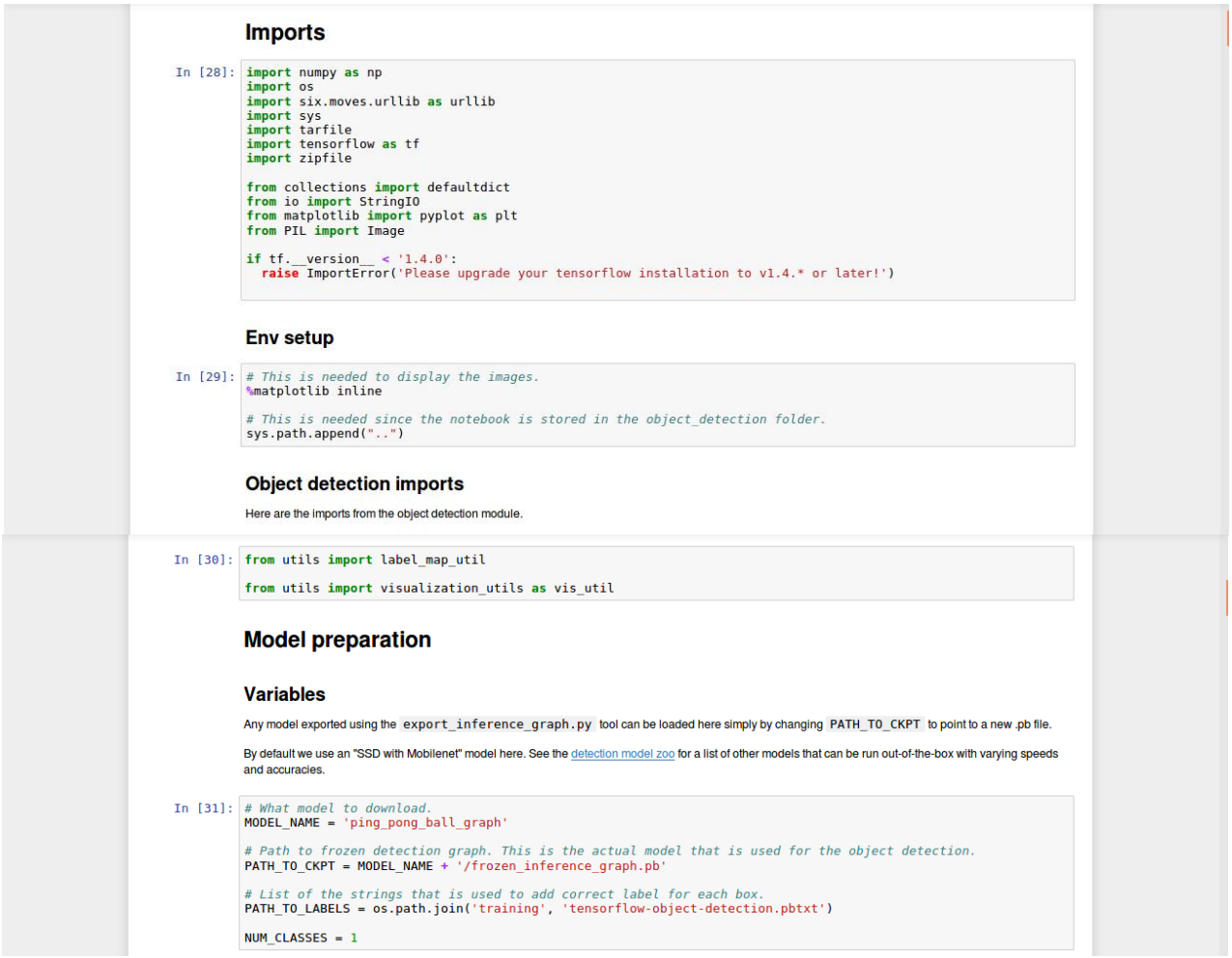
- Single ping-pong ball detection: https://www.youtube.com/watch?v=VWSaJcT6S2Y
- Multiple ping-pong balls detection: https://www.youtube.com/watch?v=g0T7vHr_gVY
- YOLOv2 inferencing before training: https://www.youtube.com/watch?v=QtSFVG_9_o4

```
GPU mode with 1.0 usage
2018-03-21 19:08:45.532425: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:895] successful NUMA node read from SysFS had nega
tive value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2018-03-21 19:08:45.532810: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1105] Found device 0 with properties:
name: Tesla C2070 major: 2 minor: 0 memoryClockRate(GHz): 1.147
pciBusID: 0000:01:00.0
totalMemory: 5.17GiB freeMemory: 4.82GiB
2018-03-21 19:08:45.532850: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1168] Ignoring visible gpu device (device: 0, name: Te
sla C2070, pci bus id: 0000:01:00.0, compute capability: 2.0) with Cuda compute capability 2.0. The minimum required Cuda capability
is 3.5.
```

Figure 26: Error produced by using a low compute capability GPU on inferencing.

5.5.1 Inference Code Snippets

Tensorflow



Imports

In [28]:
```
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile

from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image

if tf.__version__ < '1.4.0':
  raise ImportError('Please upgrade your tensorflow installation to v1.4.* or later!')
```

Env setup

In [29]:
```
# This is needed to display the images.
%matplotlib inline

# This is needed since the notebook is stored in the object_detection folder.
sys.path.append("..")
```

Object detection imports

Here are the imports from the object detection module.

In [30]:
```
from utils import label_map_util

from utils import visualization_utils as vis_util
```

Model preparation

Variables

Any model exported using the `export_inference_graph.py` tool can be loaded here simply by changing `PATH_TO_CKPT` to point to a new .pb file.

By default we use an "SSD with Mobilenet" model here. See the detection model zoo for a list of other models that can be run out-of-the-box with varying speeds and accuracies.

In [31]:
```
# What model to download.
MODEL_NAME = 'ping_pong_ball_graph'

# Path to frozen detection graph. This is the actual model that is used for the object detection.
PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'

# List of the strings that is used to add correct label for each box.
PATH_TO_LABELS = os.path.join('training', 'tensorflow-object-detection.pbtxt')

NUM_CLASSES = 1
```

Load a (frozen) Tensorflow model into memory.

In [32]:
```
detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
    serialized_graph = fid.read()
    od_graph_def.ParseFromString(serialized_graph)
    tf.import_graph_def(od_graph_def, name='')
```

Loading label map

Label maps map indices to category names, so that when our convolution network predicts `5`, we know that this corresponds to `airplane`. Here we use internal utility functions, but anything that returns a dictionary mapping integers to appropriate string labels would be fine

In [33]:
```
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=NUM_CLASSES, use_displa
category_index = label_map_util.create_category_index(categories)
```

Helper code

In [34]:
```
def load_image_into_numpy_array(image):
  (im_width, im_height) = image.size
  return np.array(image.getdata()).reshape(
      (im_height, im_width, 3)).astype(np.uint8)
```

Detection

In [35]:
```
# For the sake of simplicity we will use only 2 images:
# image1.jpg
# image2.jpg
# If you want to test the code with your images, just add path to the images to the TEST_IMAGE_PATHS.
PATH_TO_TEST_IMAGES_DIR = 'test_images'
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image{}.jpg'.format(i)) for i in range(21, 33) ]

# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8)
```

In [36]:
```
with detection_graph.as_default():
  with tf.Session(graph=detection_graph) as sess:
    # Definite input and output Tensors for detection_graph
    image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
    # Each box represents a part of the image where a particular object was detected.
    detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
    # Each score represent how level of confidence for each of the objects.
    # Score is shown on the result image, together with the class label.
    detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
    detection_classes = detection_graph.get_tensor_by_name('detection_classes:0')
    num_detections = detection_graph.get_tensor_by_name('num_detections:0')
    for image_path in TEST_IMAGE_PATHS:
      image = Image.open(image_path)
      # the array based representation of the image will be used later in order to prepare the
      # result image with boxes and labels on it.
      image_np = load_image_into_numpy_array(image)
      # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
      image_np_expanded = np.expand_dims(image_np, axis=0)
      # Actual detection.
      (boxes, scores, classes, num) = sess.run(
          [detection_boxes, detection_scores, detection_classes, num_detections],
          feed_dict={image_tensor: image_np_expanded})
      # Visualization of the results of a detection.
      vis_util.visualize_boxes_and_labels_on_image_array(
          image_np,
          np.squeeze(boxes),
          np.squeeze(classes).astype(np.int32),
          np.squeeze(scores),
          category_index,
          use_normalized_coordinates=True,
          line_thickness=8)
      plt.figure(figsize=IMAGE_SIZE)
      plt.imshow(image_np)
```

Figure 27: The full code for object detection in images through Tensorflow.

From the above code (Figure 27), the important area of the inferencing commences from the '**Model Preparation**' stage where the trained model and config files are assigned to variables. The assigned graph/model is then loaded into the memory which is later used in the main 'with' statement which ensures the code within it is "cleaned" after being used. Furthermore, the 'for' loop inside the 'with' statement traverses through all the testing images and draws the bounding box, displays the confidence score and its associated annotation for each detected object. However, this code was optimised to apply to web-cam frames (check Appendices).

YOLOv2

```
    def detect_ball(self):
        #MOVEMENT TESTING
        #self.move_robot_arm(self.arm, 85)
        #self.default_robot_pos(self.arm, 85)

        while True:
            stime = time.time()
            ret, frame = self.capture.read()
            results = self.tfnet.return_predict(frame)
            if ret:
                #Detect the aruco marker(s)
                corners = self.detect_aruco_marker(frame)

                x1 = 0.0
                y1 = 0.0
                x3 = 0.0
                y3 = 0.0

                for coords in corners:
                    x1 = coords[0][0][0]
                    y1 = coords[0][0][1]
                    x3 = coords[0][2][0]
                    y3 = coords[0][2][1]

                #Automatically detect the aruco marker's center
                #x_center = int((x1 + x3) / 2)
                #y_center = int((y1 + y3) / 2)
                x_center = 306
                y_center = 325
                xy_coord = []

                if ((x_center and y_center) != 0):

                    aruco_center = x_center, y_center
                    #print aruco center
                    xy_coord = (aruco_center[0], aruco_center[1])
                    cv2.circle(frame, aruco_center, 3, (0, 0, 255), -1)
                    cv2.putText(frame,"marker center", (aruco_center[0]+10,aruco_center[1]), cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 0, 255),1)
                    cv2.putText(frame,"("+str(aruco_center[0])+","+str(aruco_center[1])+")", (aruco_center[0]+10,aruco_center[1]+15), cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 0, 25

                for color, result in zip(self.colors, results):
                    tl = (result['topleft']['x'], result['topleft']['y'])
                    br = (result['bottomright']['x'], result['bottomright']['y'])
                    ball_x_center = (result['topleft']['x'] + result['bottomright']['x']) / 2
                    ball_y_center = (result['topleft']['y'] + result['bottomright']['y']) / 2
                    ball_center = ball_x_center, ball_y_center
                    label = result['label']
                    print results
                    confidence = result['confidence']
                    if (label in self.labels_list):
                        text = '{}: {:.0f}%'.format(label, confidence * 100)
                        frame = cv2.rectangle(frame, tl, br, color, 5)
                        frame = cv2.putText(frame, text, tl, cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 0), 2)
                        cv2.circle(frame, ball_center, 3, (0, 0, 255), -1)
                        cv2.putText(frame,"circle center", (ball_center[0]+10,ball_center[1]), cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 0, 255),1)
                        cv2.putText(frame,"("+str(ball_center[0])+","+str(ball_center[1])+")", (ball_center[0]+10,ball_center[1]+15), cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 0, 25
                        #Calculate the angle between the aruco marker and the ball
                        self.arm.move(usb_arm.Stop, 10)
                        print "Finding a ball..."
                        angle = int(math.atan2((y_center - ball_center[1]), (ball_center[0] - x_center)) * 180 // math.pi)
                        print "Angle = ", angle
                        self.move_robot_arm(self.arm, angle)
                        self.default_robot_pos(self.arm, angle)
                cv2.imshow('frame', frame)
                print('FPS {:.1f}'.format(1 / (time.time() - stime)))
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        self.capture.release()
        cv2.destroyAllWindows()

if __name__ == '__main__':
    dlRoboticArmObj = DLRoboticArm()
    dlRoboticArmObj.detect_ball()

yolo_inference_webcam.py  128:1   LF  UTF-8  Python  master  1925 files
```

Figure 28: The full code for object detection in YOLOv2.

The 'detect_ball' function was written to loop through the frames from web-cam continuously while detecting the coordinates of **Aruco marker** (more information in section 5.8) and the center of the detected ball. These two coordinates would be fed to the **angle calculation formula** and assist the robotic arm to move the specified angle.

5.5.2 Accuracy vs FPS Rate Comparison

		FPS	
Frameworks	**Accuracy (%)**	**CPU**	**GPU**
Tensorflow	85	0.4	21
YOLOv2	72	0.5	26

Table 2: A comparison of frameworks against accuracies and FPS rate.

This table compares the frameworks used in this project and their related confidence scores that the trained model achieved with the FPS rates. As deduced, both frameworks have their advantages and disadvantages. Tensorflow's model achieved a higher accuracy where it could detect the blurred balls whereas YOLOv2's model could not. However, the accuracies could be improved on YOLOv2 by bringing additional training images.

The FPS rates were similar in both the frameworks however, YOLO is known for being the fastest detection system. Hence, it can achieve higher FPS rates with better GPUs likewise with Tensorflow.

5.6 Ball Tracking

Ball tracking is a method to locate the ball throughout all the frames from using the web-cam. To emphasise, the trained model performs detection on each frame and hence, it is portrayed as the ball is being tracked. There were two different approaches implemented for tracking such as using OpenCV and by using YOLOv2. I shall compare and discuss both approaches, stating the key advantage/disadvantage.

OpenCV provides a fast method to object tracking however, not by using Deep Learning. It uses contours and colours to find a specific ball. To find an orange ping-pong ball, specific HSV values were explored. Figure 29 shows the variety of interfaces produced from the Python code where the bottom left image consists of the 'mask' interface and the bottom right is of the 'threshold' interface. The trackbar includes the HSV range values for the orange ball however, if a different coloured ball was needed to be tracked, the trackbar would have to be altered. The HSV values were also adjusted to track the white ball however, the white semi-circle in the background took precedence over the white ball, resulting in an error of tracking (as shown in Figure 42). Alternatively, the YOLOv2 approach includes training a neural network to detect the ball in every frame that is fed from the web-cam.

In comparison, the OpenCV method is faster than YOLOv2 because it does not use any trained model to perform inference on frames. Inferencing per frame takes a huge amount of load on the CPU/GPU. However, since using OpenCV only works for certain colours, it is not effective to be used for detection of ping-pong balls even though it generates a faster FPS rate on a CPU compared to YOLOv2.

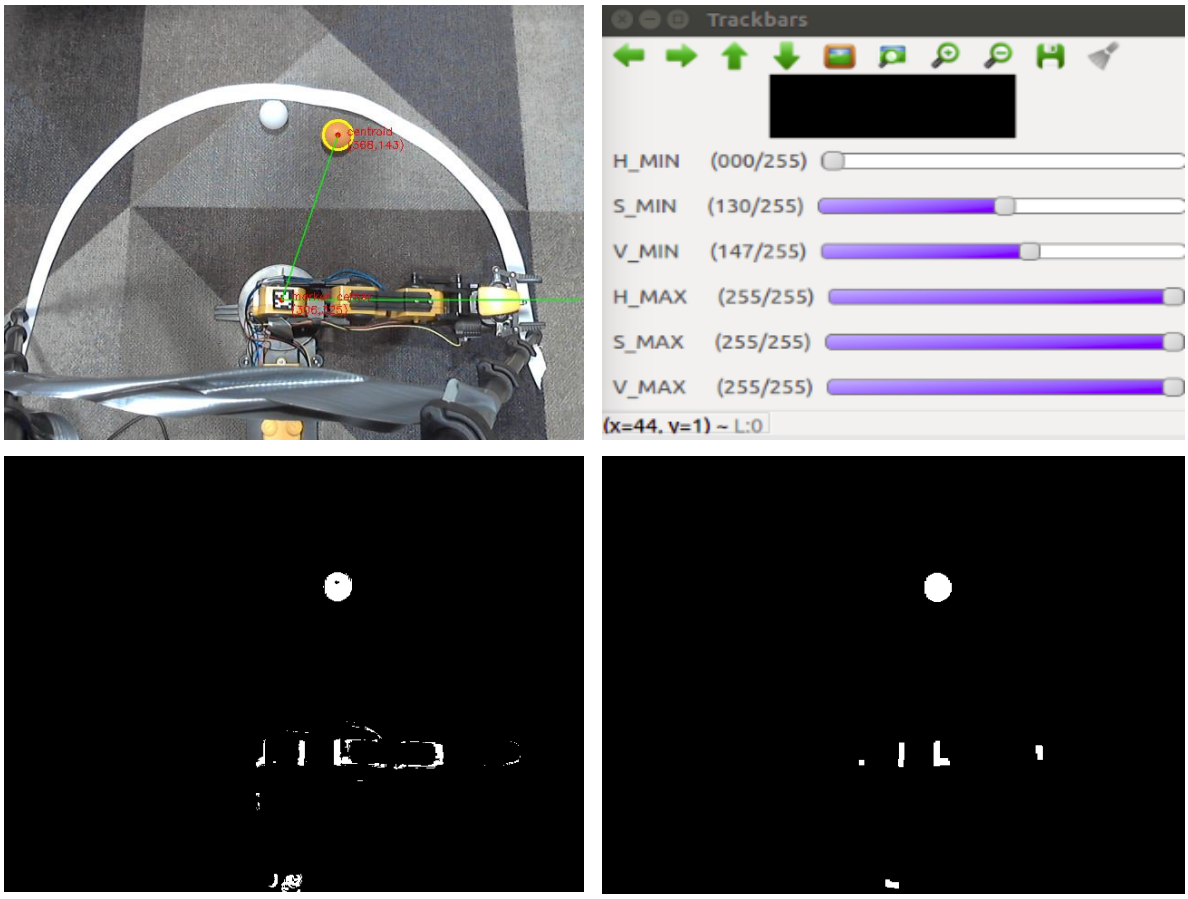


Figure 29: Group of interfaces in orange ball tracking in OpenCV.

5.7 Robotic Arm Movement

The robotic arm is based on Java with USB port to connect with the computer for deploying scripts. However, since the frameworks used in this project were capable of running Python, it was an optimal approach to use a Python-based framework for controlling the robotic arm. For moving the arm, the maximum angular limit was needed to be found through trial-and-error because there are no sensors attached to it. The joint limits of the robot are summarised in Figure 30 [29]. Furthermore, the angular limits were converted to time as the arm had pre-determined functions to work with timed movement. Equation 1 shows the formulae that was used to find the rotation rate in any joint.

$$rotation\ rate = \frac{\big((base\ right\ angle\ limit) - (base\ left\ angle\ limit)\big)}{base\ max\ rotation}$$

Equation 1: Formulae to calculate rotation rate.

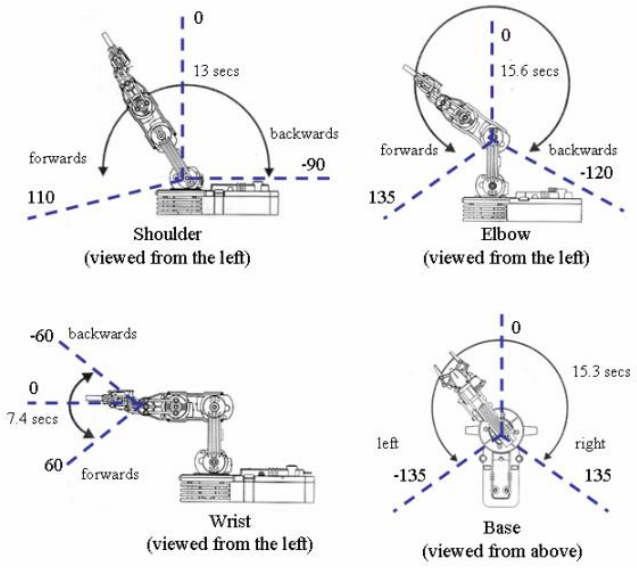


Figure 30: Joint Maximum Rotation Limits and times.

This rate is used to convert the degrees into a rotation time with a user-supplied angle offset (angle between arm and ball). One of the problems that occurred while rotating the arm is that the gear wheels slip, causing the arm to move faster for the first few milliseconds and few more milliseconds to stop moving. This causes inaccuracies in the timed-movement, effectively making the arm unsuccessful at grasping the ball. A solution to this was to add time to the movement between certain angles which can be seen in Figure 43.

5.8 Computer Vision Techniques for Angle Calculation

This stage of the project is a key feature as calculating the angle is needed to know where the ball was in the surrounding region. Angle calculation is useful when there are no sensors available. To implement this feature successfully, two points would need to be found; a fixed point of the arm which does not move regularly and a second point for the ball. These two points would act as coordinates in 2-D space. The first point was calculated with the use of a library called '**Aruco Marker**' [30] which is used to detect certain markers containing information inside them. The second point was simple because it is the midpoint of the detected ball's bounding box. Both points were included in a **trigonometry formula** to calculate degree based on quadrants (as shown in Equation 2).

$$angle = \tan^{-1}\left(\frac{(yAruco_{center} - yBall_{center})}{(xBall_{center} - xAruco_{center})}\right) * \frac{180}{\pi}$$

Equation 2: Formula to calculate angle based on two given points.

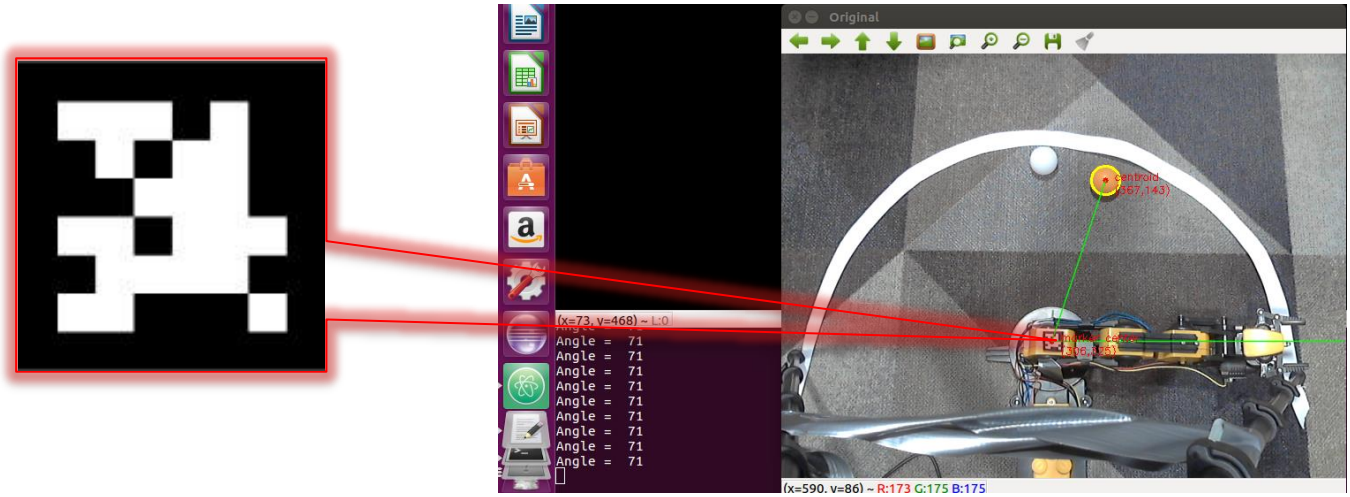


Figure 31: Calculated angle in bottom left corner with the enlargement of the Aruco marker.

5.9 Ball Grasping Algorithm

This algorithm was created to ensure the information flow throughout the system and other main features of the project.

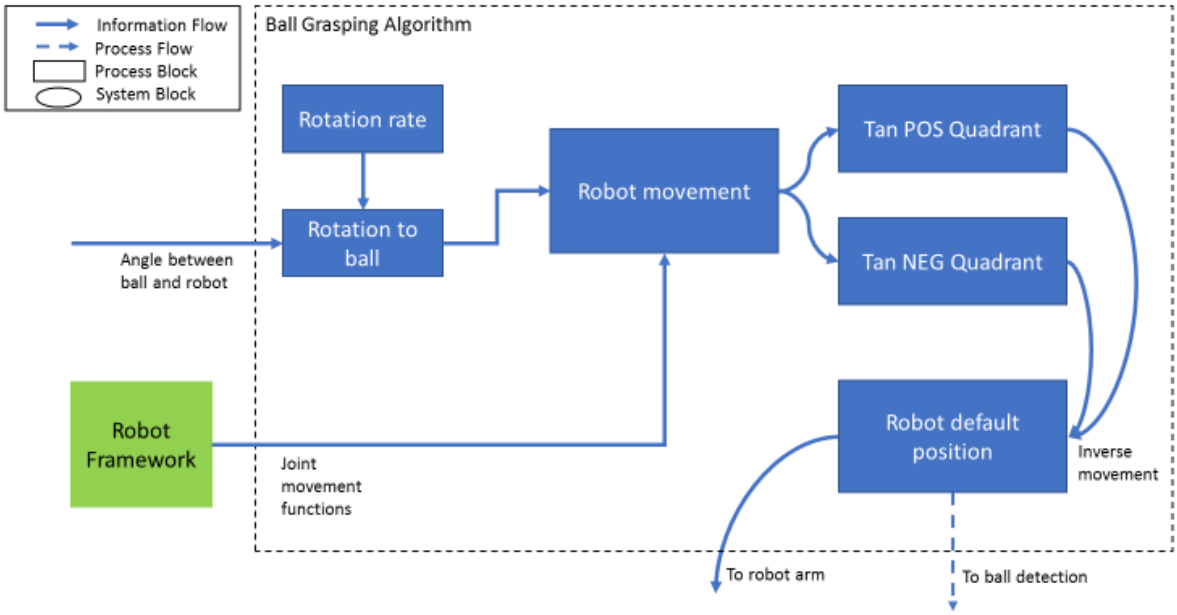


Figure 32: Block diagram of the Ball Grasping Algorithm.

The algorithm consists of certain variables that define the **ROTATION_RATE**; which is calculated from specifying the max rotation limits of the joint of the robot. Then, the angle would enter the scope of the algorithm where it would be passed to the **ROTATION_TO_OBJECT** variable to get the time (in milliseconds) to move towards the ball. Now, a main block of the algorithm takes place hence, movement of the arm. The 'Robot Framework' [20] is a simple and small API which provides all the movement of the robot. "Movements are based upon the **BitPattern class**, and you can feed arbitrary bit patterns to it, but all those the arm is currently capable of are represented above." It is also capable of combining movements with the same time-frame.

From here, there are two different behaviours; tan positive and tan negative based on the quadrants. For each of those quadrants type, the robot arm processes a different movement. For example, For the negative quadrant, it rotates clockwise while for positive angles, it moves counter-clockwise. Next, the default position function of the algorithm ensures the robot returns to the original position. The default position is the inverse of the normal grasping movement hence, it will always return to the default position. However, at some cases, the arm has few flaws; the gear chains are fairly loose in the "servos" it uses for the movement [20]. This causes the arm to move at different speeds. A solution to this was to add extra time to certain quadrant movements by finding the correct time needed through trial-and-error.

Lastly, a video of the entire system operating in similarity and synchronously to the design diagram of this project, can be seen here: https://www.youtube.com/watch?v=eSSsii3

6 Challenges/Problems

This sections briefly illustrates the challenges that were faced throughout the lifespan of this project. These problems are linked to references to locate the solution if needed. Table 3 shows the list of problems along with their solutions.

Problems	References
Undefined reference to cv::VideoCapture()	https://github.com/facebook/C3D/issues/253
Pip install fails on Cython dependency	https://github.com/h5py/h5py/issues/535
Darknet YOLOv2 added to OpenCV officially	https://github.com/pjreddie/darknet/issues/242
Tensorflow 1.4 using CUDA 9	https://github.com/tensorflow/tensorflow/issues/14244
Tensorflow GPU .whl is not supported wheel on this platform	https://github.com/tensorflow/tensorflow/issues/7552
Libcublas.so.9.0 Import error. Cannot open shared object file	https://github.com/tensorflow/tensorflow/issues/15604
DetectNet Kitti format labels in custom dataset (with custom class)	https://github.com/NVIDIA/DIGITS/issues/1561
DetectNet on custom dataset	https://github.com/NVIDIA/DIGITS/issues/1678
Cannot set Caffe path manually	https://github.com/NVIDIA/DIGITS/issues/1199
Build Caffe error	https://github.com/BVLC/caffe/issues/5645
Tensorflow Object detection API on windows – No module named “utils”.	https://github.com/tensorflow/models/issues/3164

Table 3: A table indicating the problems found in the project and their solution links.

7 Evaluation

In this section, I shall describe the evaluation method that was used for evaluating the scope of this project. Then, I explain the reasons for choosing the method by comparing against other methods. Finally, I present the evaluation results gathered and discuss the effectiveness of the results.

7.1 Evaluation Method

For the evaluation stage, I used a method in the branch of Software Testing which is known as 'White-box' testing.

White-box testing [32] is recognised as an efficient method in finding errors/problems. This type of testing is mainly done by the software developer to optimise the code, find hidden errors and to enforce the quality of the software system. A great advantage of this testing is that, "due to required internal knowledge of the software, maximum coverage is obtained" [31]. The reason for choosing this testing method compared to Black-box is because of various algorithms that would be implemented in this project hence, needing to improve the algorithm constantly. The robotic arm should be able to perform its movement smoothly in accordance to the webcam which would process the algorithm to grasp the object. Therefore, White-box testing would help optimise the algorithm and "clean" the code to generate enhanced results. This method significantly improved the functionality of the robot arm.

I carried out the testing phase and gathered data for the chosen testing scope. Next, I performed statistical analysis to evaluate the aim of this project. This analysis stage was critical to identify where the system underperformed in functionality.

7.2 Evaluation Results

The data received from testing indicated that the algorithm was **82%** successful in grasping the ball accurately whereas the **18%** resulted in failure (Figure 33). For testing, I let the robot arm attempt to grasp the ball **100 times** in different locations. Two main phases were tested in relation to the testing scope; **performance and usability**.

7.2.1 Usability

The result was carefully analysed, and it was deduced that the failures occurred due to inaccuracy of time movement (milliseconds). A main problem with this robotic arm is that the battery runs out quite fast which results in inaccurate testing results. Regarding robustness, the robot arm has problems in detecting certain coloured balls in some lighting conditions. For e.g. it would show a low confidence score for a white ball in bright lighting compared to an orange

ball. Additionally, it was found that the ball was more visible to the detection system in dark background. Furthermore, to test the priority of the grasping algorithm, two balls (white and orange) were placed at two different locations to see which ball was prioritised. Surprisingly, the ball with a higher confidence score was grasped first. Lastly, a major usability issue is the 'area of reach' for the robotic arm as it is unable to grasp a ball which is not in a desired region. The arm would still attempt to move and grasp the ball by calculating the angle, however, it will be unsuccessful. In this context, the ball would have to be on the inner-edge of the semi-circle shown in the video in section 5.9. Indeed, this is a usability problem and could only be solved with 'Future Work' and better resources.

7.2.2 Performance

Performance of the robotic arm highly depends on the deep learning model accurately detecting the model and with speed. As mentioned in section 5.5.2, the FPS rates were very low in CPU. Due to a technical problem in GPU, it was decided to use CPU for testing. However, the FPS rate was low resulting in implementing a **'wait' timer of 10 seconds** for the robotic arm. This would help the frames detect a change in movement of ball and update the new location for angle calculation. Other performance issues with the model could be solved with more training data to get higher accuracy for the detection.

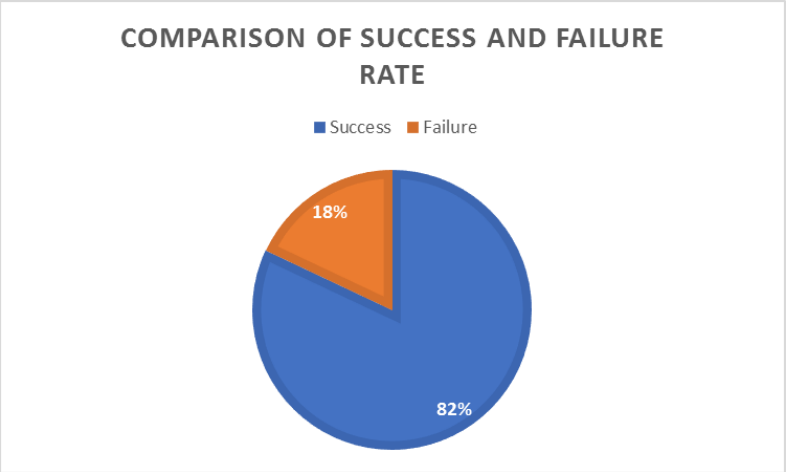


Figure 33: A pie-chart of the success vs failure rate in testing.

8 Conclusion and Future Work

After evaluating the test results, it was evident that the system did not perform that effectively as proposed in the beginning of the project. This was because of some errors likely with the robotic arm which made it incapable of grasping the ping-pong ball at certain locations. The resources and time available for this project made it successful to create the proposed system however, with general errors such as performance and usability. The frameworks that were used in this project gave a variety of conclusion and results. There are two options to choose a framework based on; accuracy or FPS rate. For e.g. one may need higher accuracy for their detection system where Tensorflow would be suitable. On the other hand, YOLOv2 only provides great FPS rate and less accuracy compared to Tensorflow. On the assumption that, both accuracy and speed is required for a system, the 'Future Work' section below provides a solution to this problem. To conclude, the solution provided in this report solves the problem to some extent by creating a system and a 'Ball Grasping Algorithm' to grasp a ball effectively. However, the failures of this problem can be overcome by improving the resources available and by choosing the recommended options in the Future Work section.

8.1 Future Work

In this section, I discuss several possible future research directions.

8.1.1 More effective detection model

For this approach, I could train a neural network with the recently released **YOLOv3** which is extremely fast and accurate compared to YOLOv2. Furthermore, it provides a unique way to adjust tradeoff between accuracy and speed by changing the size of the model without any retraining required.

8.1.2 Using Kalman Filter for prediction of moving ball

Kalman Filter is used to predict the trajectory of moving ball hence, this algorithm would enhance the system in this project. The scope of this problem can be adjusted to work for moving ball rather than stationary and while using a robotic arm that has a faster movement rate.

8.1.3 Inverse and Forward Kinematics

Inverse Kinematics requires the robot arm to use its gripper to a position in 3-D space. This would allow the arm to have a certain logic containing its movement limits where it would adjust itself based on the location of the ball.

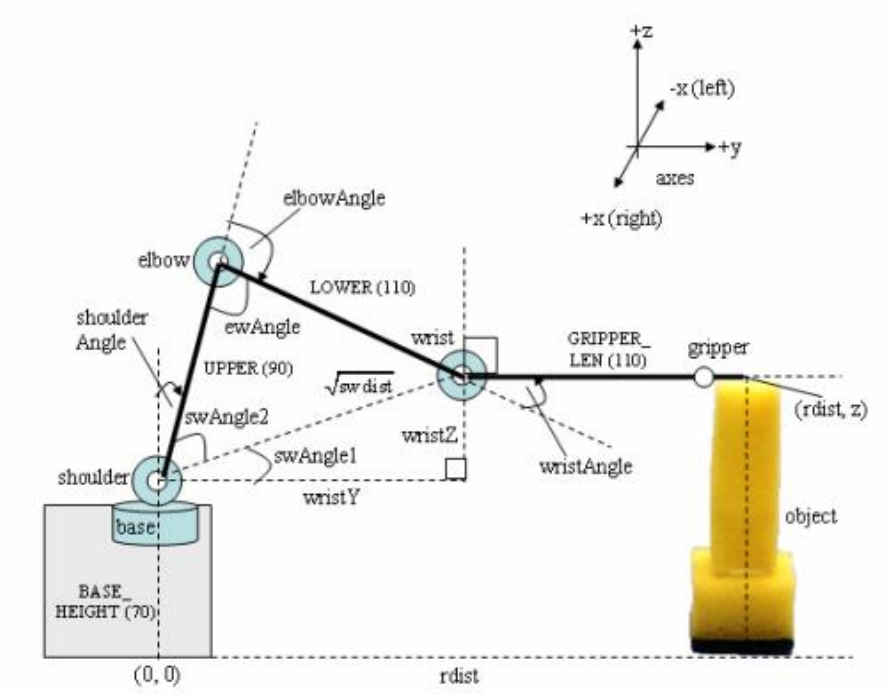


Figure 34: Inverse Kinematics for the Robotic Arm [29].

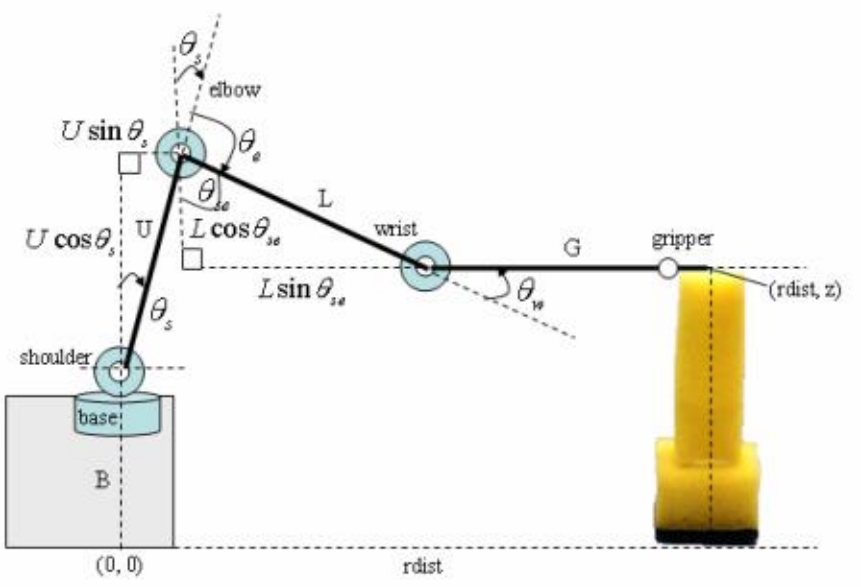


Figure 35: Forward Kinematics calculations for the robotic arm [29].

Based on the above calculations, the arm would be able to adjust its movement where the ball is hard to reach. Currently, the ball can only be placed at certain distance to the robot which cannot use other joints except for base rotation movement. This causes issues when the ball is not in the desired position. Therefore, this approach would help it to rotate any of its joints to by knowing the current position of each joint to stay within limit and grasp the ball successfully.

References

[1] Robotshelpingkids.yale.edu. (2017). Overview | Socially Assistive Robotics. [online] Available at: http://robotshelpingkids.yale.edu/overview [Accessed 27 Oct. 2017].

[2] Perrin, S. (2017). Ultra-fast, the robotic arm can catch objects on the fly. [online] Actu.epfl.ch. Available at: https://actu.epfl.ch/news/ultra-fast-the-robotic-arm-can-catch-objects-on-th/ [Accessed 27 Oct. 2017].

[3] Kim, S., Shukla, A. and Billard, A. (2014). Catching Objects in Flight. IEEE Transactions on Robotics, 30(5), pp.1049-1065.

[4] IEEE Spectrum: Technology, Engineering, and Science News. (2017). Fast Robot Arm Catches Flying Objects. [online] Available at: https://spectrum.ieee.org/automaton/robotics/robotics-hardware/epfl-fast-robot-arm-catches-flying-objects [Accessed 27 Oct. 2017].

[5] VersionOne. (2017). Agile Methodologies for Software Development. [online] Available at: https://www.versionone.com/agile-101/agile-methodologies/ [Accessed 27 Oct. 2017].

[6] Segue Technologies. (2017). Waterfall vs. Agile: Which Methodology is Right for Your Project? [online] Available at: https://www.seguetech.com/waterfall-vs-agile-methodology/ [Accessed 27 Oct. 2017].

[7] Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J. and Quillen, D. (2017). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. The International Journal of Robotics Research, p.027836491771031.

[8] Brownlee, J. (2018). How Much Training Data is Required for Machine Learning? - Machine Learning Mastery. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/much-training-data-required-machine-learning/ [Accessed 12 Mar. 2018].

[9] Cvlibs.net. (2018). The KITTI Vision Benchmark Suite. [online] Available at: http://www.cvlibs.net/datasets/kitti/ [Accessed 12 Mar. 2018].

[10] Barker, J., Prasanna, S., Barker, J., Gray, A., Tao, A., Barker, J., Sarathy, S. and Heinrich, G. (2018). Deep Learning for Object Detection with DIGITS. [online] NVIDIA Developer Blog. Available at: https://devblogs.nvidia.com/deep-learning-object-detection-digits/ [Accessed 13 Mar. 2018].

[11] GitHub. (2018). NVIDIA/DIGITS. [online] Available at: https://github.com/NVIDIA/DIGITS/tree/master/digits/extensions/data/objectDetection [Accessed 13 Mar. 2018].

[12] Image-net.org. (2018). ImageNet. [online] Available at: http://www.image-net.org/download-bboxes [Accessed 14 Mar. 2018].

[13] Host.robots.ox.ac.uk. (2018). The PASCAL Visual Object Classes Homepage. [online] Available at: http://host.robots.ox.ac.uk/pascal/VOC/ [Accessed 17 Mar. 2018].

[14] TensorFlow. (2018). Importing Data | TensorFlow. [online] Available at: https://www.tensorflow.org/programmers_guide/datasets [Accessed 14 Mar. 2018].

[15] Caffe.berkeleyvision.org. (2018). Caffe | Deep Learning Framework. [online] Available at: http://caffe.berkeleyvision.org/ [Accessed 15 Mar. 2018].

[16] Tao, A., Barker, J., Sarathy, S., Barker, J., Gray, A., Barker, J., Prasanna, S., Heinrich, G. and Gray, A. (2018). DetectNet: Deep Neural Network for Object Detection in DIGITS. [online] NVIDIA Developer Blog. Available at: https://devblogs.nvidia.com/detectnet-deep-neural-network-object-detection-digits/ [Accessed 15 Mar. 2018].

[17] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. arXiv preprint arXiv:1612.08242, 2016

[18] GitHub. (2018). thtrieu/darkflow. [online] Available at: https://github.com/thtrieu/darkflow [Accessed 17 Mar. 2018].

[19] GitHub. (2018). tzutalin/labelImg. [online] Available at: https://github.com/tzutalin/labelImg [Accessed 4 Apr. 2018].

[20] GitHub. (2018). orionrobots/robot_arm. [online] Available at: https://github.com/orionrobots/robot_arm [Accessed 4 Apr. 2018].

[21] NVIDIA. (2018). NVIDIA Jetson Modules and Developer Kits for Embedded Systems Development. [online] Available at: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/ [Accessed 18 Mar. 2018].

[22] Medium. (2018). Speed/accuracy trade-offs for modern convolutional object detectors. [online] Available at: https://medium.com/@phelixlau/speed-accuracy-trade-offs-for-modern-convolutional-object-detectors-bbad4e4e0718 [Accessed 19 Mar. 2018].

[23] Towards Data Science. (2018). Deep Learning for Object Detection: A Comprehensive Review. [online] Available at: https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9 [Accessed 23 Mar. 2018].

[24] Stats and Bots. (2018). Improving Real-Time Object Detection with YOLO – Stats and Bots. [online] Available at: https://blog.statsbot.co/real-time-object-detection-yolo-cd348527b9b7 [Accessed 23 Mar. 2018].

[25] Redmon, J. (2018). YOLO: Real-Time Object Detection. [online] Pjreddie.com. Available at: https://pjreddie.com/darknet/yolo/ [Accessed 23 Mar. 2018].

[26] GitHub. (2018). thtrieu/darkflow. [online] Available at: https://github.com/thtrieu/darkflow#training-on-your-own-dataset [Accessed 23 Mar. 2018].

[27] Medium. (2018). Data Augmentation Techniques in CNN using Tensorflow. [online] Available at: https://medium.com/ymedialabs-innovation/data-augmentation-techniques-in-cnn-using-tensorflow-371ae43d5be9 [Accessed 29 Mar. 2018].

[28] Raj, N. (2018). Why every TensorFlow developer should know about TFRecord! - Skcript. [online] Why every TensorFlow developer should know about TFRecord! - Skcript. Available at: https://www.skcript.com/svr/why-every-tensorflow-developer-should-know-about-tfrecord/ [Accessed 30 Mar. 2018].

[29] Fivedots.coe.psu.ac.th. (2018). [online] Available at: https://fivedots.coe.psu.ac.th/~ad/jg/nui06/robotArm.pdf [Accessed 2 Apr. 2018].

[30] Docs.opencv.org. (2018). OpenCV: Detection of ArUco Markers. [online] Available at: https://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html [Accessed 2 Apr. 2018].

[31] Farcic, V. (2017). Black-box vs White-box Testing. [online] Technology Conversations. Available at: https://technologyconversations.com/2013/12/11/black-box-vs-white-box-testing/ [Accessed 27 Oct. 2017].

[32] Invensis Blog. (2018). What is White Box Software Testing: Advantages and Disadvantages. [online] Available at: https://www.invensis.net/blog/it/white-box-software-testing-advantages-disadvantages/ [Accessed 3 Apr. 2018].

[33] Nielsen, M. (2018). Neural Networks and Deep Learning. [online] Neuralnetworksanddeeplearning.com. Available at: http://neuralnetworksanddeeplearning.com/chap6.html [Accessed 5 Apr. 2018].

[34] Wang, Zhenyi, "Automatic Brain Tumor Segmentation by Deep Convolutional Networks and Graph Cuts" (2018). Electronic Thesis and Dissertation Repository. 5189. https://ir.lib.uwo.ca/etd/5189

[35] Alves, R., Depth, R., Brad Templeton, R., Szollosy, M., Tobe, F., Nash, A., Depth, R., Tobe, F., News, M., Zimmermann, T. and Brad Templeton, R. (2018). Smart Grasping System available on ROS Development Studio | Robohub. [online] Robohub.org. Available at: http://robohub.org/smart-grasping-system-available-on-ros-development-studio/ [Accessed 5 Apr. 2018].

[36] IEEE Spectrum: Technology, Engineering, and Science News. (2017). How Google Wants to Solve Robotic Grasping by Letting Robots Learn for Themselves. [online] Available at: https://spectrum.ieee.org/automaton/robotics/artificial intelligence/google-large-scale robotic-grasping-project [Accessed 27 Oct. 2017].

9 Personal Reflection

9.1 Reflection on Project

After viewing back on the project, I would have implemented the system with a faster robotic arm that is capable of outputting sensory data to track the movement of the arm. Furthermore, I would use a GPU for testing and the YOLOv3 method for even faster detection and higher accuracy.

9.2 Personal Reflection

If I had my time again, I would have fixated on a training dataset and gathering more data for the category, hence, not spending long time to gather the dataset, allowing me to spend more time on the implementation. This would also allow for a higher accuracy in training. Then, I would test the system against human participants with the 'Black-box' testing method to gather better evaluation results for analysis.

Appendices

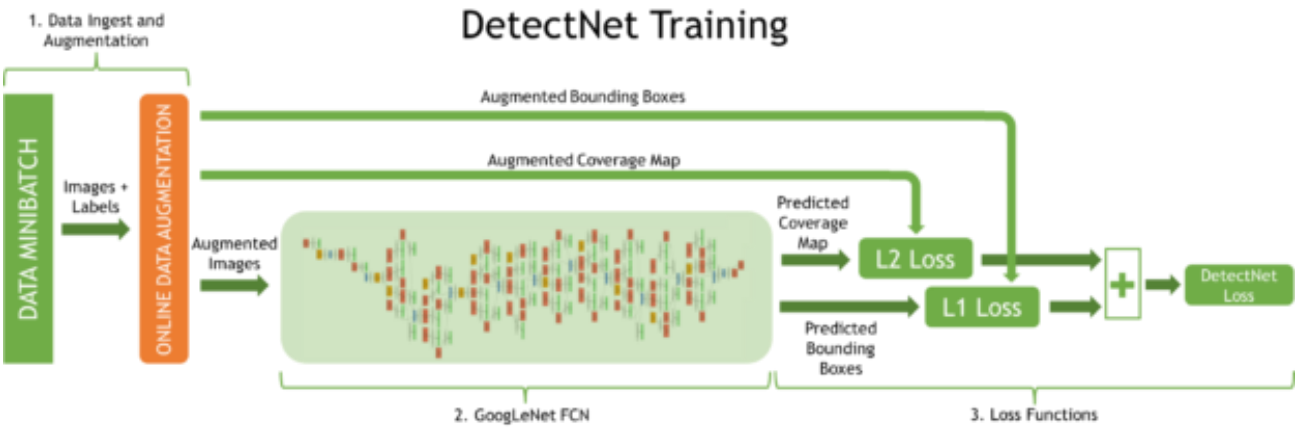


Figure 36: The DetectNet structure for training.

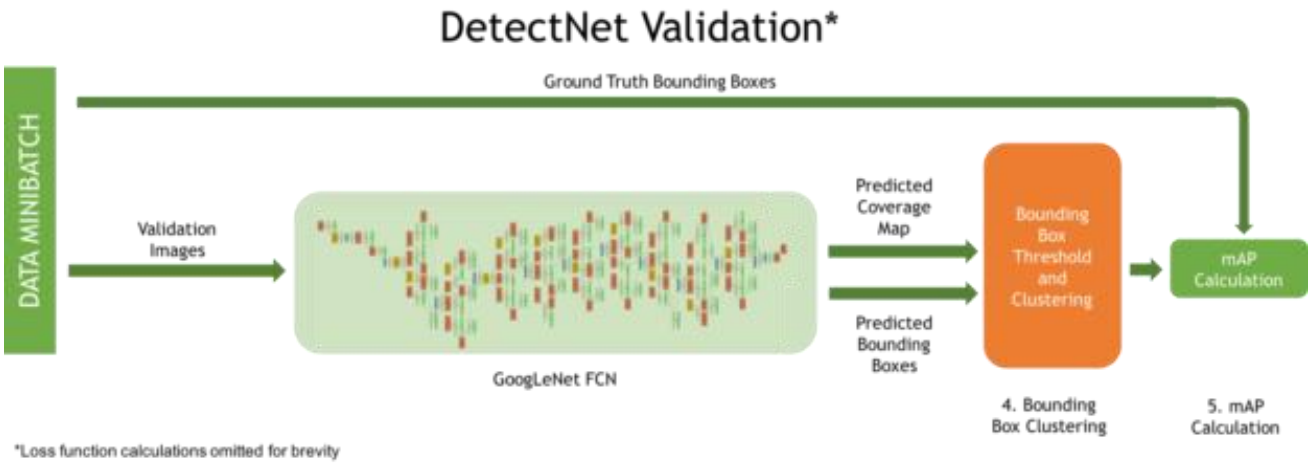


Figure 37: The DetectNet structure for validation.

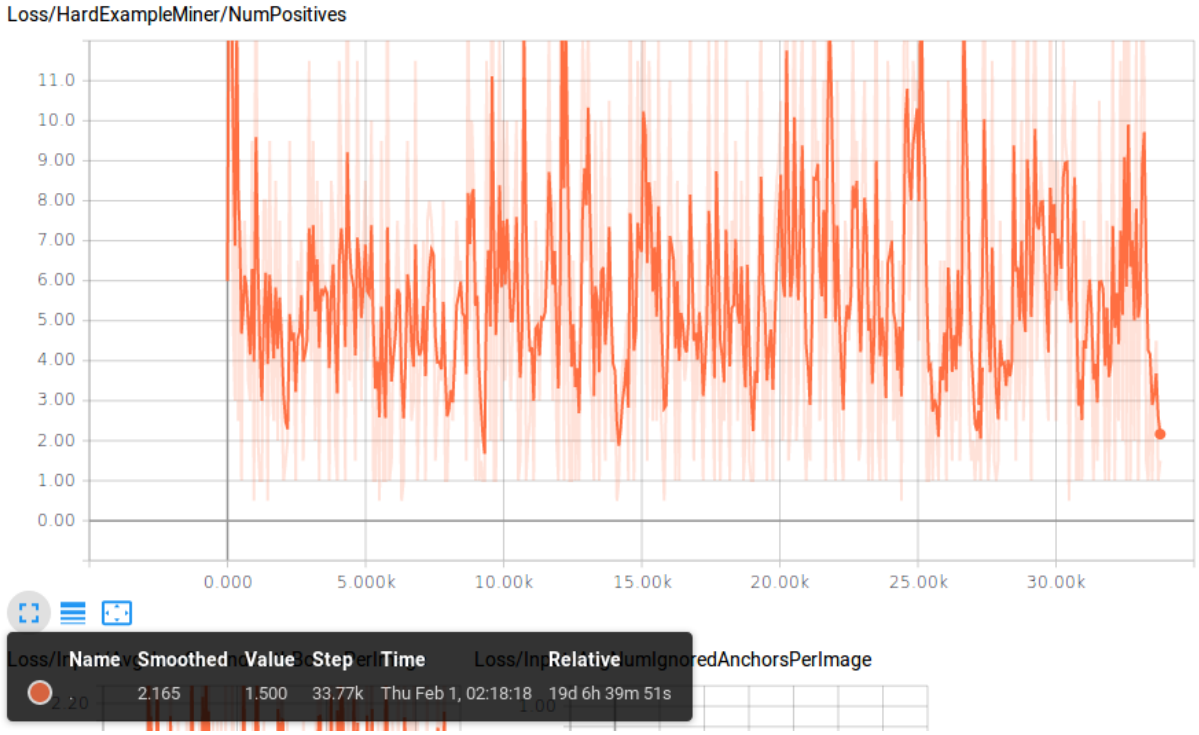


Figure 38: Loss of No. of Positives graph of the trained model on Tensorflow.

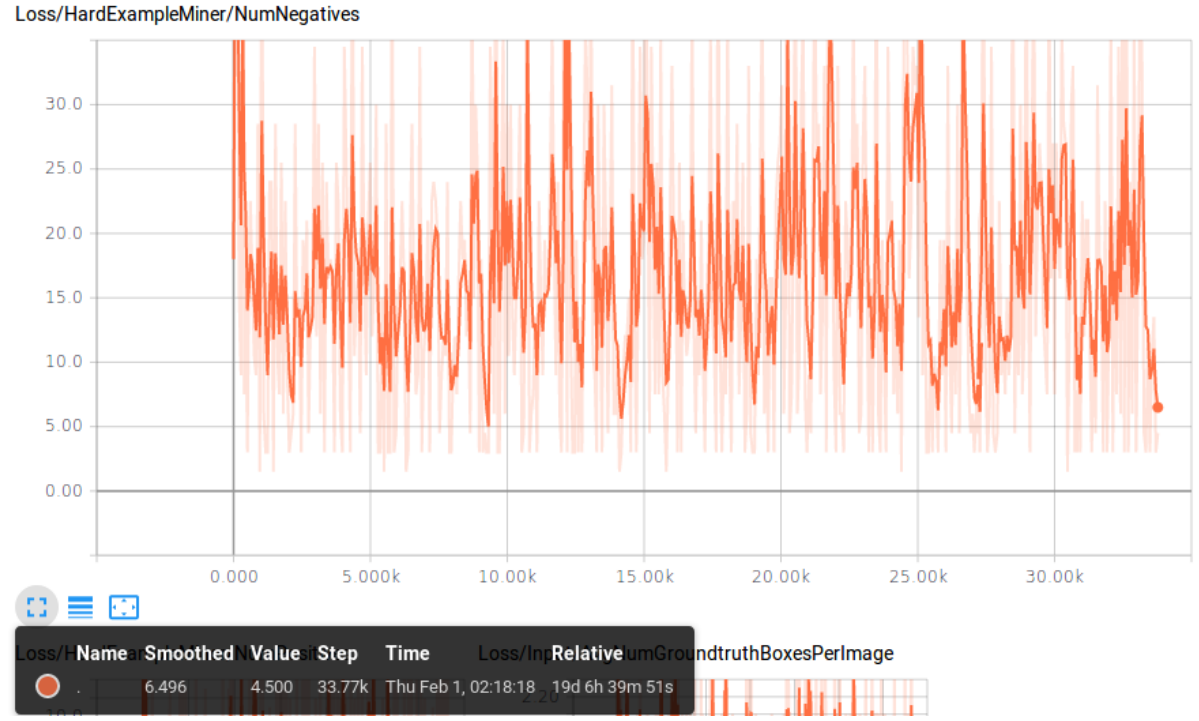


Figure 39: Loss of No. of Negatives graph of the trained model on Tensorflow.

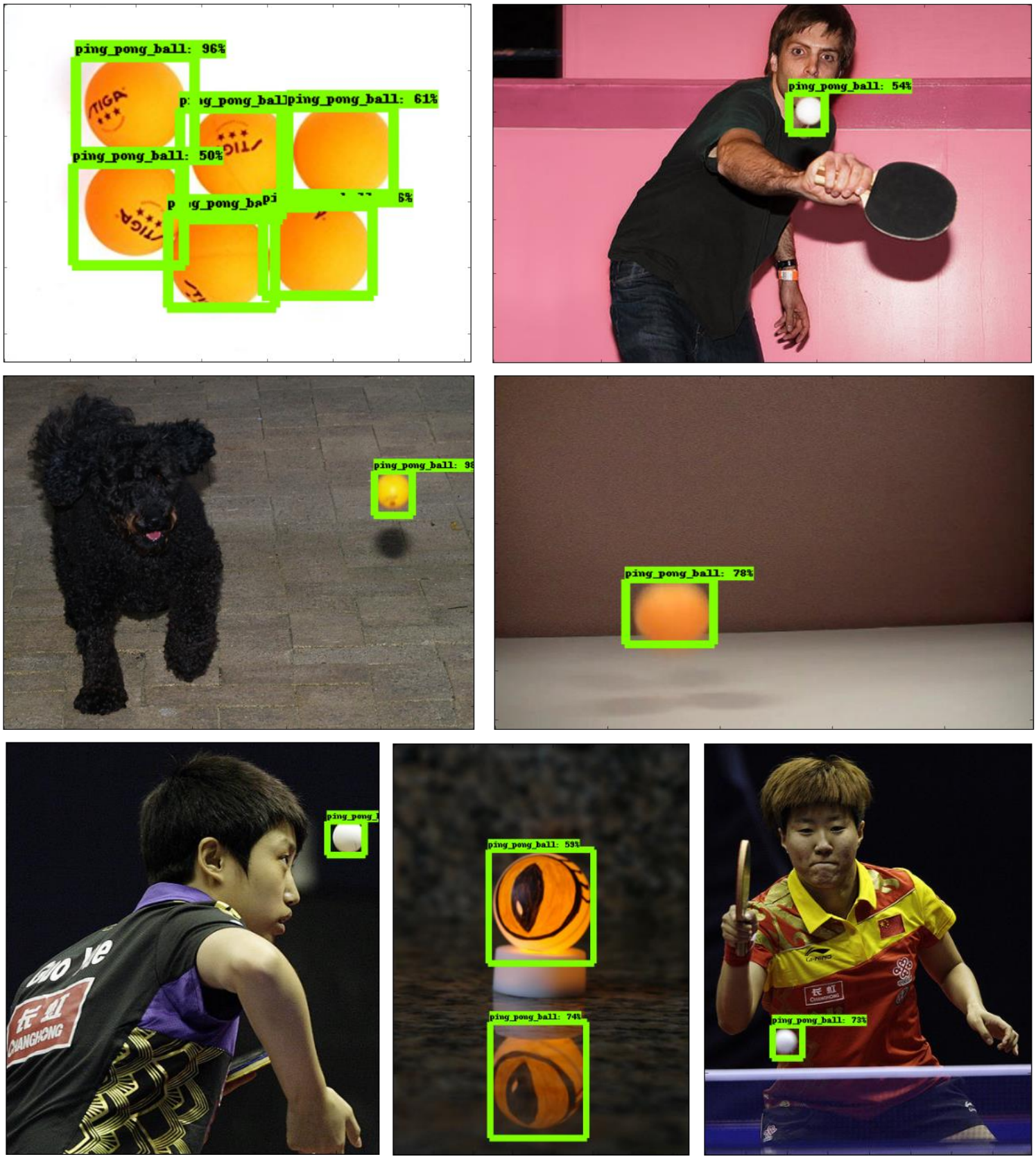


Figure 40: Sample test images where ping-pong balls were detected using Tensorflow.

```
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)

# In[34]:

def load_image_into_numpy_array(image):
  (im_width, im_height) = image.size
  return np.array(image.getdata()).reshape(
      (im_height, im_width, 3)).astype(np.uint8)


# # Detection

# If you want to test the code with your images, just add path to the images to the TEST_IMAGE_PATHS.
PATH_TO_TEST_IMAGES_DIR = 'test_images'
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image{}.jpg'.format(i)) for i in range(21, 33) ]

# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8)


# In[36]:


with detection_graph.as_default():
  with tf.Session(graph=detection_graph) as sess:
    # Definite input and output Tensors for detection_graph
    image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
    # Each box represents a part of the image where a particular object was detected.
    detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
    # Each score represent how level of confidence for each of the objects.
    # Score is shown on the result image, together with the class label.
    detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
    detection_classes = detection_graph.get_tensor_by_name('detection_classes:0')
    num_detections = detection_graph.get_tensor_by_name('num_detections:0')
    while True:
      ret, image_np = cap.read()
      # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
      image_np_expanded = np.expand_dims(image_np, axis=0)
      # Actual detection.
      (boxes, scores, classes, num) = sess.run(
          [detection_boxes, detection_scores, detection_classes, num_detections],
          feed_dict={image_tensor: image_np_expanded})
      # Visualization of the results of a detection.
      vis_util.visualize_boxes_and_labels_on_image_array(
          image_np,
          np.squeeze(boxes),
          np.squeeze(classes).astype(np.int32),
          np.squeeze(scores),
          category_index,
          use_normalized_coordinates=True,
          line_thickness=8)

      cv2.imshow('Ping-Pong Ball Detection', cv2.resize(image_np, (640, 480)))
      if (cv2.waitKey(25) & 0xFF == ord('q')):
          cv2.destroyAllWindows()
          break
```

Figure 41: The full code for object detection in web-cam through Tensorflow.

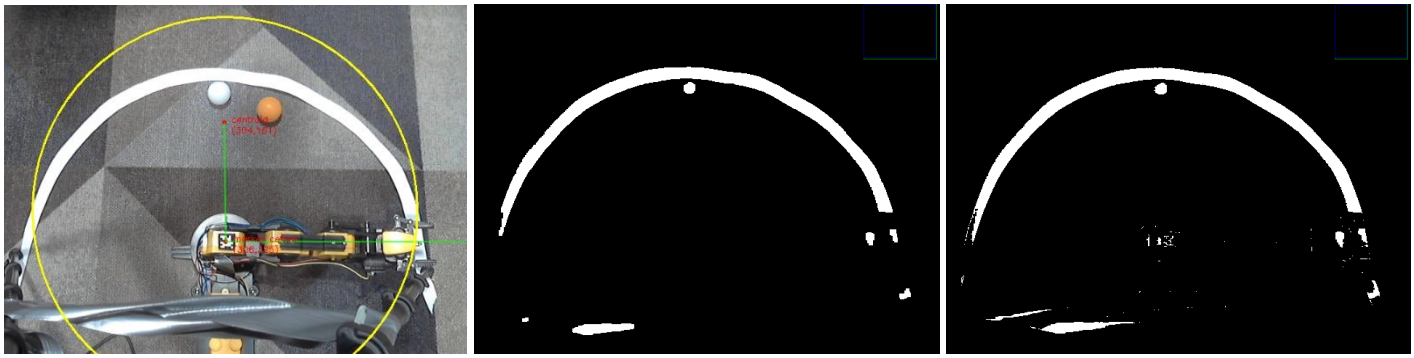


Figure 42: Original, Mask and Threshold interfaces for attempt of white-ball tracking.

```
    def move_robot_arm(self, arm, angle):

        ROTATION_RATE = ((BASE_RIGHT_ANGLE_LIMIT) - (BASE_LEFT_ANGLE_LIMIT)) / BASE_ROTATION_MAX

        ROTATION_TO_OBJECT = angle / ROTATION_RATE

        #ROTATION_TO_OBJECT_ROUNDED = round(ROTATION_TO_OBJECT, 2)

        # Wait 4 seconds before issuing a command for moving the ball around and testing new angles
        #time.sleep(4) #1.52 before
        if (0 <= angle <= 90):
            self.arm.move(usb_arm.BaseCtrClockWise, ROTATION_TO_OBJECT + 2.5)
            self.arm.move(usb_arm.ElbowDown, 1.55)
            self.arm.move(usb_arm.GripsClose, 0.6)
        if (91 <= angle <= 180):
            self.arm.move(usb_arm.BaseCtrClockWise, ROTATION_TO_OBJECT + 3.5)
            self.arm.move(usb_arm.ElbowDown, 1.55)
            self.arm.move(usb_arm.GripsClose, 0.6)
        if (-1 >= angle >= -45):
            NEW_ANGLE = abs(angle)
            ROTATION_TO_OBJECT = NEW_ANGLE / ROTATION_RATE
            self.arm.move(usb_arm.BaseClockWise, ROTATION_TO_OBJECT)
            self.arm.move(usb_arm.ElbowDown, 1.55)
            self.arm.move(usb_arm.GripsClose, 0.6)
        if (-135 >= angle >= -179):
            NEW_ANGLE = (180 - abs(angle)) + 180
            ROTATION_TO_OBJECT = NEW_ANGLE / ROTATION_RATE
            self.arm.move(usb_arm.BaseCtrClockWise, ROTATION_TO_OBJECT + 3.5)
            self.arm.move(usb_arm.ElbowDown, 1.55)
            self.arm.move(usb_arm.GripsClose, 0.6)
        if (-46 >= angle >= -134):
            print "Angle is out of range. Plese place the ball inside the range."
```

Figure 43: Snippet of code for movement of robot arm.

```
        v1_min, v2_min, v3_min, v1_max, v2_max, v3_max = get_trackbar_values(range_filter)

        # v1_min = 0
        # v2_min = 130
        # v3_min = 147
        # v1_max = 255
        # v2_max = 255
        # v3_max = 255

        thresh = cv2.inRange(frame_to_thresh, (v1_min, v2_min, v3_min), (v1_max, v2_max, v3_max))

        kernel = np.ones((5,5),np.uint8)
        mask = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)
        mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

        # find contours in the mask and initialize the current
        # (x, y) center of the ball
        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)[-2]
        center = None

        # only proceed if at least one contour was found
        if len(cnts) > 0:
            # find the largest contour in the mask, then use
            # it to compute the minimum enclosing circle and
            # centroid
            c = max(cnts, key=cv2.contourArea)
            ((x, y), radius) = cv2.minEnclosingCircle(c)
            M = cv2.moments(c)
            center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
            if (radius > 15) and ((x_center and y_center) != 0):
                # draw the circle and centroid on the frame,
                # then update the list of tracked points
                cv2.circle(image, (int(x), int(y)), int(radius),(0, 255, 255), 2)
                cv2.circle(image, center, 3, (0, 0, 255), -1)
                cv2.line(image,(aruco_center[0], aruco_center[1]), (center[0], center[1]),(0, 255, 0), 1, cv2.LINE_AA)
                cv2.putText(image,"centroid", (center[0]+10,center[1]), cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 0, 255),1)
                cv2.putText(image,"("+str(center[0])+","+str(center[1])+")", (center[0]+10,center[1]+15), cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0, 0, 255),1)

            angle = int(math.atan2((y_center - center[1]), (center[0] - x_center)) * 180 // math.pi)
            print "Angle = ", angle
```

Figure 44: Snippet of code for ball tracking in OpenCV.