Friday, 7th May 2020

Position Location System

BEng Individual Project Report

Will Prior

Student Number: 179178337

Supervisor: Dr Stephen Pennock

Final Year

~11250 Words

1 Abstract

Over 25 years of GPS has shown it to be an incredibly useful tool allowing devices to locate themselves within metres in outdoor environments. However due to the scattering of the signals by the walls and other solid surfaces, locating devices inside of buildings or in urban areas using this technique is far from reliable. This has led to the increase in research into indoor positioning systems which use a range of technologies such as Wi-Fi, visible light, and acoustic methods to name a few. This paper will outline the research and prototyping of an optical imaging based self-positioning system which can be used in applications such as on robots for its navigation and tracking.

Initial research was placed into alternative systems to assess what had already been achieved and what is possible before concentrating on understanding optical methods and how they could be implemented. The basic concept of the system was to use a reference point detection system with a trilateration or triangulation technique to find the final location of the system. The research showed that a deep learning approach and a geometric circle intersection triangulation method was of most suitable choice due to their practicality with the resources available, and their accuracy in reference point and device positioning, respectively.

An overview of the system consists of a raspberry pi, a pi camera, custom and opensource software and the numbers 1 to 6 each written individually on 6 A4 pieces of paper used as reference points. The pi camera is directed at the ceiling with at least three reference points in its angle of view. Using a combination of TensorFlow's object detection API and OpenCV's image processing tools the device is then able to locate three reference points in each video frame using a bounding box round each. The reference point location information is then used to locate both, the angle of the reference points from the horizontal direction the device is facing and make an estimate of the distance covered on the ceiling in the images horizontal. These respective pieces of information are then used in a triangulation and scale factor equation to gather the X, Y and Z coordinates of the device in the room it is situated.

The reference point detection system uses a premade supervised deep learning object detection model produced by TensorFlow. To be able to detect the custom written reference points the model required training – which involves affectively teaching the model what it needs to look for from showing it a set of labelled images. After trying a range of premade datasets to train the model with no success, a custom one was built. This consisted of a large set of manually labelled images of the reference points in different positions on the ceiling. The result of training the model was very successful with it being able to distinguish the reference points on the ceiling from a range of distances and orientations of the camera.

The testing completed on the complete system consisted of the device being moved along a variety of routes in the rooms horizontal X, Y plane, analysing the effect of changing orientation and location under different sets of reference points. This was followed by some vertical depth tests – where the device was placed at a range of heights in three orientations to see how well the z coordinate could estimate its position. The final test carried out was a system performance test, assessing the speed of the system when the device was both stationary and moving.

The outcome showed the system to have good level of accuracy with position metrics produced having an average error of just under ±10cm. The system speed shows the results to be output approximately every two seconds, showing there to be improvements to be made in this area to bring it up to real-time reporting, suggestions of which are highlighted throughout the discussion and future work sections.

Contents

2 Introduction

From as early as the 1960's technologies have been in development to be able to gain the position of objects and people for uses such as navigation and tracking. With the Global Positioning System (GPS) being introduced in 1990 the ability to be able to track an outdoor object with an accuracy of 1-10 meters was introduced. By 1995, GPS was fully functional, consisting of space-based satellites which provide location and time information in all weather conditions anywhere on earth. However, as GPS uses trilateration, three or more satellites need to be always in its line of sight, otherwise the system would fail or encroach error [1]. The accuracy of the location given by the system can be increased with number of satellites. Although, even with these accuracy improvements, there became new challenges when positions of finer degrees than metres are required. This issue is exacerbated when in urban and indoor areas as the concrete walls cause attenuation of the signals, thus they cannot pass through them to reach the receiver in question. At its best some very sensitive GPS chips can occasionally gain signal of very low accuracy indoors, showing this technique to be very unreliable to locate objects in enclosed environments [2] . This prompted further research into indoor positioning systems (IPS).

With increasing numbers of people spending more time indoors due to technological advancements, indoor positioning equipment has been in larger demand. In 2020, the global Indoor positioning and navigation market was valued at $6.92 billion and is projected to grow to $23.6 billion by 2025, with the USA holding 40% of the current market value and Asia having the largest projected percentage growth. This shows it to be a technology that is being used heavily in the current and will be in future times, with large companies such as Google, Qualcomm and Microsoft developing technology in this field [3]. Some examples of current applications are in factories to avoid accidents and save time in logistical processes, in customer navigation around Airports and Railway Stations, in robotics, and in market research. The latter involving analysis of advertisement positioning and customer interaction to find the optimal solution. Or to track the behaviour of customers in shops to optimise the shelf layout to encourage purchasing. [4]

There have been many variations in the way these systems have been achieved using a range of mediums such visible light, acoustic, Wi-Fi, and local RF systems to name just a few. These gather the initial information such as initial distances or angle data, which in most cases is used in a triangulation, or trilateration algorithm to gain the devices final location.

2.1 Aims and Objectives

The main aim of this project is to investigate and design a robot-based indoor self-positioning system using optical imaging principles. This idea was broken down into a further set of aims which consisted of it being accurate to within ±20cm of error, compact, inexpensive, and easy to install into a given environment. From the hardware available the obvious choice was the raspberry pi and pi camera with their size, price and processing power making them perfect for the project.

The knowledge of the camera being used meant that a clear set of successive objectives could be set for the remainder of the project as seen below:

- Assess reference point detection techniques and implement the most appropriate.
- Process the data recorded from the detection technique to obtain the desired input information for the position location calculations.
- Use the processed data to locate the robot's position.
- Display the location data in an easy to consume format.

3 Past Work

3.1 Alternative Systems

Although Optical imaging principles will be used in this project it is important to assess other methods in which the system could be implemented to understand can be achieved.

3.1.1 Wi-Fi Based System

Wi-Fi positioning systems make use of Wi-Fi transmitters as tags that send packets to several Wi-Fi access points in the local environment. The receivers report the time and signal strength of the reading to a backend service which uses a set of algorithms to compute the position, with the final information usually sent to the cloud. This system can give a degree of accuracy between three to five meters if the system is using TDOA (time difference of arrival) and there are at least three access points that have received the tag transmission signal. [5]

3.1.2 Ultra-Wide-Band System

Ultra-Wide-Band (UWB) Systems use a network of UWB readers and UWB tags to locate themselves. The readers transmit a very wide pulse over a GHz spectrum and then listen for a 'chirps' from the tags. These chirps are very wide instantaneous coded burst signals allowing for highly accurate time measurement to be recorded by the readers, these values are subsequently used to calculate the distance of the tags from the readers. Due to the extreme width of the signal used in these systems the accuracy of the location information given is one of the best on the current market. [5]

3.1.3 Acoustic System

Acoustic systems work in a very similar way to that of the UWB system but instead of using radio signals it uses ultrasonic pulses which are out of the range of human hearing. The tags emit the pulses and ultrasonic receivers in the room pick up the sounds and can locate the distance between them and the tags. These distances are then used in a trilateration technique, which will be explained in section 3.3, to locate the tags. The distances input being calculated by a time of flight (TOF), TDOA or other technique which uses the knowledge of the speed of sound in air and the time taken for the sound to be received. [6]

From evaluating these systems many of them have their advantages such as the high accuracy of the Acoustic and UWB systems. However, all three systems mentioned require two-way communication between a tag and a reader, increasing the complexity of the design of each. This also means there would be an added cost as you require electronic hardware and software on both the reader and tag sides of the system. As the main aims of the project are to create an affordable and compact system, the extra hardware and cost would make these methods less desirable. On the other hand an optical imaging system being able to use its own vision, only requires one sensor and one microcontroller at its minimum, with its reference points being physical objects or cheaply implementable symbols. This creates a practical, simple, and low-cost system with no transceiver-to-transceiver communication latency, giving the potential of producing a faster more efficient device of similar accuracy.

3.2 Reference Point Detection

To be able to locate itself the device needs to be able to identify and know the position of some reference points around it. As optimal imaging is being used, research has been carried out into image/object detection algorithms to locate and classify the reference points within a video frame.

3.2.1 Deep Learning Based Detection

With the more recent developments in deep learning there have been some highly accurate and easy to implement object detection models created using neural networks such as R-CCN, Faster R-CCN, SSD and YOLO to name a few. These models can be categorised by the number of stages they use to perform the image detection. In the one stage approach such as YOLO and SSD the classification and regression of the image is done in a single shot using regular and dense sampling with respect to location scales and aspect ratios. On the other hand in the two-stage approach, which is applied in the R-CCN and Faster R-CCN models, the models work with the first stage being the region proposal of the object in the image and the second stage being the classification and the addition of the bounding boxes to the object's location. The single stage approach models are very quick at detecting objects in the images at the expense of a reduced degree of accuracy in comparison to a two-stage model. [7] But this quick lower processing time and power makes them far more suitable for smaller processors such as those found in phones or micro-controllers.

3.2.2 Heuristic Based Detection

Deep learning is not the only way that image detection can be carried out, with some well-known heuristic approaches also being an option. For example, a naïve approach to attempting something of this nature could be a sliding window approach, where you could take a smaller image of the object you are looking for and slide it over the image in question and try to find any close matches [8]. Or Another approach could be to locate objects using their colour or if they are within a given colour range [9]. Both the above could be expanded to involve more and more conditions until the detector finds what is required. Using methods such as this have their advantage such as far less computing power required then any deep learning algorithm and the fact that they do not need to be trained reducing the time for their implementation. However, heuristic approaches tend to have lower accuracy and are less reactive to changes in their environment. Therefore they can be easily affected by the changes in size of the object required and the lighting in the image.

Considering the previous, the deep learning model seems to be the most affective choice for the reference point detection. This is because instrument accuracy and reliability are incredibly important factors in a location system especially when the degree of accuracy is a main objective. The model that will be used will be single stage to allow it to be implemented on a micro-controller without having to compromise on the speed of the output.

3.3 Calculating Location Coordinates

3.3.1 X, Y Coordinates

If distances were easily calculated from the reference point detection information, then trilateration would be used. As mentioned previously this is a technique employed by GPS, in which the distance between at least three reference points and the device is measured. A sphere with radius of its measured distance is then created around each reference point with the intersection of these spheres said to be the final location of the device in question. [10]

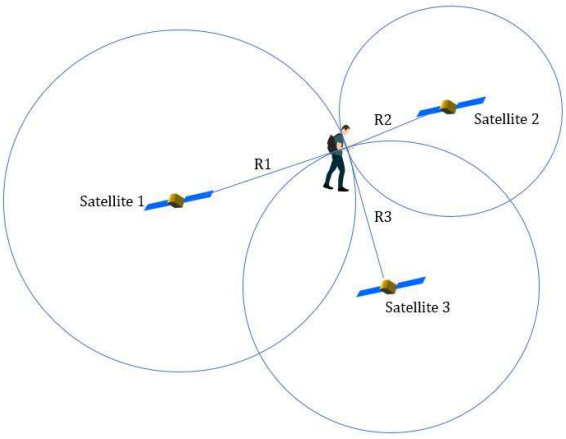

Figure 1 Trilateration Diagram [10]

The triangulation method, also known as the three-point resection problem, uses the angle between the device's direction of travel and the reference points to find its location. There are several ways in which this can be implemented in practise including iterative search, geometric, and geometric circle intersection methods. [10]

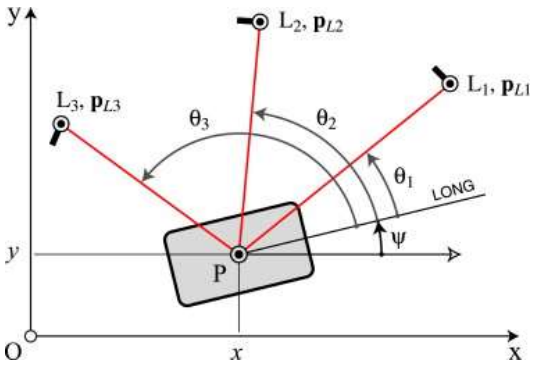

Figure 2 Triangulation Angles Diagram [11]

Without the use of any depth sensors or any other distance measuring equipment and only optical imaging sensors it seems that the triangulation method would be the more suitable accurate option. Although trilateration would be possible as mentioned in the Single-Camera Trilateration paper produced by State University of New York Polytechnic Institute, and Stony Brook University [12], the process of triangulation seems the less complex and more accurate for the medium of the system being used.

3.3.2 Z Coordinate

To calculate the z coordinate, the depth of the image to the ceiling can be subtracted from the height of the room as shown in equation 1.

$$Z\ coordinate = Height\ of\ the\ room - \ depth\ of\ the\ ceiling\ in\ the\ image \quad (1)$$

Equation 4 can be used to calculate the depth of the image to the ceiling. To use equation 4 characteristics of the camera alongside distance information calculated from the reference point detection must be known. Referring to figure 3, using the cameras focal distance, Og, and the angle of view of the camera, the physical width of the image at the focal distance, ab, can be calculated using basic trigonometry as shown by equation 2.

$$ab = 2 \times Og \times sin\left(\frac{1}{2} angle\ of\ view\right) \qquad (2)$$

Assuming reference point detection has been carried out and three reference points have been successfully located the length AB can be found. This is achieved by initially calculating the distance in the images horizontal between two adjacent reference points, this is then divided by the number of horizontal pixels between these two reference points to gain a distance per pixel measurement. Assuming you then know the resolution of the image, the distance per pixel can be used in the equation 3 to find the length AB.

$$Length, AB = Horizontal\ resolution\ \times distance\ per\ pixel\ (cm) \qquad (3)$$

Now you have all the input values equation 4 can then be carried out followed by equation 1 to gain your final z coordinate.

$$OG = \frac{Og}{ab} \times AB \qquad (4)$$

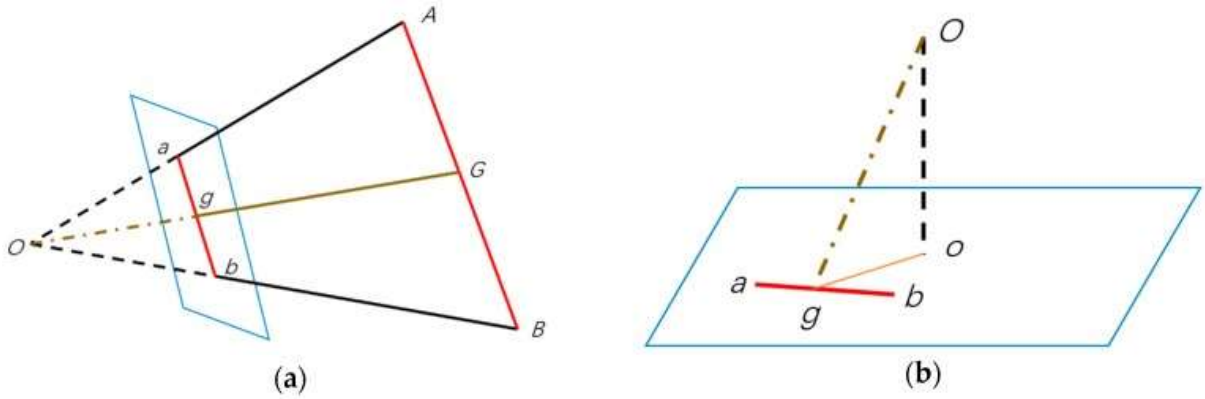

Figure 3 Depth Estimation Diagram [13]

3.4 Triangulation Methods

As mentioned in section 3.3.1, there are lots of different ways triangulation can be implemented but three of the main methods that are used are explained below:

3.4.1 Iterative Search

Iterative search is an algorithm that uses the principles that given the devices orientation is known its position can be located from two known landmarks/reference points. As in triangulation there is usually three known landmarks available there are three pairs of landmarks (1 and 2, 1 and 3, 3 and 2) giving the opportunity to guess the orientation of the device. This is done via repeatedly feeding in different angles of orientation into the system at regular intervals, applying the two landmark and orientation triangulation and finding the actual location when the output from all three pair calculations matches up. It could take large amounts of computation time to gain the actual angle therefore instead, a degree of closeness is created using the perimeter length of the circle in which all three predicted location points lie on. This can be used as a threshold as to when to stop the algorithm and then an error range can be made as the final location is going to be in between all three points. The pseudo algorithm can be seen in figure 4:

1. For an orientation from -90° to +90° in increments of 0.1° do the following:
    a. compute robot position from landmarks 1 and 2 by triangulating from the two landmarks.
    b. compute robot position from landmarks 1 and 3 by triangulating from the two landmarks.
    c. compute robot position from landmarks 2 and 3 by triangulating from the two landmarks.
    d. if an error was returned because the landmarks and robot are colinear, then triangulation can not be performed. Return with error.
    e. compute perimeter distance from points found in steps a, b, and c.
    f. save the values for the position that corresponds to the minimum perimeter computed so far.
2. Return with solution.

Figure 4 Pseudo Iterative Search Triangulation Algorithm [14]

The advantage of this algorithm is that a solution is guaranteed to be found. However, as is the nature of three object triangulation methods there are two solutions one at angle Θ and one at angle Θ+180°, hence the reason why the algorithm restricts the search between ±90°. This, however, assumes the robot's orientation error, the difference between the actual and predicted orientation, is within this bound. This algorithm has the potential to be very computationally expensive, if small orientation angle steps are used for example 0.1°. Some solutions for this could be to use better search algorithms or to parallelise the algorithm – both of which increase the complexity of the algorithm or require specialised hardware. [14]

3.4.2 Geometric Triangulation

This method implements triangulation using the geometry of the landmarks/reference points and the relative angles between the landmarks as viewed from the robot. For this method to work certain conditions must be met:

1. The robot must be at the centre of the landmarks.
2. The landmarks need to be labelled consecutively and, in a counter-clockwise order.
3. The angle β between the landmarks 1 and 2 and the angle between landmarks 1 and 3 α must be less than 180°

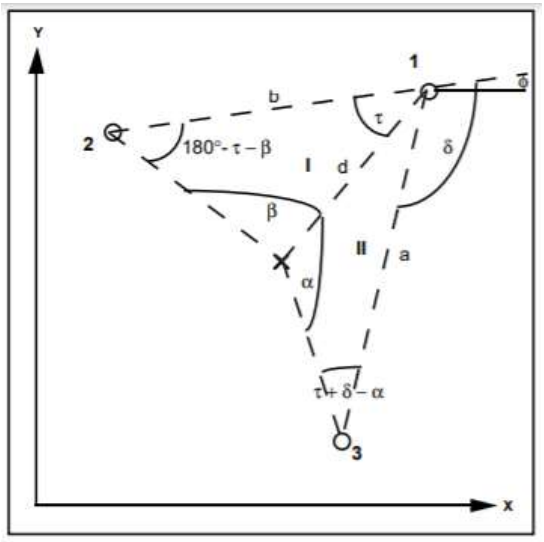

Figure 5 Geometric Triangulation Diagram [14]

Using this and the knowledge of the law of sines the algorithm in figure 6 was created:

1. Properly order landmarks.
2. Let $\alpha = 360° - Lo3 + Lo1$
3. Let $\beta = Lo2 - Lo1$
4. Let ϕ be the angle between the positive x-axis and the line formed by the points of landmarks 1 and 2.
5. Let σ be the angle between the positive x-axis and landmarks 1 and 3, plus ϕ.
6. Let $\gamma = \sigma - \alpha$.
7. Let "p" be a constant used from computing the law of sines (see below). $p = \frac{a*\sin(\beta)}{b*\sin(\alpha)}$.
8. Let $\tau = \tan^{-1}\left(\frac{\sin(\beta) - p*\sin(\gamma)}{p*\cos(\gamma)-\cos(\beta)}\right)$
9. Let $d = \frac{b*\sin(\tau+\beta)}{\sin(\beta)}$. d is the distance from landmark 1 to the robot.
10. $Rx = Lx1 - d*\cos(\phi+\tau)$
11. $Ry = Ly1 - d*\sin(\phi+\tau)$
12. $R\theta = \phi + \tau - Lo1$

Figure 6 Pseudo Geometric Triangulation Algorithm [14]

This method unlike the iterative search has a definite answer with it being as accurate as the data provided. The algorithm also only works consistently when the robot is inside the landmark triangles – with it working outside this in some cases but these areas are difficult to determine and highly dependent on how the angles are defined. [14]

3.4.3 Geometric Circle Intersection

This method makes two circles through pairs of landmarks/reference points and the robot's location i.e. 1 circle will be through landmark 1 and 2 and the robot and the second will be through landmark 2 and 3 and the robot. Even though at this point the robot's location is unknown there are only two positions in which the robot could be located which can be seen by the two circle intersections in figure 7 below.

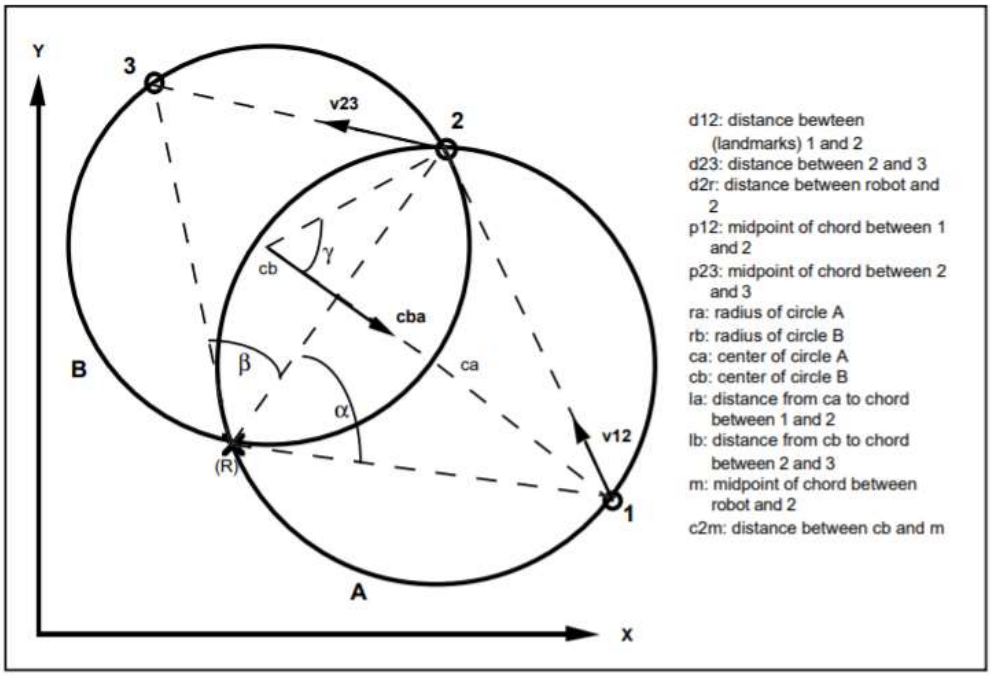

Figure 7 Geometric Circle intersection Triangulation Diagram [14]

From the information given, the equation of both circles can be determined and therefore used to locate their intersections. Even though there will be two solutions one of them is already known as

landmark 2 leaving the other to be the location of the robot. One pseudo algorithm example of a geometric circle intersection can be seen in figure 8 below:

1. Properly order landmarks. (see Geometric Triangulation).
2. $\alpha = Lo2 - Lo1$
   2a. If α is too small, or equals 90° or 270°, return with error because division by 0 will occur.
3. $\beta = Lo3 - Lo2$
   3a. If β is too small, or equals 90° or 270°, return with error because division by 0 will occur.
4. $ra = \frac{d12}{2.0*\sin(\alpha)}$
5. $rb = \frac{d23}{2.0*\sin(\beta)}$
6. $la = \frac{d12}{2.0*\tan(\alpha)}$
7. $lb = \frac{d12}{2.0*\tan(\beta)}$
8. Let v12x and v12y be the unit vector from landmark 1 to landmark 2.
9. Let v23x and v23y be the unit vector from landmark 2 to landmark 3.
10. $cax = p12x - la * v12y$
11. $cay = p12y + la * v12x$
12. $cbx = p23x - lb * v23y$
13. $cby = p23y + lb * v23x$
14. Return an error if the centers of the two circles are too close (we used 10 units).
15. If γ is very large, then return an error.
16. Let cbax and cbay be the unit vector from the center of circle B to the center of circle A.
17. $d2r = 2 * rb * \sin(\gamma)$
18. $c2m = rb * \cos(\gamma)$
19. $Rx = 2 * mx - Lx2 + 0.5$
20. $Ry = 2 * my - Ly2 + 0.5$
21. $\phi = \arctan(\frac{Ly1-Ry}{Lx1-Rx})$; the heading of landmark 1 from the true robot position.
22. If $\phi > 0.0$, then orientation error = $-(Lo1 - \phi)$ else orientation error = $360° + \phi - Lo1$
23. Return with solution

Figure 8 Pseudo Geometric Circle Intersection Triangulation Algorithm [14]

Due to the use of simple vector manipulations, the algorithm is very fast to find the solution. However, errors occur when all the landmarks and the robot all lie on or are close to the same circle leaving the algorithm useless. [14]

The iterative Search method performs well and has the advantage of always giving a result to a certain accuracy, using the perimeter marker. However, for the application in this project, the computational power, and the time to achieve the location accuracy that is required is not suitable. This is because of the limited processing power available, using a microprocessor, causing extra latency when the updates are required in as close to real-time as possible. The Geometric method is very quick and can achieve a result as accurate as its input data. But the inconsistent performance of the location values output when outside the triangle of the reference is a concern especially as there is not a guarantee that this will always be the case. Therefore, from the research it seems the Geometric Circle intersection triangulation seems to be the most suitable for this project with a quick consistent output in most situations. As for the potential errors, they are very unlikely conditions, and software can be easily written to mitigate these circumstances.

4 Method

4.1 System Overview

Initially there was a differ in directions in which this hardware could be used. Either there would be multiple pi cameras and a single reference point on the robot to detect and locate. The other being there is a single camera mounted on the robot and multiple reference points in the camera's view. After research and facility analysis it was decided that the configuration being used was the single camera approach. This was due to two main factors, one of which was cost. If the first approach was used at least three cameras would be required to perform triangulation, potentially requiring three micro-processors tripling the cost of the project hardware. Furthermore, due to the impact of COVID-19 the facilities and space available to be able to test a fully calibrated multi-camera system were not available until the final stages of this project, hindering initial productivity. Therefore, instead of the robot tracking system, a robot self-positioning system was chosen to be designed.

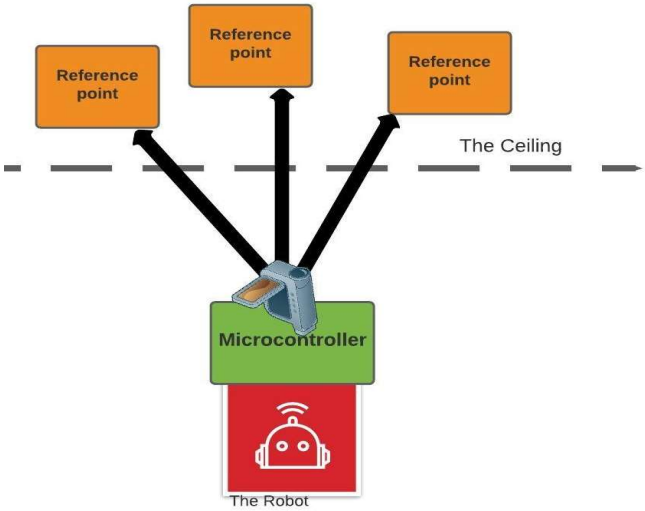

Figure 9 Basic System Diagram

4.1.1 The Hardware

To implement the Indoor positioning System the hardware in this project consisted of:

- raspberry Pi 4 Computer model B 4GB RAM
- A raspberry pi camera – video resolution 1080p
- A series of large numbers from 1-6 written on A4 pieces of paper as reference points.

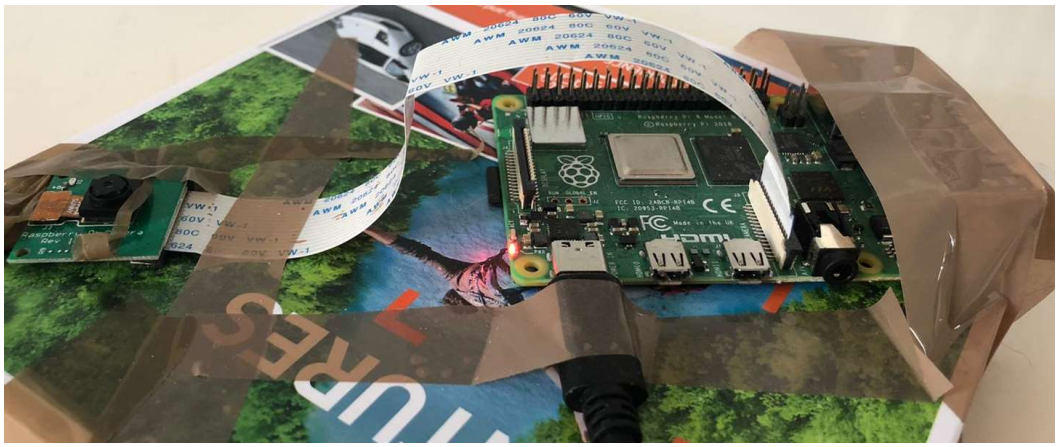

Figure 10 Image of raspberry pi and pi camera secured into position.

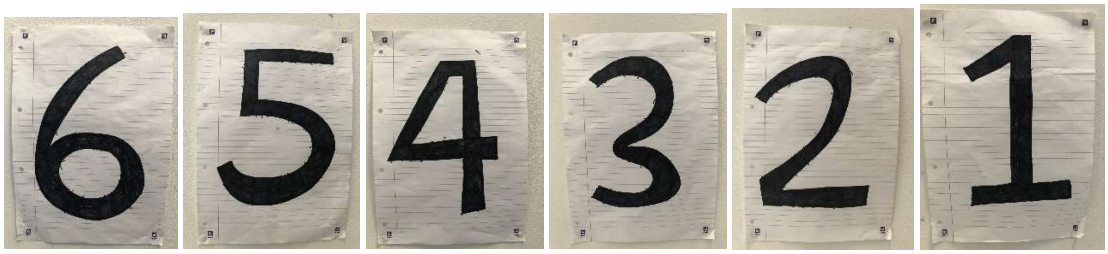

Figure 11 Images of the 6 Reference Points

4.1.2 High Level Functionality

A high-level overview of the system consists of a pi camera directed at the ceiling in which at least 3 of the reference point numbers will be in the cameras field of view. The image detection algorithm on the raspberry pi will then identify and locate the reference points in the image. The reference point location information in the image is then converted into a format in which it can be used in the triangulation and z depth algorithms to locate the devices final 3d location.

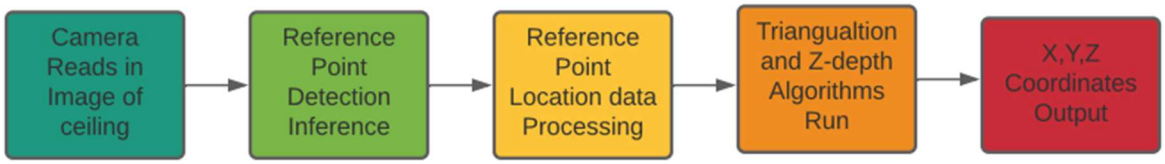

Figure 12 High Level System Block Diagram

4.1.3 Project Assumptions

With the above in mind a couple of assumptions have been made about the design to allow basic prototyping and testing:

- The pi camera will always be parallel to the ceiling.
- The depth of the camera from the reference points will be no more than 2.5 metres.
- All reference points will be at the same height from the floor.

4.2 Reference Point Detection

After deciding to use a deep learning algorithm for the reference point detection, research began into how to implement it. TensorFlow [15] an end-to-end open-source platform for machine learning, offers a pre-made object detection API with a 'model zoo' [16] of implemented algorithms, including R-CCN and SSD as mentioned in section 3.2. It also contains a lightweight version of the API which has been developed for use on smaller devices such as mobiles and micro controllers called tensorflowlite (TFlite) [17], confirming the use of this technology. All models in the zoo are all pre trained on a generic dataset called COCO 2017 [18], this is a dataset consisting of over 330 thousand images with 80 object categories – meaning the model can identify 80 different object classes in an image. However for this implementation the classified objects in this dataset did not consist of numbers between 1 and 6 hence the model had to be trained on a custom dataset – consisting of 100's of images of the reference points in different positions on the ceiling. After the training of the model was complete it was ready to be implemented into the system.

4.2.1 Choosing the model

When choosing the most appropriate model in the TensorFlow model zoo, metrics were provided of trade-offs between accuracy and time to produce a result on a set of test images from the COCO dataset. With all the models' speeds ranging from over 300 ms to 6 ms and the accuracy ranging between 20 and just over 50, the SSD MobileNet V2 FPN 640x640 model was chosen with a speed of

48 and an accuracy 0f 29.1. This is a single stage algorithm which performs at an intermediate level between the slow, highly accuracy models and vice versa. It was also important that the resolution of the images was kept high enough so that the detection model was more responsive to reference points located further away in the image.

4.2.2 The Dataset

With the COCO dataset not providing labelled images containing numbers like the reference points, research for a further dataset was required. The Mnist dataset [19], showed promising potential, consisting of 5000 handwritten digits from 250 different people. Each image in the dataset has a resolution of 28x28 and portrays an individual digit filling most of the picture. As these were individual images, the dataset had to be augmented and mixed into a larger image for there to be different digits in different locations, sizes, and orientations. This was to simulate how the images would look on the ceiling of the test environment. To do this a pre-made script from a GitHub repository called yymnist [20] was used for this exact purpose. After creating the dataset using this script the data was used to train the model but unfortunately because of the difference in background between the camera frames and test data it was not effective at detecting the reference points.

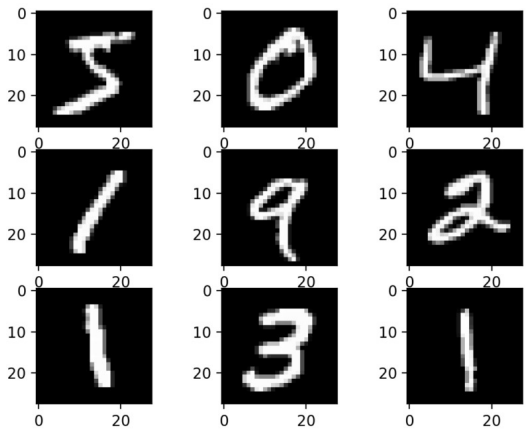

Figure 13 Examples of mnist dataset [21]

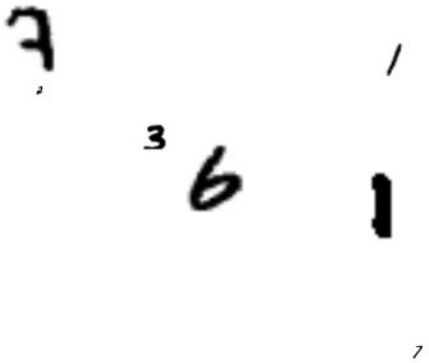

Figure 14 Augmented mnist dataset image

Therefore a custom dataset was created using images taken of the test environment. All the reference points were placed in different locations around the test environment and using an iPhone 7's front facing 7-megapixel camera multiple pictures were taken of each reference point number in different orientations, at different distances, in different lighting conditions. Each image was then labelled using labelImg [22], an application that facilitates picture annotations by the user drawing a bounding box around objects in images and labelling them. This creates an output file alongside the original image, the type depending on the labelling format the user chooses. In this case the PASCAL labelling format was chosen so an xml was produced per image consisting of the position of the bounding boxes and the associated reference identification tags. The final dataset consisted of just under 650 labelled images.

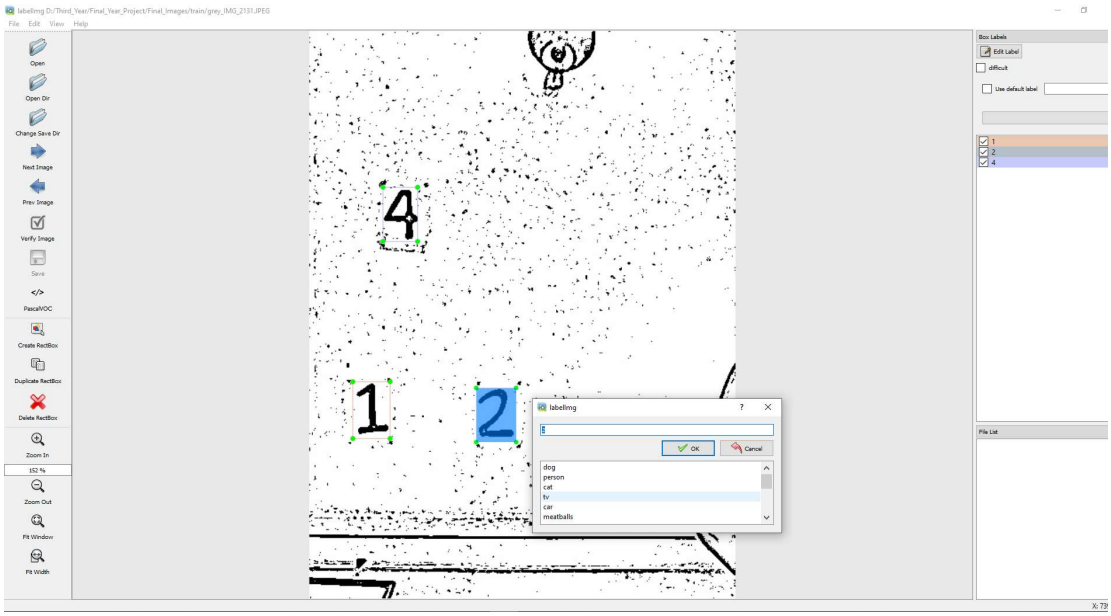

Figure 15 Screenshot Of labelImg in use

4.2.3 Pre-training processing

After some initial attempts at training the algorithm using the custom dataset photos, it was found that the model was still not locating the reference points as accurate as was required. To counter this, the images were pre-processed to remove as much unneeded information as possible leaving only the reference points in view. To do this the images were read into the following code as greyscale, making each pixel consist of a single value between 0 and 255 instead of the three R, G, B values. A median Blur filter was then applied followed by an adaptive threshold filter turning each pixel black or white depending on its value above or below a certain threshold. The image was then duplicated and turned back into the RGB format as shown using the np.repeat function to make the image information in a format fit for training the model. Figure 17 below shows the gradual addition of each image processing filters to gain the final image format used to train the models.

```
for filename in os.listdir(dir_path):
   if filename.endswith(".JPEG"):
      img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
      img = cv2.medianBlur(img,5)
      threshold =
cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,11,2)
      threshold = np.repeat(threshold[..., np.newaxis], 3, -1)
      cv2.imwrite("grey_"+filename,threshold)
```

Figure 16 Image Processing Code Used Pre training

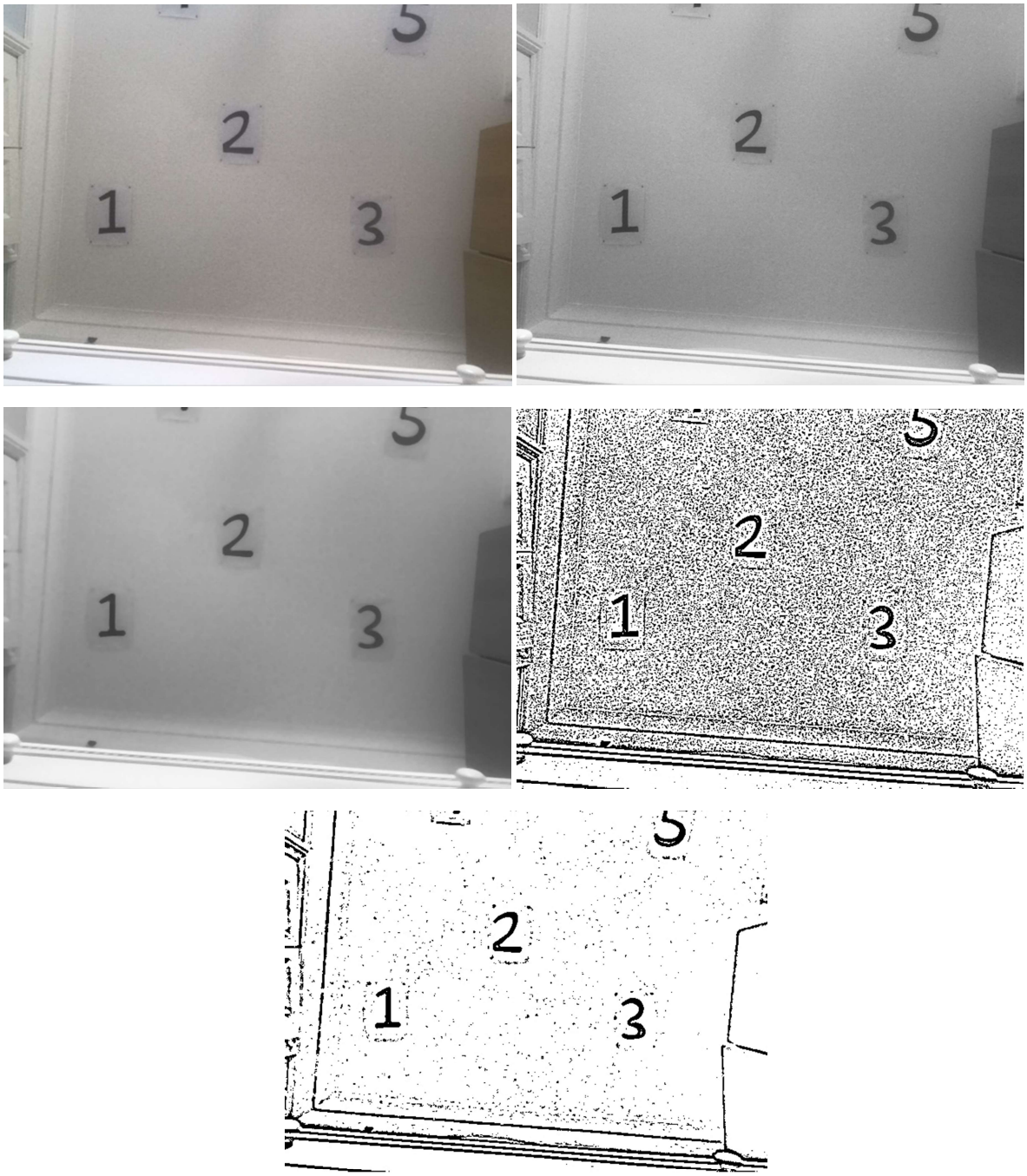

Figure 17 Group of Imaging showing the change in the image after each line of code in figure 16.

4.2.4 Training

With the aid of the following document [23] the training was configured and ran.

The first decision was on the hardware to use, TensorFlow has the option to run its training on Google Collaboratory, a research service which allows anyone to write and execute arbitrary python code on remote computers through their web browser. The other option is if the user has a dedicated GPU or fast CPU, they could run it on this. As the laptop available had an inbuilt dedicated GPU the computation was executed locally.

Before the training could start, the dataset had to be split into train and test data, the first being used for teaching the algorithm the location of objects and what to look for and the second being used to check the accuracy of the algorithm during training and monitor its progress. The data was

split approximately 15% test and 75% train. After the data was split and converted into the correct format and the training algorithm was configured the training began. TensorFlow has an inbuilt dashboard called TensorBoard [24] which allows you to monitor the training progress through a user-friendly web browser GUI. It predominantly records the output data of the loss function; this quantises how far the predictions of the object detection model deviates from the actual object locations in the test images and displays it in a graphical fashion as shown in figures 18 and 19. Usually in the training process the value of the loss function is output every 100 steps which allows the user to see the progression of the training. The training is finished when either the loss graph starts to plateau, or the learning graph drops to zero showing that no further learning is taking place. Figures 18 and 19 show the final object detection model training graphs used in the project this process took approximately 15.5 hours to train. Note how the learning rate becomes effectively zero at approximately 95k steps which is also shown through the plateauing of the loss/total_loss graph at the same range signalling that it was a good time to stop the algorithm training.

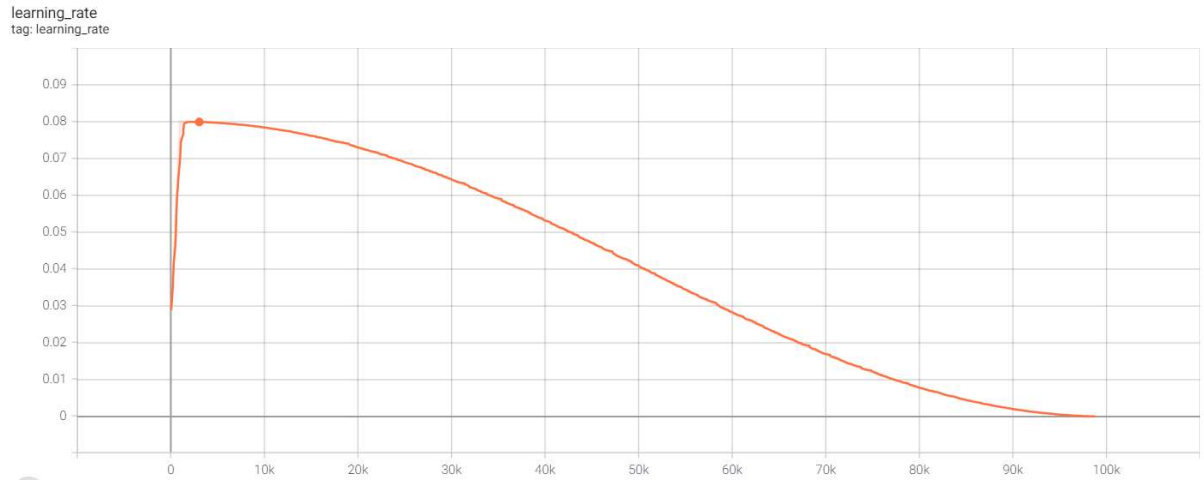

Figure 18 TensorBoards Learning Rate graph

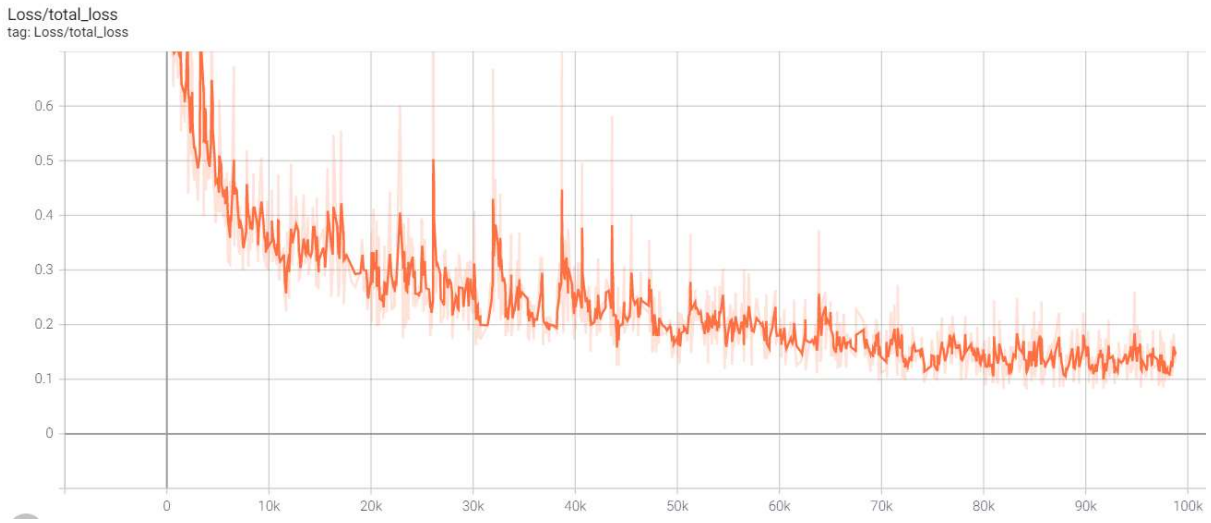

Figure 19 TensorBoards Total loss graph

4.2.5 TfLite Conversion and Inference

The final step before the object detection model can be used on the raspberry pi is converting the trained model into a TfLite model, so it can run smoother with less computing power. This is achieved due to two main factors, the interpreter it uses and the compression of the model it runs.

For any TensorFlow model to run on a device they need an interpreter to know how to use the model. In the original TensorFlow this interpreting package is very large and has lots of very specific libraries which are not needed in all types of inference. TfLite has its own interpreter called tflite_runtime, a slimmed down version of the TensorFlow package containing all the core functionalities making it ideal to run models with no wasted disk space.

During the conversion of the fully trained TensorFlow model to TensorFlow Lite the model is compressed [25]. The factor of compression can vary from model to model. The affect in this project was a compression factor of nearly a quarter with the model size decreasing from 11,549 KB to 4,128 KB. There are two main ways the models are compressed, the initial being that it transforms the models' data into FlatBuffers [26]. These are known for their effect of reducing size and faster inference time due to the data not needing to be unpacked in an extra step. The second is quantization [27] of the model. Depending on the hardware available to run the model, different types of quantization are advised. As only a single quad-core CPU is available on the raspberry pi, dynamic quantisation was chosen. This is a process in which all the 32-bit floating-point weights of the model are quantized to 8-bit integer values and all further calculations are done in this format improving latency on the system.

When the conversion to TfLite was complete the object detection model could then be implemented in the code. As shown in the pre-processing section 4.2.3, the images used for the model training were pre-processed. Because of this, the images taken by the camera must also be altered in the same way for the model to work correctly. Therefore the same code is used on each frame captured before the reference point detection model is run on the frame. To begin the inference and start the object detection on the captured frame the following code is used:

```
# Normalize pixel values if using a floating model (i.e. if model is non-quantized)
 #if floating_model:
 input_data = (np.float32(input_data) - input_mean) / input_std
# Perform the actual detection by running the model with the image as input
interpreter.set_tensor(input_details[0]['index'],input_data)
interpreter.invoke()
# Retrieve detection results
boxes = interpreter.get_tensor(output_details[0]['index'])[0] # Bounding box coordinates of
detected objects
classes = interpreter.get_tensor(output_details[1]['index'])[0] # Class index of detected objects
scores = interpreter.get_tensor(output_details[2]['index'])[0] # Confidence of detected objects
```

Figure 20 Code used to run the TfLite model and analyse its output [28]

As seen from the code snippet the images are initially normalised to convert them to a suitable format for the model runner. The output of the inference gives three key tensors of information located in the output_details variable. This variable contains the bounding box information made from a tensor list of all the box coordinates in the form of [ymin, xmin, ymax, xmax]. The accuracy score of each of the bounding boxes as a confidence level of how well what is identified in the bounding box is correct, given in another tensor list. And finally the object tags themselves, also

given as a list tensor. This information is stored in the boxes, scores, and classes variables respectively with the information associated via their index, in descending order in accordance with the class accuracy score as shown by the example printed output in figure 21.

```
boxes: [[ 4.4935960e-01  2.5311375e-01  5.6909949e-01  3.9682543e-01]
 [ 1.2591863e-01  6.0190207e-01  2.4998239e-01  7.5109297e-01]
.
.
 [ 6.0373533e-01  8.5678262e-01  7.8827703e-01  9.9655586e-01]
 [ 8.8978000e-03  4.3124035e-03  1.0078575e-01  3.7488293e-02]]
classes: [3. 2. 0. 2. 0. 5. 5. 2. 0. 4.]
scores: [0.96774954 0.956555   0.4894709 0.11068884 0.10077515 0.07698089 0.04009509
0.03965437 0.03783795 0.03737625]
```

Figure 21 Print out of TfLite model output data format

4.3 Angle Detection

Using Triangulation to find the final position of the device means at least three angles need to be found from the device to the three unique reference points. All angles calculated in a circular fashion around the centre point of the image between the reference points centre location in the image with the vertical line towards the top of the image taken as 0° as shown in figure 22.

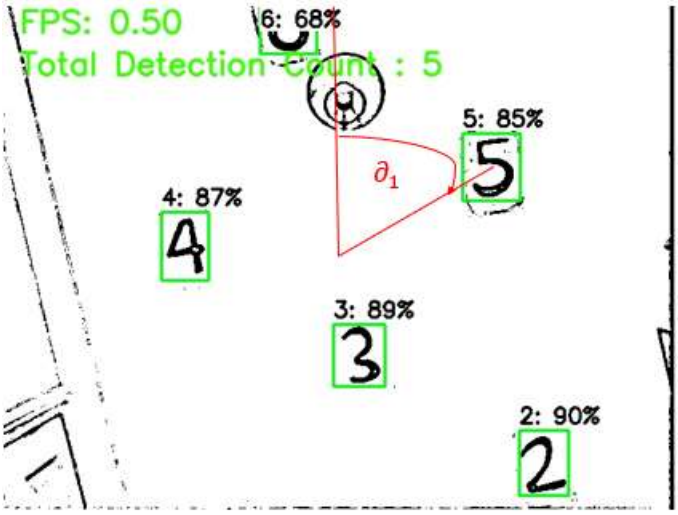

Figure 22 Diagram visualising the bounding boxes and Angle Detection.

In the classes tensor from the object detection output, any number of the same class/reference point tag can appear as shown in figure 21. As all the reference points used are unique, the classes tensor, which is ordered from highest to lowest accuracy score, is filtered through to locate the first 3 different reference points and find their indexes. The below code shows how this has been achieved:

```
def find_class_box_index(classArray, labels):
    classArray = classArray.tolist()
    index = [[0, labels[int(classArray[0])]]]
    value = [classArray[0]]
    for number in classArray:
        if number not in value:
            index.append([classArray.index(number), labels[int(number)]])
            value.append(number)
            if len(index)==3:
                break
    return index
```

Figure 23 Code to find Index of Different reference point bounding boxes.

With the above indexes the appropriate bounding boxes are then found in the boxes tensor. The centre of each box is then found with a simple mean of the max and min of the x and y coordinates separately.

Finally to gain the angle of each reference point, basic trigonometry in SOHCAHTOA and a quadrant system is used to check which quarter of the image the reference point is in and adds the appropriate multiple of π/2. This can be seen implemented in the following code:

```
def calculate_angles(centreCoords, imageCentre):
    angles = []
    for centre in centreCoords:
        x = abs(centre[0]-imageCentre[0])
        y = abs(centre[1]-imageCentre[1])
        if centre[0]>imageCentre[0] and centre[1]<imageCentre[1]:
            angle = math.atan(x/y)
        elif centre[0]>imageCentre[0] and centre[1]>imageCentre[1]:
            angle = math.atan(y/x)+ math.pi/2
        elif centre[0]<imageCentre[0] and centre[1]>imageCentre[1]:
            angle = math.atan(x/y) + math.pi
        elif centre[0]<imageCentre[0] and centre[1]<imageCentre[1]:
            angle = math.atan(y/x)+ (3*math.pi)/2
        angles.append(angle)
    return angles
```

Figure 24 Angle Calculation code

4.4 Calculating Location Coordinates

4.4.1 X, Y Coordinate Implementation

As mentioned in section 3.4, a geometric circle intersection method is used for implementing the triangulation. The particular algorithm being used is known as the ToTal Algorithm [29] which uses mostly very basic arithmetic operations, and two cot(.) operations. The outcome is a fast, simple set of easily implemented instructions, which reliably calculate the location of the device in question. A code implementation of the algorithm can be seen in figure 25 with the pseudo code found in appendix 1.

```
def ToTal_Algorithm(refCoords):
    # compute modified beacon coordinates
    x_1 = refCoords[0][0] - refCoords[1][0]
    y_1 = refCoords[0][1] - refCoords[1][1]
    x_3 = refCoords[2][0] - refCoords[1][0]
    y_3 = refCoords[2][1] - refCoords[1][1]
    # compute three cot(.)
    T12 = mpmath.cot(refCoords[1][2]-refCoords[0][2])
    T23 = mpmath.cot(refCoords[2][2]-refCoords[1][2])
    T31 = (1 - T12*T23)/(T12 + T23)
    #compute modified circle centre coordinates
    x_12 = x_1 + T12*y_1
    y_12 = y_1 - T12*x_1
    x_23 = x_3 - T23*y_3
    y_23 = y_3 + T23*x_3
    x_31 = (x_3 + x_1) + T31*(y_3 - y_1)
    y_31 = (y_3 + y_1) - T31*(x_3 - x_1)
    # compute k_31
    k_31 = x_1*x_3 + y_1*y_3 + T31*(x_1*y_3 - x_3*y_1)
    #compute D
    D = (x_12 - x_23)*(y_23 - y_31) - (y_12 - y_23)*(x_23 - x_31)
    #compute the robot position
    xR = refCoords[1][0] + (k_31*(y_12 - y_23))/D
    yR = refCoords[1][1] + (k_31*(x_23 - x_12))/D
    return xR, yR
```

Figure 25 ToTal Algorithm code Implementation

The ToTal algorithm was chosen instead of others in its family as it does not have any major limitations – with others requiring either the angles to be input in a certain order, having to be within the triangle of reference points, or have blind spots. There are of course some more reliable algorithms available although with the increase in reliability also comes the increase in complexity and hence the speed of it is degraded. ToTal has also been extensively tested and compared to other similar algorithms the most recently developed being the Ligas algorithm. Analysing the benchmark results in appendix 2, you will see that the ToTal is the fastest and most simple algorithm with the number of operations such as trigonometric functions and square roots also listed in the table.

Particular care has to be taken when using this algorithm, due to the potential of cot values tending to infinity and the D value, as seen in figure 25, tending to 0 as this could also create a tend to infinity. The situations in which these occur are when the bearing angle between two reference points is equal to 0 or π. To mitigate these situations the algorithm can be slightly altered as shown in the pseudo code in appendix 3. This was implemented in a separate function in the code, as seen in appendix 4, and depending on the values of the angles input will determine the function called.

4.4.2 Z-Depth Coordinate implementation

Implementing the method described in section 3.3.2, due to different orientations of the robot, the physical distance in the horizontal of the image between two reference points is not always easy to calculate. The reference points coordinates are known, and hence the direct distance between the two points can be easily calculated using Pythagoras' theorem. The locations of the centre of the reference points in terms of pixels is then used to calculate the angle required to resolve the direct distance in the images x direction. Once we have this value the distance per pixel can be calculated

to find the distance covered by the image in its x plane. This length alongside the scale factor compiled from the cameras characteristics is then used in the equation in section 3.3.2 to calculate the depth. The acquisition of the scale factor values will be explained in a later section. The implementation of this can be seen in the figure 26 below:

```
def z_depth_calculator(refCoords):
    #order the reference in size according to the x values
    refCoords.sort(key=lambda l:l[3])
    # calculate the angle between the two chosen reference points using pixel values
    x = refCoords[2][3]-refCoords[0][3]
    y = abs(refCoords[2][4]-refCoords[0][4])
    pixelAngle = math.atan(x/y)
    # calculate the actual distance between two reference points
    rx = abs(refCoords[2][0]-refCoords[0][0])
    ry = abs(refCoords[2][1]-refCoords[0][1])
    actualDistance = math.sqrt(rx*rx + ry*ry)
    # resolve the actual distance in the x direction of the image
    imageXDistance = actualDistance * math.sin(pixelAngle)
    #work out the distance per pixel and multiply up by the image width size to get actual image distance
    disPerPixel = imageXDistance/x
    actualImageXdistance = disPerPixel*640
    #use the scale factor of the length of the image at the focal distance to calculate the depth
    zDepth = 88.5/92 * actualImageXdistance
    return zDepth
```

Figure 26 z-depth calculations code implementation

4.5 Output UI

To make the output information quick and easy to understand a basic map and point system was created to show the device' location in real-time. OpenCV's rectangle function was used to represent the shape of the room and where different pieces of furniture were. Several unfilled circles were also used to represent the locations of the reference points and a small filled black circle was used to show the current location of the device in the x, y horizontal plane with the z coordinate written in the bottom right-hand corner as seen in figure 27. The X, Y and Z values are also posted to the command line as shown in the figure 28. All values output by the system are in centimetres.

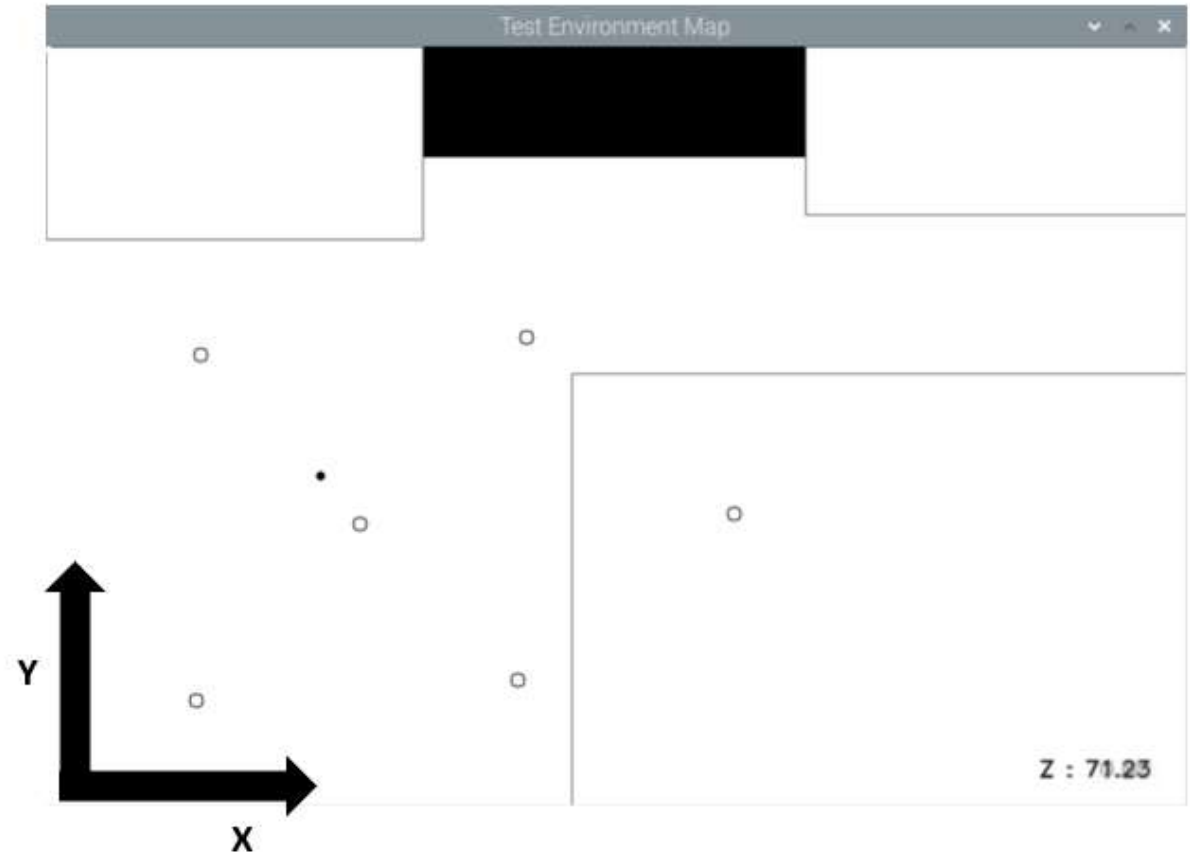

Figure 27 Screenshot of the UI output by the system

```
x coordinate is 137.966637182218
y coordinate is 74.4345008464571
z coordinate is 66.13687900633002
```

Figure 28 Example Output of the systems command Line

4.6 Integration

To bring together all previously mentioned, an open-source piece of code by armaanpriyadarshan [28] was edited. This code consisted of basic functions that were required in the system such as running the video stream in a separate thread, running the object detection interpreter and displaying and drawing the bounding box around reference point in the images, the latter being used during the testing stage, an example of which can be seen in figure 22.

The final code was split into two main python scripts triangulation.py and Position Location.py. triangulation.py consists of all the functions used in all the post object detection calculations including the z-depth and triangulation algorithm implementations. Position Location.py is the altered script mentioned above in which the camera is set up to capture 640x480 images, the object detection model interpreter is prepared to run inference and all of the functions from triangulation.py are called to run the system. A program flow diagram can be seen in figure 29:

Please see the full code in the following gitHub Repository:

https://github.com/will-prior/BEng-Final-Year-Project-Code-Position-Location-System.git

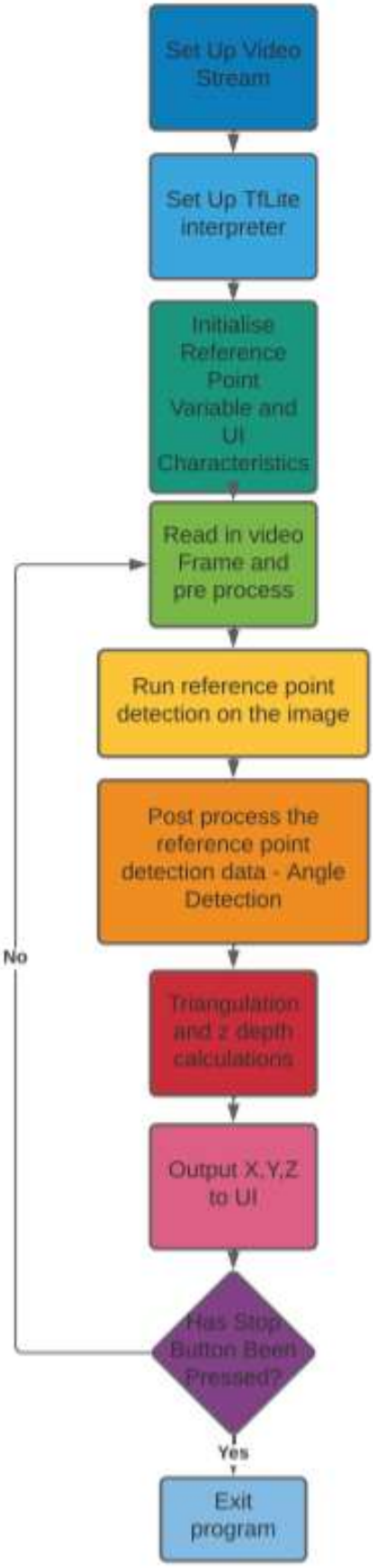

Figure 29 System software Flow Diagram

5 System Testing

5.1 The Test Environment and Test Configuration

Working from home due to the Covid-19 pandemic meant that only small-scale testing could be completed. The room chosen for the testing has the dimensions of 390x260x231.5 cm but due to the obstruction of furniture this was reduced to a testing environment with dimensions 185x225x231.5 cm. With the X, Y axis of the room run with 0 in the bottom left-hand corner of the room as seen in figure 27 and the Z coordinates taken from ground level as 0.

From the error analysis of the ToTal algorithm carried out in the paper written by Vincent Pierlot and Marc Van Droogenbrook [29], it was clear although the algorithm was able to locate devices outside the triangle of the reference points, the most accurate was inside. Using this information the reference points were set up in a triangular formation to both reduce the number required and optimise the algorithms output accuracy. The test environment and the corresponding reference point locations can be shown in figure 27.

During Testing the camera was constantly being moved to different locations around the room, at different heights and on different surfaces. In each case it had to be recalibrated with the ceiling to make sure that it achieved the assumption of parallelism with it. To achieve this, one of openCV's drawing function was used to draw a circle with radius of three pixels at the centre of each frame captured by the camera in the video stream. A small circle was then placed on the ceiling and its X, Y location was measured. The camera pointed at the ceiling, was then positioned in the same X, Y, coordinates on the ground with the live stream showing the view of the ceiling. The centre of the image was then manually lined up with the dot on the ceiling and secured in place, the camera was then rotated and checked in different orientations, making the ceiling and camera image parallel.

5.2 Camera Testing

To be able to complete the depth calculations, the horizontal angle of view and a focal distance ratio from the camera was required. The specification of the camera states it to have an angle of view of 62° and the focal distance to be 0.3m. The dimensions of which however were unknown, hence the following test was carried out.

To work out the horizontal angle of view, a live stream from the camera was positioned with the lens facing the centre of the width of the door. The camera was then moved closer to the door until the vertical edges of the image frame lined up with the width of the door. From this position the width of the door, 92cm and the distance of the camera to the door 88.5cm was recorded. A simple SOHCAHTOA trigonometric calculation was then carried out using half the door width and the distance away from the camera, to find half the angle of view which was subsequently doubled to gain the full horizontal angle of view at 54.92°.

These measured distances could then be used as a scale factor of distance and width to be used in equation 4 to calculate the z depth of the camera from the ceiling. Hence the final scale factor was 88.5/92, as can be seen implemented in the code in figure 26.

5.3 Location Position Testing

5.3.1 X, Y Position Testing

To test the X, Y position accuracy of the system, 5 routes were determined around the test environment to monitor the reaction of the system to different ways of moving around the room, the routes can be seen in figure 30. 7 to 11 positions were determined along each route at regular intervals of 15-25 cm. At each position, the device was given a couple of seconds to settle onto the

final position and then 3-4 readings were taken. Each reading recorded – the x, y z coordinates and the accuracy score of the reference points for the given reading. These tests were carried out at 3.5cm from the floor, to push the reference point detection system to its maximum possible. This was carried out by mounting the raspberry pi and camera on a flat-topped slim box and moving the box around the floor from position to position.

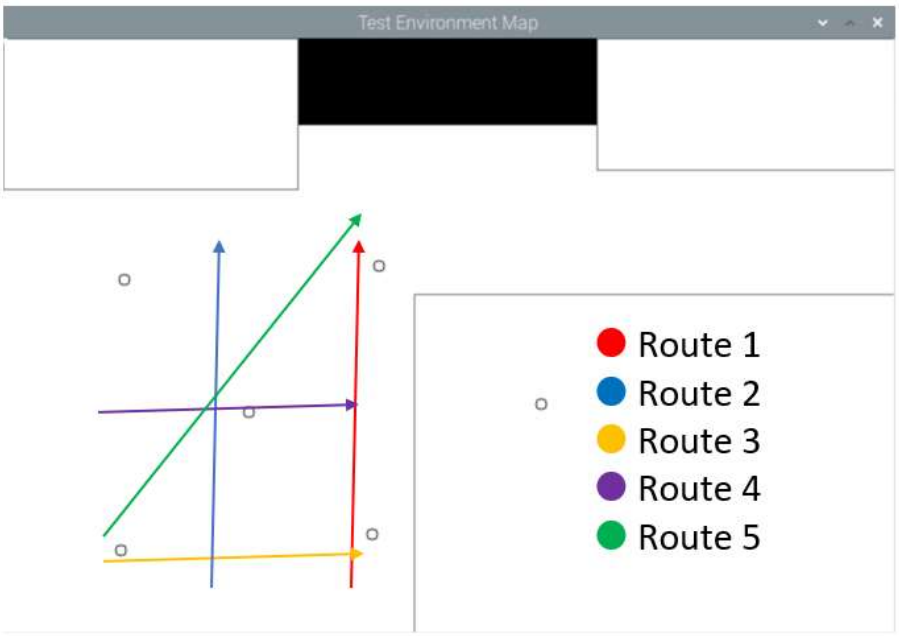

Figure 30 X, Y, Position Testing Routes

Routes 1 and 2 test the devices reaction to changes in the y coordinates of the room. With 1 used to monitor how it works while being directly underneath/in line with some of the reference points and 2 to see how it works cutting across the reference point triangle. Routes 3 and 4 were used to test how the device coped with changes in the x coordinates. 3 also checking to see how it coped having all the reference points on one side of the cameras viewpoint and 4 used to check how much of an improvement there would be when in the same orientation through the middle of the reference points. Finally test 5 was done to show how effective it was at tracking itself when at an angle to the reference point and the environmental axis.

Once the information had been captured error analysis took place in which some further calculations were required to get some of the error information. As well as individual coordinate error there will also be a combined error calculation involving the Euclidean distance between the actual and estimated position of the device. The following equation can be used to gain this information:

$$3d\ Euclidean\ Distance\ Error\ Equation:$$

$$\Delta d_{3d} = \sqrt{(x_{true} - x_{estimated})^2 + (y_{true} - y_{estimated})^2 + (z_{true} - z_{estimated})^2} \qquad (2)$$

5.3.2 Z-Depth Testing

To see how reactive the system was to change in vertical distance and the associated errors that came with it the following test was carried out. A single position, [150, 100] was chosen in the test environment in which 3 distinct reference points could be easily seen in the view of the camera at different vertical heights. Starting at 3.5 cm from the ground, three readings were taken in orientations of 0, 45 and 90 degrees to the room's axis. The device was then raised vertically to a new position and the same three readings were taken again, this happened for 5 different heights

the final being 71.5 cm off the ground. Like the previous test, the readings noted the x, y, z locations as well as the accuracy score of each reference point used to calculate the output values. Due to lack of equipment it was not easy to find objects which had a flat-topped surface which increased in height in regular intervals, hence the range of heights shown in the appendix 5 were used.

5.4 Code Performance Testing

To understand how quickly the device can react to changes in its position, some basic code profiling, and metrics such as the time between outputs were analysed. The profiling split the code into its 4 main blocks, the image capture and filter application processing time, the reference point detection time, the image data processing time and finally the triangulation and z coordinate calculation time. This was carried out using OpenCV's timing tools in cv2.getTickCount() and cv2.getTickfFrequency(), these gather the number of CPU cycles completed in between two blocks of code and the tick frequency is used to gain the number of CPU cycles per second so can be used to find the time in seconds using the following equation:

$$Time(s) = \frac{number\ of\ CPU\ cycles}{frequency\ of\ CPU\ cycles} \qquad (5)$$

The program was run for 100 results capturing the profiling data including the time between outputs. All results were then averaged and used to give an insight into which part of the code causes the bottleneck in the programs run time. This process was run twice once while the camera is stationary and another with the camera constantly being moved within the test zone to see if there is a significant processing time difference.

6 Results

6.1 Location Accuracy

Each Route has been plotted with the reference point locations, the actual position in which the device was placed the three/ four estimated points recorded by the device at each location and then the average of these points.

6.1.1 X, Y Position Testing Results

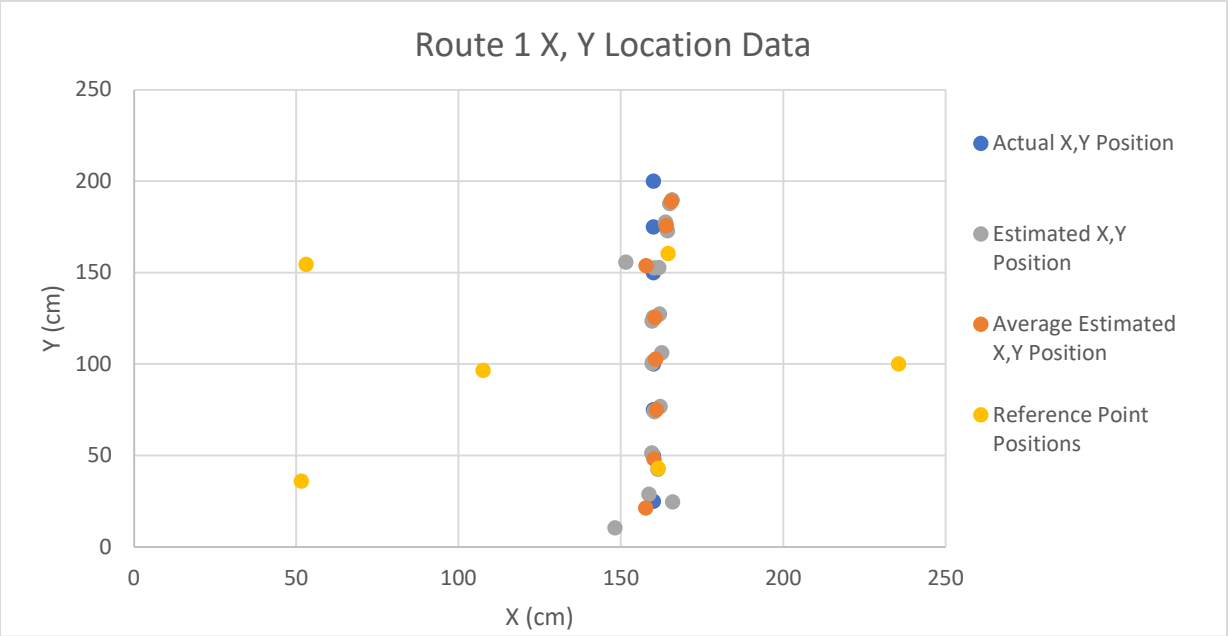

Figure 31

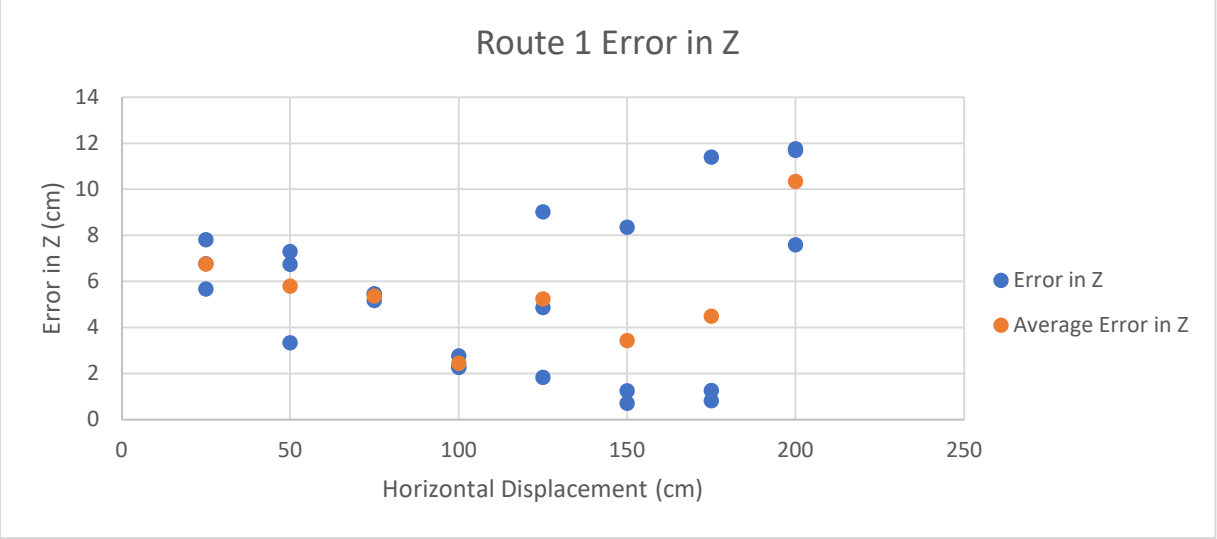

Figure 32

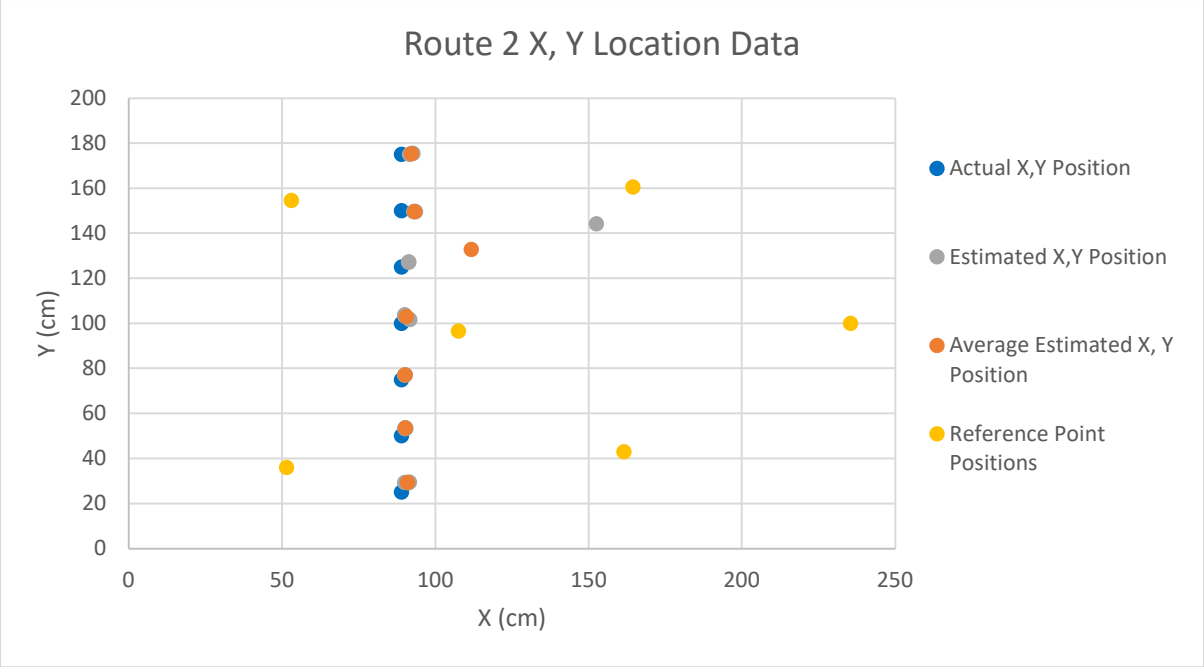

Figure 33

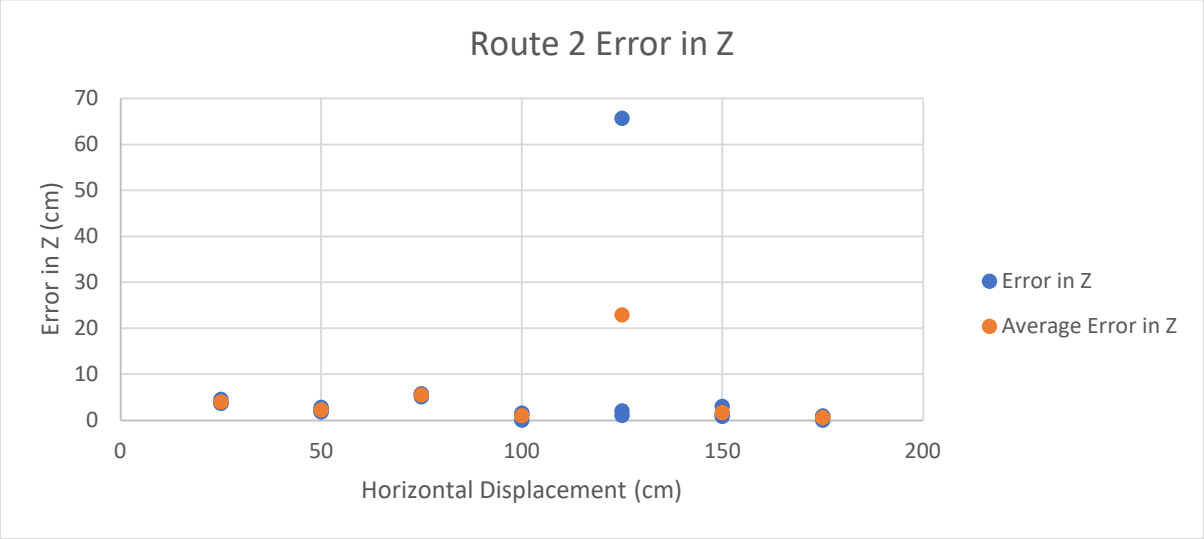

Figure 34

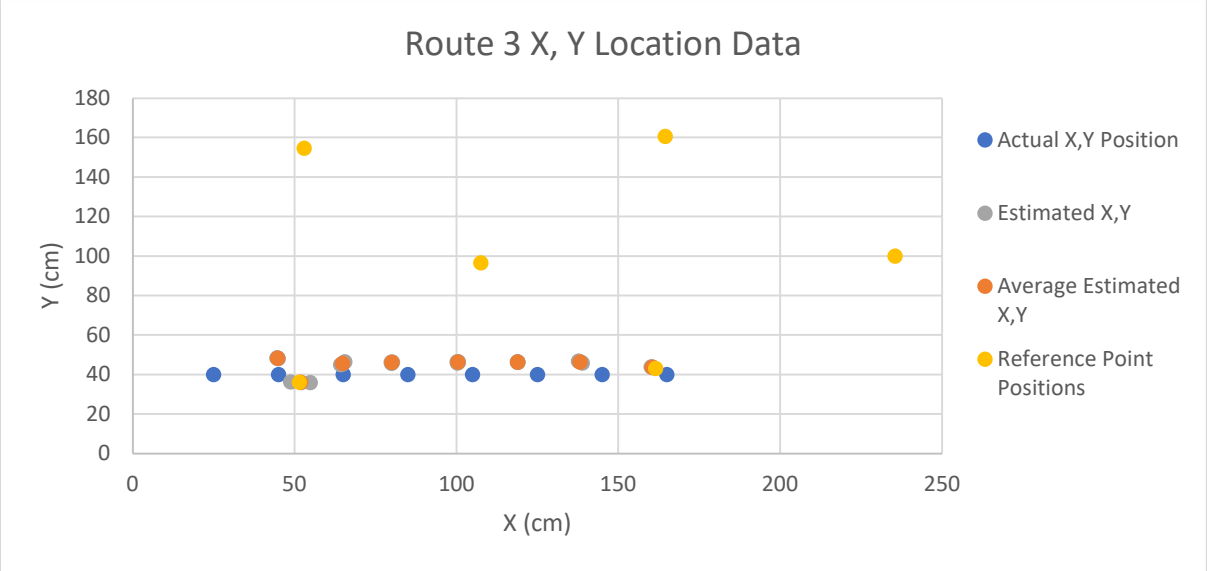

Figure 35

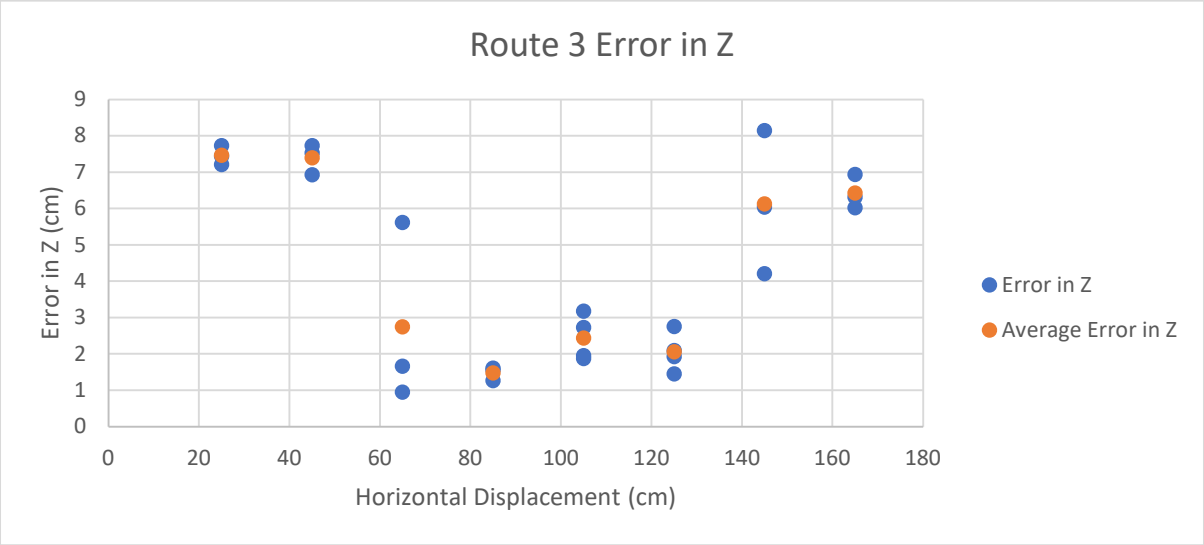

Figure 36

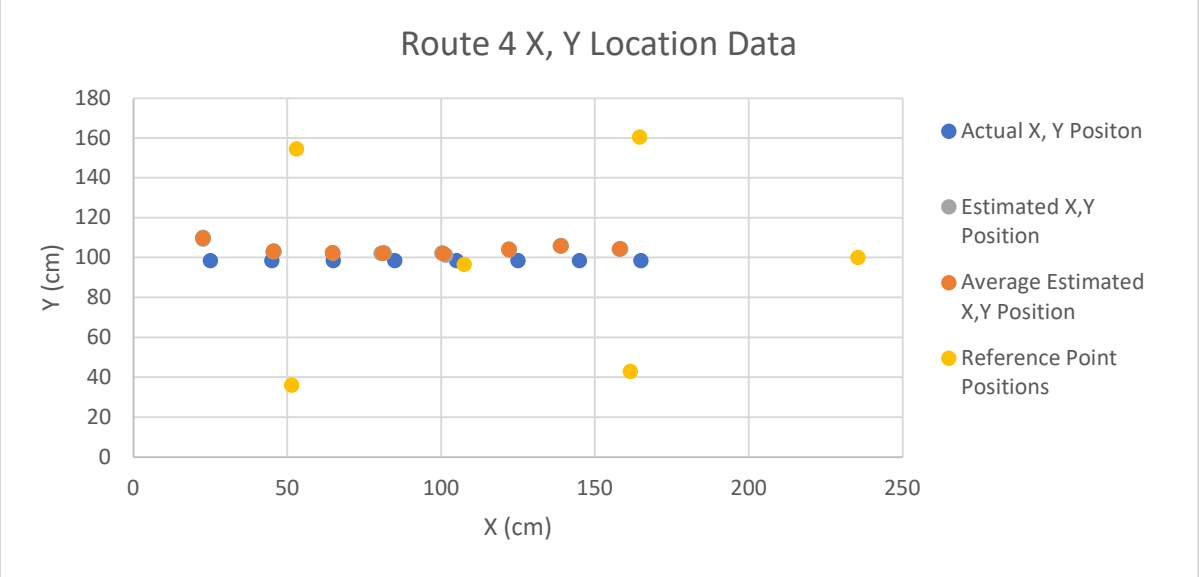

Figure 37

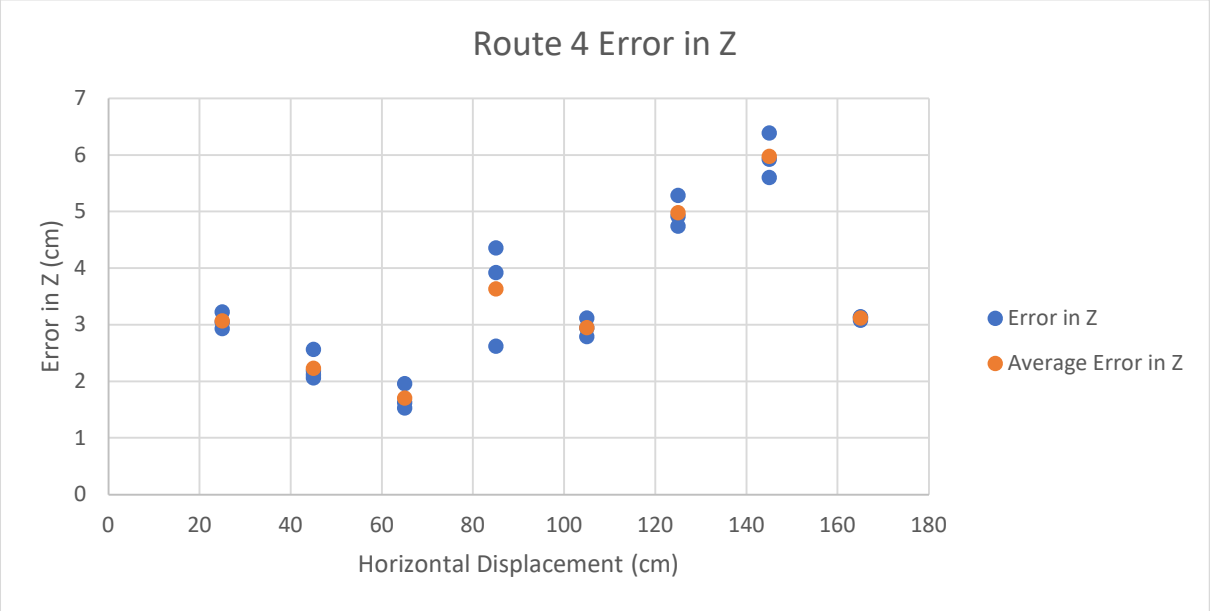

Figure 38

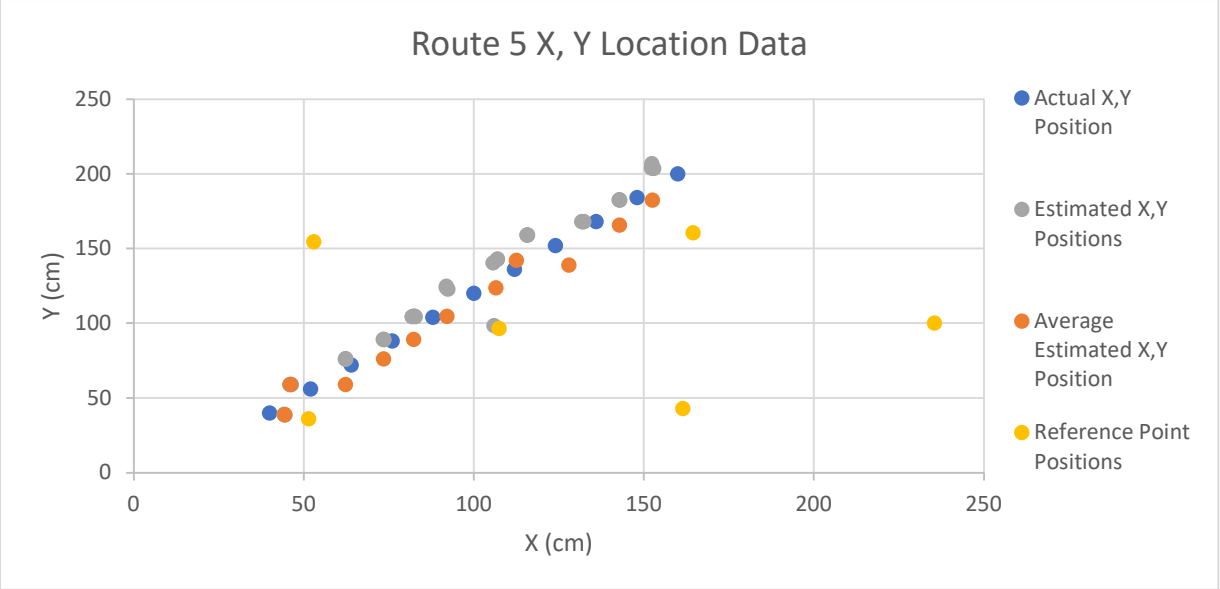

Figure 39

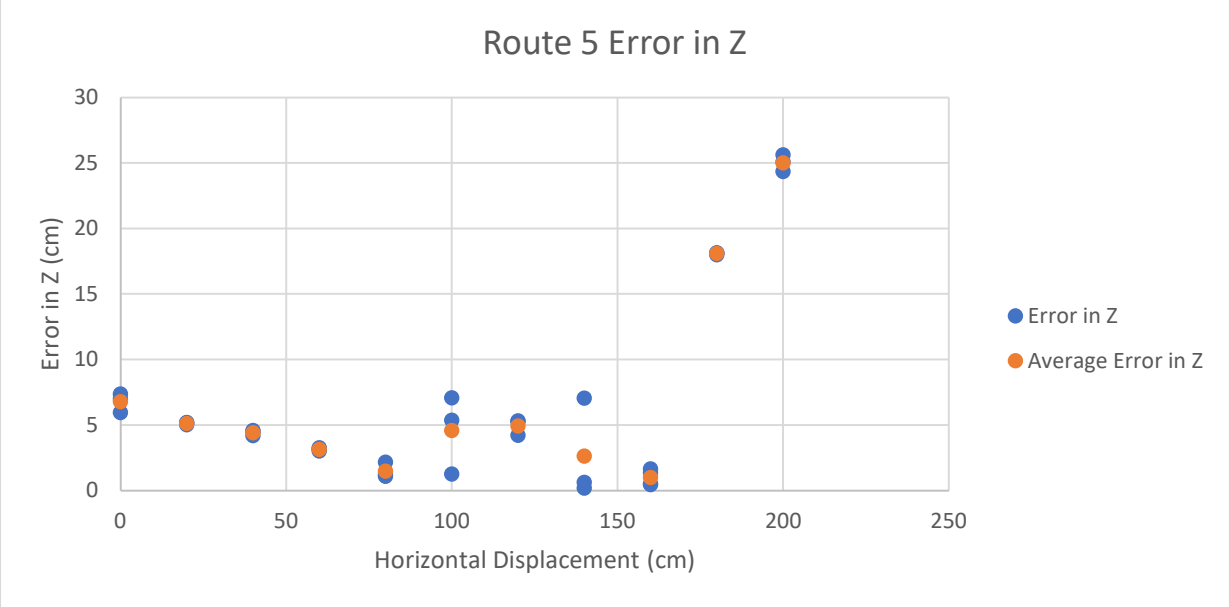

Figure 40

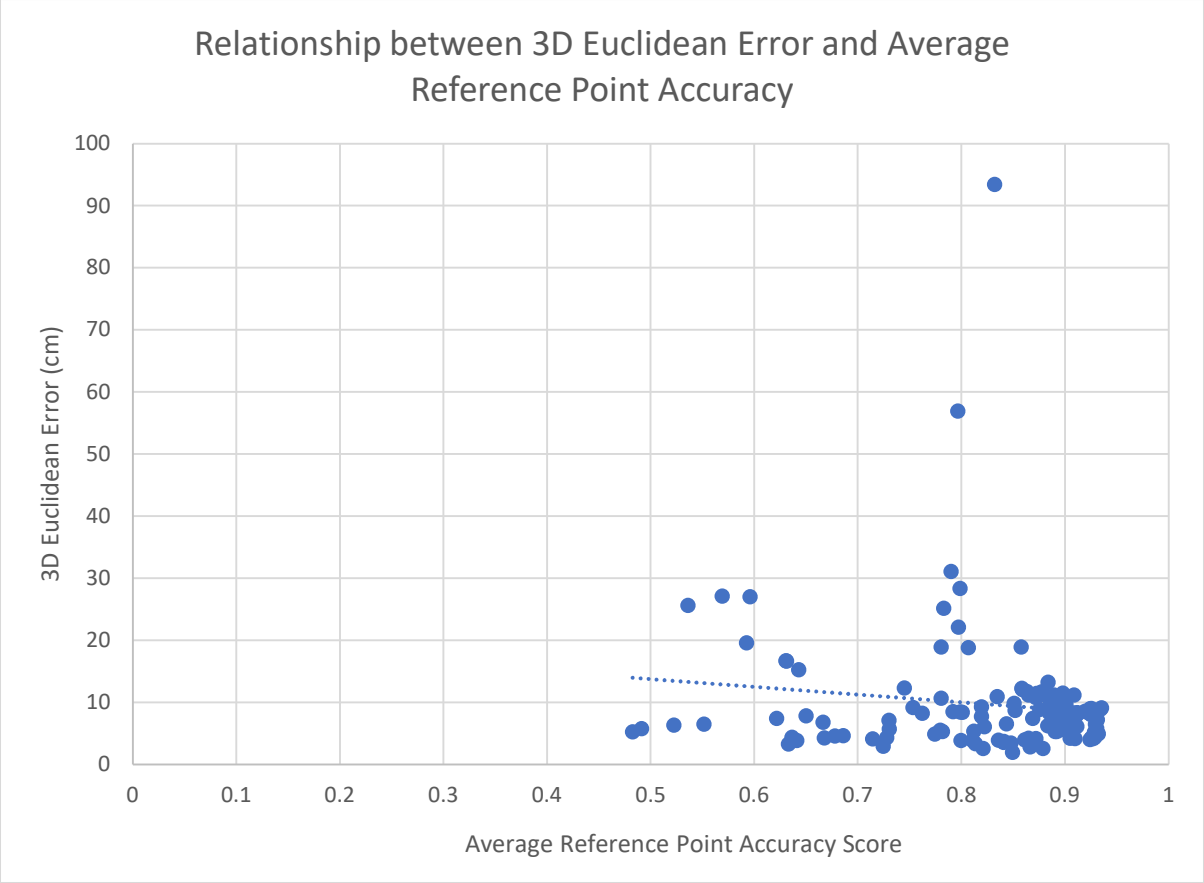

Figure 41

Routes	Include anomalies	Error in x	Error in y	Error In Z	3d Euclidean Error	Average Reference Point Average
Route 1	TRUE	2.8437488	3.930823	5.481163	8.151563	0.740372548
	FALSE	2.8437488	3.930823	5.481163	8.151563	0.740372548
Route 2	TRUE	4.9787755	3.047492	5.270151	8.603917	0.735713003
	FALSE	2.1872191	2.280636	2.393598	4.565516	0.731121904
Route 3	TRUE	6.8712956	6.013035	4.347471	11.15181	0.867356511
	FALSE	4.259702	6.279461	3.940405	8.928201	0.877380185
Route 4	TRUE	2.7875419	5.628462	3.408171	7.748555	0.905779764
	FALSE	2.7875419	5.628462	3.408171	7.748555	0.905779764
Route 5	TRUE	6.0244086	4.551306	6.859683	11.98042	0.84315037
	FALSE	4.9711937	2.6945	5.191068	8.570156	0.874890617
Average	TRUE	4.7011541	4.634224	5.073328	9.527253	0.818474439
	FALSE	3.4098811	4.162776	4.082881	7.592798	0.825909004

Table 1 X, Y testing Error Summary Table

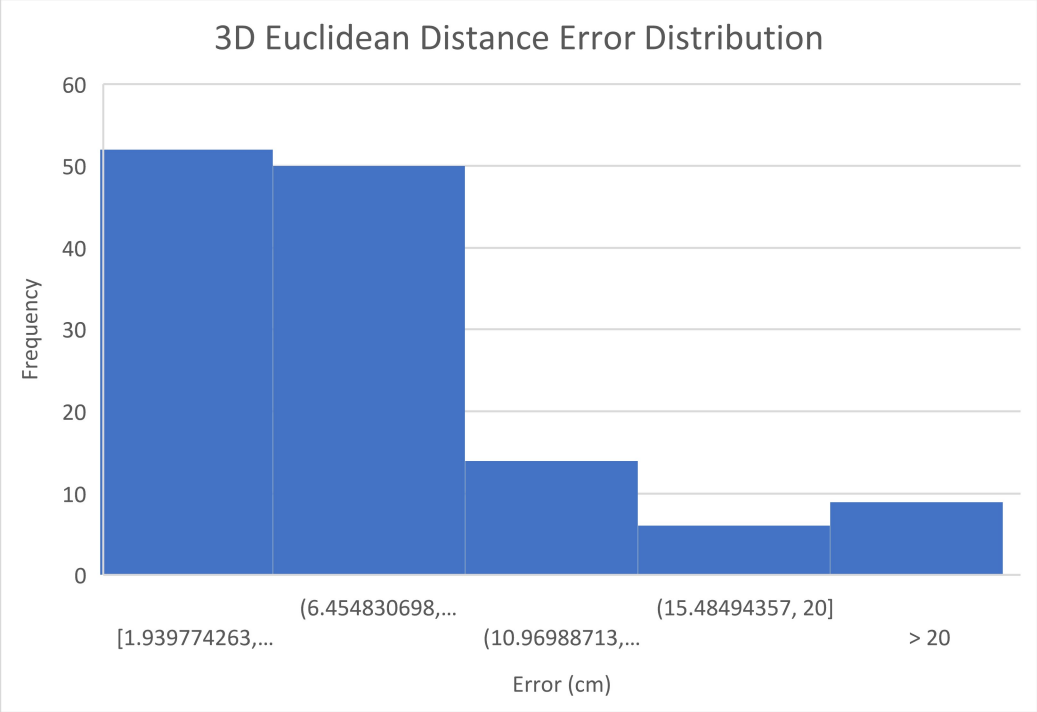
3D Euclidean Distance Error Distribution
60
50
40
30
20
10
0
Frequency
[1.939774263,...
(6.454830698,...
(10.96988713,...
(15.48494357, 20]
> 20
Error (cm)

Figure 42

6.2 Z-Depth Testing Results

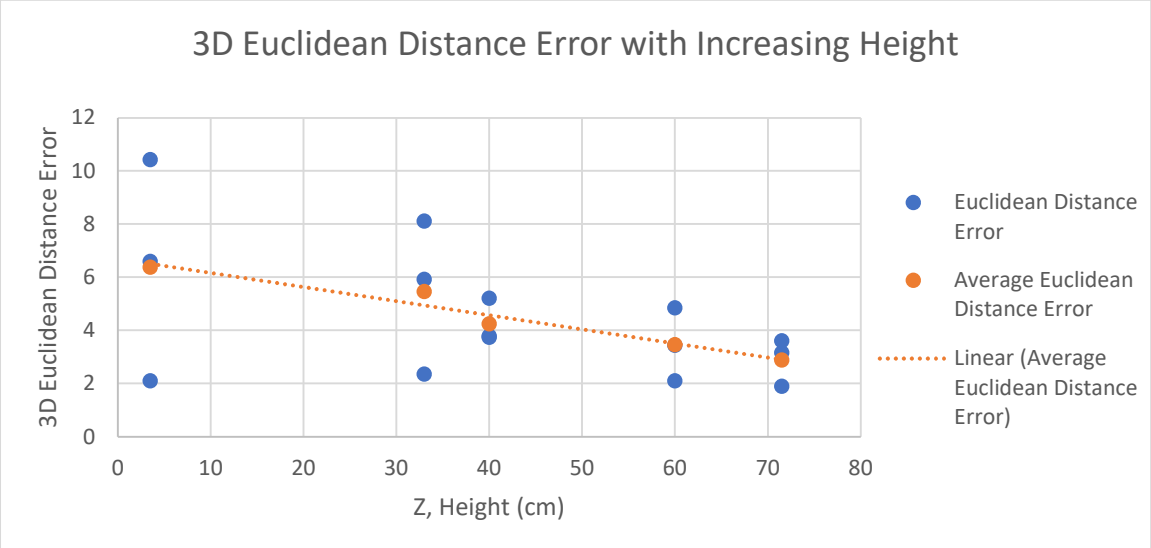

Figure 43

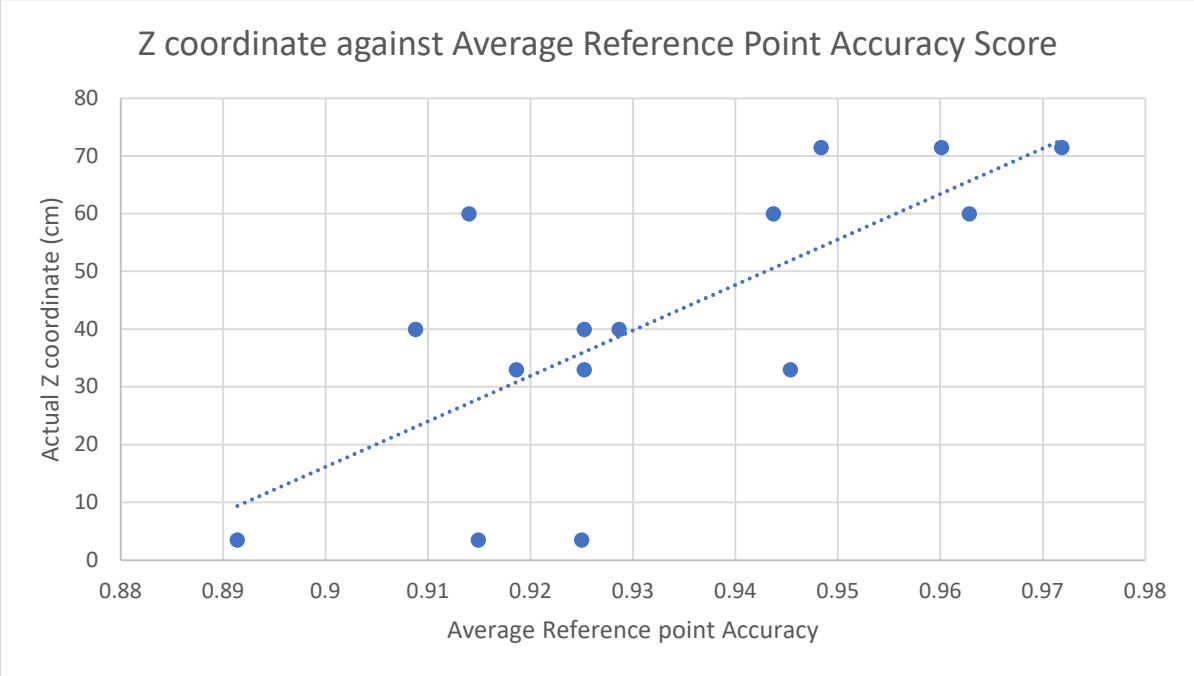

Figure 44

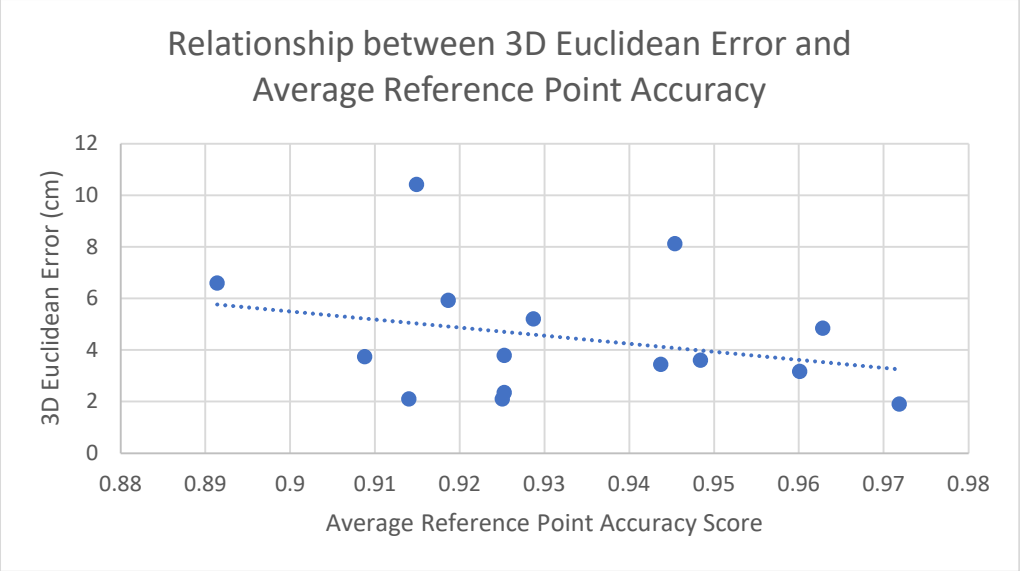

Figure 45

6.3 Code Performance Results

Code Section	Stationary	Moving
Time to Produce an output (s)	1.9747838	1.960497
Image Capture and Filter application Time (s)	0.0855564	0.08942
Image Detection Time (s)	1.8826367	1.863261
Data processing Time (s)	0.0007861	0.001217
Location Calculation Time (s)	0.0058045	0.006599

Table 2 Table of Software performance Metrics

7 Discussion

7.1 Result summary

7.1.1 X, Y Position Testing

Although this test was predominantly set up to analyse the accuracy of the triangulation algorithm in the X, Y plane it also gave good insight into how moving the device in the X, Y plane affected the z calculations.

Route 1 results seem to show the system to work well with an average error in the x and y under 4 cm and 5.5 cm in the z dimension. Observing figure 31, the positioning is particularly good in between the reference points, where y is between 50 and 150, with very close spacing of estimated values in this area. The result with the largest error seems to be at the point [25, 160] where there is an almost anomalous result with a Euclidean distance error of just under 20cm as shown in appendix 6. This poor result is potentially the outcome of the low identification accuracy score of one of the reference points, having less than 50% accuracy. The z error results show the accuracy of the depth measurement to increase as it reaches a position where both reference points are horizontally in line on opposite sides of the cameras view when it is in position [160, 100] in figures 31, and 32. This would be concurrent with the way it is calculated as mentioned in section 4.4.2, with the larger horizontal length between reference points and reduced amount of resolving required to gain the distance in the images x axis, lessening the uncertainty in the calculation.

Route 2 results shown in figures 33 and 34, shows the system to work well in both the X, Y plane and the z-depth calculations. There seems to be a single anomalous result associated with the position [89,100], from looking into the reference point accuracy there does not seem to be anything wrong with the lowest accuracy at just under 75%, this raises some concerns about whether it identified one of the reference points incorrectly. Excluding this value, this route performs far better than Route 1 showing the increased availability in reference points in its surrounding to have a positive effect on its accuracy, showing x y and z errors to be less than 3 cm as shown in table 1.

Analysing route 3, there seems to be an error in the y direction by approximately the same amount for nearly all the results. This could either be the effect of having all the reference points on one side of the image causing larger error in the algorithm. However as it is persistent in route 4 also it is most likely a calibration error. This y offset could therefore be a consequence of the camera being at a slight tilt in the positive y direction. The estimated values for the point [25,40] all seem to be incorrectly placed in the x direction, this could potentially be a blind spot for the camera where only a section of some of the reference points are visible in the image. This would explain the low average accuracy confidence score given by estimated values as seen in table 1, these lower scores increase the possibility of wrong identification of reference points which would cause an error as large as this. The z error for this route also seems to show a trend in which it decreases as the device becomes more in line with the reference point [107.5, 96.5] and then decreasing as it moves away from it, very similarly to the pattern shown in route 1.

On Route 4 the device performed its best out of all the tests with an overall Euclidean distance error of less than 8cm. The route shows some large acceptable errors in the X, Y plane at the point [25,98.5], associated with a low confidence accuracy level of one of the reference points, and has also been subjected to an offset in the positive y direction as mentioned previously. The strong performance in both the x and z coordinates, shows both the triangulation and z depth algorithms to work well in this routes reference point conditions. The z coordinates achievements looking to be the result of the availability of reference points existing on opposite ends of the images horizontal in all areas of the route.

Route 5 shows that the orientation of the camera does make the system more susceptible to error shown through its large number of anomalies and resultant high 3d Euclidean average error as seen in table 1. It seems to affect the z depth calculation the most, with the x and y coordinates less affected with their average errors being at the top end of the range displayed by the other routes. This can be most obviously illustrated in appendix 10 by the results at the point [160,200], the overall error showing all the estimations made to be anomalous. But looking into the results closer shows only the z coordinate to cause this large overall error and the x and y coordinates to be within acceptable levels. It must also be said at this point that the accuracy scores of two of the three reference points at this test coordinate are particularly low indicating a possible partial obstruction of reference points. However, the total average error achieved by this route is still well within the objective error range of ±20 cm showing it still to perform to satisfactory levels.

The outcome of these tests shows mostly positive results, demonstrating the system to work well in a range of different orientations and positions. The lowest error rates seem to be achieved when the device was in the middle of a reference point triangle as shown by the low error achieved by route 4 and 2 when ignoring anomalous results in table 1, aligning with the performance of most 3-point triangulation algorithms. It responded remarkably well to changes in the y direction with routes 1 and 2 showing the closest fit to the actual coordinates in which it was travelling along, despite their low average reference point accuracy score. Table 1 results also show the object detection algorithm to be susceptible to the orientation of the device with an obvious relationship showing the average reference point accuracy to be higher in the route 3 and 4 direction, lower in the 1 and 2 direction and route 5 being the intermediate. Additionally, figure 41 shows there to be a very shallow relationship between the overall error of the result and the average accuracy score of the reference points detection, showing that improving the object detection algorithm could decrease the overall error of the system. Overall, this testing has shown the system to perform well with it meeting the project accuracy objective of achieving an average error of ±20cm. Results showing that the overall average error including anomalies to be less than 10 cm as shown in table 1. And that less than 7% of the errors are greater than 20 cm as seen in figure 42, these errors being easily identified either by the reference point coverage or the accuracy levels of the object detection model, both of which have viable solutions to improve.

7.1.2 Z-Depth Testing

From the depth testing results it is clear to see that there is a relationship between the size of the z coordinate, and accuracy of results. As shown in figure 43 the 3d Euclidean distance between the actual point and the point estimated decreases as z increase and the device is moved vertically closer to the reference points. This is because as the device is lifted closer to the ceiling the reference points spread out further and further in the image, reducing the uncertainty in the centre position of the reference points and allowing more accurate angle calculations to be carried out, reducing the triangulation error. The same can be said for the z coordinate calculation the further the reference points are apart in the horizontal of the image the less estimation has to be done on the width of the image therefore increasing the accuracy of the depth perception. It is also noted as seen in figure 44, that on average the closer the device is to the ceiling the greater the confidence score of the object detection system. As also seen in the previous testing, Figure 45 confirms this shallow relationship between the average accuracy of the reference point identification and error in position.

7.1.3 Code Performance Testing

From the results shown in table 2, there does not seem to be any clear difference in the time to output results in both moving and stationary positions. This can be reflected in most of the profiling

sections apart from the data processing time with the moving average measurement looking to take a substantial proportion of time longer than that of the stationary. This could be to do with larger lists of potential reference points output by the object detection model when the device is moving and taking a not so focussed image, causing the increase in processing time.

Before seeing the result an idea to make sure the device was running as smoothly and as efficiently as possible was to use Numba [30] a python paralleliser to accelerate the codes performance. However, it is clear from table 2 that the bottleneck in the system speed is the object detection inference, it taking over 95% of the time to produce a result from the system. This leaving any subsequent parallelism introduced to have very little effect on runtime without improving this area. A very disappointing result considering the research, and hence requiring a significant improvement in this area to increase the performance of the system. For this to work practicality on a robot the current level of performance is most likely not acceptable, suggestions as to how this could be improved can be found in the next section.

7.2 Product Limitations

The previous section illustrates that the system has the potential to be very accurate in locating itself. However there are some aspects of the design which may become an issue when put into a practical environment.

One of these limitations would be the fact that it always needs three reference points in its vision. This means that the reference point spacing and the no obstruction of view of the camera is incredibly important. The current triangular reference point layout seemed to work for the most part but there were some issues which can be seen in the results where there were some ‘blind spots’ or only half reference points were visible opening up the system to larger error. There are of course many practical solutions to this issue including denser spacing of the reference points or using a camera with a larger field of view.

Another issue is the vertical distance the system is away from the reference points. As seen by the depth investigation there was an increase in error as the camera was moved further vertically away from the reference points. This could mean when implementing the system in a place where there are higher ceilings error could encroach. To mitigate this, the reference points could be mounted at lower positions in the room, i.e. if in a warehouse, on the edge of shelves. From a system design perspective there could also be an enhancement in the object detection algorithm, which could mean training the object detection model on a dataset where the reference points are further away from the camera getting it used to identifying objects further away.

The largest limitation found during the testing was the speed of the system to output results. As shown from the performance profiling it seems the system is outputting results approximately every 2 seconds. If the robot uses the positioning information for its navigation, this is going to be a key factor in determining its maximum safe speed around its working area. The profiling shows that the main bottleneck of the system is in the object detection system. This could be improved in a couple of ways including choosing a different model. Looking at TensorFlow’s model zoo [16] page shows the relative metrics of each model, the first model that could be tried would be the same type of model but one which takes smaller resolution images, the SSD MobileNet V2 FPNLite 320x320. The metrics show it to take nearly half as long as the current model to run inference at the expense of accuracy of the model. This however was not initially chosen due to the concern with the lack of resolution causing it to be less effective at recognising reference points at further distances away from the camera. Another option is to use an additional piece of hardware called an edge TPU [31] this is a specialised coprocessor for speeding up machine learning inference. It is low power, and

very high performing with a capability to perform 4 trillion operations per second, using 0.5 watts of power. How this translates to the speed up of the model depends on a couple of factors such as the host CPU and USB bus speed, but with the benchmarks set on the coprocessors website it looks to have the potential to speed up the inference speed by 10-fold, analysing the MobileNet v2 SSD (224x224) results [32].

8 Appreciation for Engineering Uncertainty

Testing has showed that error or uncertainty to some degree always creeps into the output of the system. The sources of which are going to be suggested in this section.

Many measurements have been taken in this project. These involve measuring the dimensions of the test environment, measuring where the reference points lie on the ceiling, and measuring the current position of the device during testing. All of which will contain some form of uncertainty, as a tape measure was used to take all these measurements this shows there to be an uncertainty in these measurements of ±0.05cm. However this is making large assumptions about the environment and the way in which these measurements were taken. For example, when measuring the dimensions of the room it was assumed that all walls are square, and the floor and ceiling are all completely horizontal. Whereas this is probably not the case even in modern construction, causing further uncertainty.

Another source of uncertainty and error is in the position of the camera. A slight tilt in the camera has already shown to cause an offset error as seen in routes 3 and 4 of the X, Y position testing. The calibration tests certainly would have helped to reduce this error, however with the lack of correct equipment it was hard to consistently reduce this uncertainty to a degree in which it was unnoticeable. If this were to be mounted on a robot, the camera would be placed in a gyroscopic frame to maintain its parallel orientation to the ceiling.

Finally, another source of error in the calculations could be in the sourcing of the centre of the reference points from the images. The object detection model will not always gain the bounding box coordinates which fits around the exact dimensions of the reference points due to inaccuracies of the model and the varying vertical distance and orientations of the camera. This could mean when the code locates the centre of the bounding box it does not find where the centre of the reference point actually is. Therefore, there is a slight offset giving an error in the reading and uncertainty in the calculations thereafter. This could be mitigated by training the model better to produce more accurate bounding boxes around the reference points. It could also be improved by using square reference points making them less susceptible to when the camera images them at different orientations.

9 Conclusion

This paper has explored the use of optical imaging principles in creating an accurate compact Indoor self-positioning system. The system consisting of raspberry pi and pi camera, using TensorFlow's deep learning object detection API as a reference point detection system and a series of geometric calculations to locate the device' final location.

From a lower-level perspective, this project saw the use of basic image processing techniques to remove noise from images, so its useful information was more easily identifiable. It also saw the successful creation of a custom dataset of the reference points using labelImg. This dataset was then used to train one of TensorFlow's pre-made object detection models to effectively pick out reference points in images taken by the pi camera. This reference point detection system was then implemented into a program on the raspberry pi, in which geometric calculations were carried out on the reference point data. These calculations helped gain appropriate distance and angle data to be able to apply a geometric circle intersection triangulation algorithm and scale factor depth calculation method which were used to compute the final x, y and z coordinates of the device.

The outcome showed a positioning system that could locate itself within 20 cm, satisfying one of the projects main aims, with the average error being just under ±10cm. System behaviours were also identified with the output location value seeming to be the most accurate when surrounded by reference points, although still able to locate to acceptable levels when not. It also showed that the output location has less error, and the accuracy scores of the reference point detection system improves when the device is vertically closer to the reference points providing it can still see three full reference points. Furthermore, the paper also highlights a location accuracy dependence on the reference point detection performance, with a lower reference point detection accuracy score typically meaning a larger error in the output location.

Testing also identified that there is still improvement to be made in the system speed. The system currently produces results approximately at 2 second intervals – with the objective of the system to be able to produce results in real-time this area still needs improvement. However, with the bottleneck already identified and many improvement suggestions of future work throughout the report, it is believed that dramatic enhancements could be made in this area in a short time frame.

The system also has some clear advantages over others, one being the systems minimal hardware to work. Unlike other forms of indoor positioning such as the Wi-Fi, Ultra-Wide-band, and Acoustic based methods previously mentioned, it only needs to be able to see the reference points above it which could be as simple as the numbers used in this project written on pieces of paper. This makes it very easy and cheap to implement with no wiring or electronic infrastructure required for the system, meaning no major construction is required during installation. The system could also be incredibly scalable with the reference point design, including its spacing and pattern, easily made modular meaning the same level error and performance can be expected from the system in larger area with similar vertical ceiling heights. Finally, it has been cheap to produce with very little specialised equipment. Working on the project for approximately three months and with the average electrical engineer wage being approximately £30,278 [33], it can be estimated that the current development cost has been £7,569 plus the combined cost of the raspberry pi [34] and the camera module [35] being £78.

Therefore, this project has highlighted a successful proof of concept that a compact, inexpensive, easy to install self-positioning system can be implemented using off the shelf components, opensource supervised learning code and a set of geometric techniques.

10 Future Work

From analysing the performance of the system there are two main areas that would cause a significant improvement in the system. The most prominent being the speed of the reference point detection and the second being implementing a smoothing algorithm on the results.

As mentioned previously there are a few ways the inference time can be improved. I believe the most promising method would be to implement the edge TPU, even though this adds extra hardware cost, I believe it would cause a sufficient improvement justifying its use on a robot. Further investigation into the choice of model could also be beneficial in optimising this part of the system, with frequent updates in TensorFlow's backend this process is only going to get faster.

From position testing it is evident that taking the average of the three results, reduced the error of the reading and dampened the effect of any anomalous estimation. Hence, the systems output accuracy would benefit from further investigation into implementing an algorithm which does this for each output such as a sliding average filter [36] or a Kalman Filter [37]. However, this would only be beneficial once the speed of the output was improved otherwise it would cause further latency.

11 References

[1] A. H. K. M. S. B. Youssef N. Naggar, "A Low Cost Indoor Positioning System Using Computer Vision," *International Journal of Image, Graphics and Signal Processing,* vol. 11, no. 4, p. 19, 2019.

[2] K. Cengiz, "Comprehensive Analysis on Least-Squares," *IEEE INTERNET OF THINGS JOURNAL,* vol. 8, no. 4, p. 15, 2021.

[3] "Global Indoor Positioning and Navigation Market (2020 to 2025) - Featuring Google, Qualcomm & Microsoft Among Others - ResearchAndMarkets.com," businesswire, 15 06 2020. [Online]. Available: https://www.businesswire.com/news/home/20200615005247/en/Global-Indoor-Positioning-and-Navigation-Market-2020-to-2025---Featuring-Google-Qualcomm-Microsoft-Among-Others---ResearchAndMarkets.com. [Accessed 05 05 2021].

[4] L. Morlas, "All about Indoor Positioning Systems (IPS)," Bitbrain, 27 5 2020. [Online]. Available: https://www.bitbrain.com/blog/indoor-positioning-system. [Accessed 5 5 2021].

[5] B. Ray, "How An Indoor Positioning System Works," Air Finder by Link Labs, 16 8 2018. [Online]. Available: https://www.airfinder.com/blog/indoor-positioning-system. [Accessed 5 5 2021].

[6] M. A. A. K. R. Cem Sertatıl a, "A novel acoustic indoor localization system employing CDMA," *Digital Signal Processing,* vol. 22, no. 3, pp. 506-517, 2012.

[7] F. S. a. A. S. a. P. Dutta, "A Review of Object Detection Models based on Convolutional Neural Network," *ArXiv,* 2019.

[8] M. Alatortsev, "Object detection without Machine Learning," towards data science, 5 2 2018. [Online]. Available: https://towardsdatascience.com/object-detection-without-machine-learning-aed3c5b668f3. [Accessed 5 5 2021].

[9] J. Bullock, "Real-time Object Detection Without Machine Learning," towards data science, 24 11 2019. [Online]. Available: https://towardsdatascience.com/real-time-object-detection-without-machine-learning-5139b399ee7d. [Accessed 5 5 2021].

[10] I. Biswas, "Triangulation vs Trilateration vs Multilateration – for Indoor Positioning Systems," Path Partner, 6 6 2019. [Online]. Available: https://www.pathpartnertech.com/triangulation-vs-trilateration-vs-multilateration-for-indoor-positioning-systems/. [Accessed 5 5 2021].

[11] J. A. B. Josep M.Font-Llagunes, "Consistent triangulation for mobile robot localization using discontinuous angular measurements," *Robotics and Autonomous Systems,* vol. 57, p. 931–942, 2009.

[12] W. L. L. X. Z. Yu Zhou, "Single-Camera Trilateration," *applied Sciences,* 2019.

[13] C. R. L. D. C. Y. W. D. Xiao A, "An Indoor Positioning System Based on Static Objects in Large Indoor Scenes by Using Smartphone Cameras," *Sensors,* 2018.

[14] C. C. a. F. V. Koss, “A Comprehensive Study of Three Object Triangulation,” University of Michigan, Department of Electrical Engineering and Computer Science, Michigan.

[15] TensorFlow, “TensorFlow,” TensorFlow, [Online]. Available: https://www.tensorflow.org/. [Accessed 5 5 2021].

[16] TensorFlow, “TensorFlow 2 Detection Model Zoo,” TensorFlow, [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md. [Accessed 5 5 2021].

[17] TensorFlow, “For Mobile & IoT,” TensorFlow, [Online]. Available: https://www.tensorflow.org/lite. [Accessed 5 5 2021].

[18] COCO, “Common Objects in Context,” COCO, [Online]. Available: https://cocodataset.org/#home. [Accessed 5 5 2021].

[19] TensorFlow, “Datasets -mnist,” TensorFlow, [Online]. Available: https://www.tensorflow.org/datasets/catalog/mnist. [Accessed 5 5 2021].

[20] YunYang1994, “yymnist,” 26 8 26. [Online]. Available: https://github.com/YunYang1994/yymnist. [Accessed 5 5 2021].

[21] J. Brownlee, “How to Develop a CNN for MNIST Handwritten Digit Classification,” Machine Learning Mastery, 24 8 2020. [Online]. Available: https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/. [Accessed 5 5 2021].

[22] tzutalin, “labelImg,” [Online]. Available: https://github.com/tzutalin/labelImg. [Accessed 5 5 2021].

[23] “Training Custom Object Detector,” TensorFlow 2 Object Detection API tutorial, [Online]. Available: https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html. [Accessed 5 5 2021].

[24] TensorFlow, “TensorBoard,” TensorFlow, [Online]. Available: https://www.tensorflow.org/tensorboard. [Accessed 5 5 2021].

[25] TensorFlow, “TensorFlow Lite,” [Online]. Available: https://www.tensorflow.org/lite/guide#4_optimize_your_model_optional. [Accessed 5 5 2021].

[26] “FlatBuffers,” [Online]. Available: https://google.github.io/flatbuffers/. [Accessed 5 5 2021].

[27] TensorFlow, “Post-training quantization,” TensorFlow, [Online]. Available: https://www.tensorflow.org/lite/performance/post_training_quantization. [Accessed 5 5 2021].

[28] armaanpriyadarshan, “TensorFlow-2-Lite-Object-Detection-on-the-Raspberry-Pi,” 22 11 2020. [Online]. Available: https://github.com/armaanpriyadarshan/TensorFlow-2-Lite-Object-Detection-on-the-Raspberry-Pi/blob/main/TFLite-PiCamera-od.py. [Accessed 5 5 2021].

[29] M. V. D. Vincent Pierlot, “A New Three Object Triangulation Algorithm for Mobile Robot Positioning,” *IEEE TRANSACTIONS ON ROBOTICS,* vol. 30, no. 3, pp. 566-577, 2014.

[30] “Numba,” Numba, [Online]. Available: https://numba.pydata.org/. [Accessed 6 5 2021].

[31] coral, “USB Accelerator,” coral, [Online]. Available: https://coral.ai/products/accelerator/. [Accessed 5 5 2021].

[32] Coral, “Edge TPU performance benchmarks,” Coral, [Online]. Available: https://coral.ai/docs/edgetpu/benchmarks/. [Accessed 5 5 2021].

[33] payscale, “Average Electronics Engineer Salary in United Kingdom,” payscale, [Online]. Available: https://www.payscale.com/research/UK/Job=Electronics_Engineer/Salary. [Accessed 5 5 2021].

[34] “Raspberry Pi 4 Model B,” The Pi Hut, [Online]. Available: https://thepihut.com/collections/raspberry-pi/products/raspberry-pi-4-model-b. [Accessed 5 5 2021].

[35] “Raspberry Pi Camera Module V2,” The Pi Hut, [Online]. Available: https://thepihut.com/products/raspberry-pi-camera-module. [Accessed 5 5 2021].

[36] P. T. Heaver, “Smoothing,” Department of Chemistry and Biochemistry, The University of Maryland at College Park, 3 2021. [Online]. Available: https://terpconnect.umd.edu/~toh/spectrum/Smoothing.html. [Accessed 5 5 2021].

[37] H. B. Youngjoo Kim, “Introduction to Kalman Filter and Its Applications,” *Introduction and Implementations of the Kalman Filter,* 2018.

12 Appendices

Appendix 1: The Total Algorithm [29]

Given the three beacon/reference points coordinates $\{x_1, y_1\}$, $\{x_2, y_2\}$, $\{x_3, y_3\}$, and the three angles φ_1, φ_2, φ_3:

1. compute the modified beacon coordinates:

$$x'_1 = x_1 - x_2, y'_1 = y_1 - y_2,$$

$$x'_3 = x_3 - x_2, y'_3 = y_3 - y_2,$$

2. compute the three $\cot(.)$:

$$T_{12} = \cot(\varphi_2 - \varphi_1), \quad T_{23} = \cot(\varphi_3 - \varphi_2),$$

$$T_{31} = \frac{1 - T_{12}T_{23}}{T_{12} + T_{23}},$$

3. compute the modified circle centre coordinates:

$$x'_{12} = x'_1 + T_{12}\, y'_1, y'_{12} = y'_1 - T_{12}\, x'_1,$$

$$x'_{23} = x'_3 - T_{23}\, y'_3, y'_{23} = y'_3 + T_{23}\, x'_3,$$

$$x'_{31} = (x'_3 + x'_1) + T_{31}\,(y'_3 - y'_1),$$

$$y'_{31} = (y'_3 + y'_1) - T_{31}\,(x'_3 - x'_1),$$

4. compute k'_{31}:

$$k'_{31} = x'_1 x'_3 + y'_1 y'_3 + T_{31}(x'_1 y'_3 - x'_3 y'_1),$$

5. compute D (if $D = 0$, return with an error):

$$D = (x'_{12} - x'_{23})(y'_{23} - y'_{31}) - (y'_{12} - y'_{23})(x'_{23} - x'_{31}),$$

6. compute the robot position $\{x_R, y_R\}$ and return:

$$x_R = x2 + \frac{k'31(y'12 - y'23)}{D}, y_R = y2 + \frac{k'31(x'23 - x'12)}{D}$$

Algorithm 2 Final version of the *ToTal* algorithm.

In these special cases in which the indexes {*i*, *j*, *k*} have to be replaced by {1, 2, 3}, {3, 1, 2}, or {2, 3, 1} if $\varphi_{31} = 0$, $\varphi_{23} = 0$, or $\varphi_{12} = 0$ respectively.

Appendix 2: BenchMarks of the Total Algorithm against other trangulation algorithms. The test consisted of 10^6 runs of each algorithm at random locations of the same square shaped error analysis. The average time per run and the number of basic matmatical operations were also counted per algorithm. [29]

	Algorithm	+	×	/	$\sqrt{x}$	*trigo*	time (*s*) [T]
[46, 47]	ToTal [1]	30	17	2	**0**	**2**	**0.163**
[38]	Ligas [1]	29	22	2	**0**	**2**	0.171
[29]	Font-Llagunes [1]	23	17	2	**0**	5	0.223
[42]	Cassini [2]	**19**	**8**	3	**0**	4	0.249
[5]	Cohen [1]	37	15	3	2	4	0.272
[2]	Easton [2]	22	24	**1**	**0**	5	0.298
[9]	McGillem [1]	37	18	5	2	8	0.340
[17]	Hmam [2]	29	11	3	3	9	0.428
[5]	Cohen [2]	26	11	3	2	11	0.437
[34]	Esteves [2]	43	14	2	2	11	0.471
[42]	Collins [2]	34	10	2	2	11	0.485
[9]	McGillem [2]	29	9	3	2	11	0.501
[42]	Kaestner [2]	28	10	3	2	11	0.504
[45]	Tsukiyama [1]	52	22	3	5	14	0.596
[16]	Casanova [1]	52	21	4	5	14	0.609
[32]	Tienstra [2]	33	18	8	3	9	0.640
[31]	Font-Llagunes [1]	62	25	6	1	8	0.648
[8]	Madsen [2]	38	24	5	3	15	0.707

[T] For 10^6 executions on an Intel(R) Core(TM) i7 920 @ 2.67GHz.

[1] Geometric circle intersection [2] Trigonometric solution

Table 1 Comparison of various triangulation algorithms to our *ToTal* algorithm.

Appendix 3: The Special Case Total algorithm when $\varphi_{ki} = 0 \vee \varphi_{ki} = \pi$

Given the three beacon coordinates $\{x_i,\ y_i\}$, $\{x_j,\ y_j\}$, $\{x_k,\ y_k\}$, and the three angles φ_i, φ_j, φ_k:

1. compute the modified beacon coordinates:

$$x'_i = x_i - x_j,\ y'_i = y_i - y_j,$$

$$x'_k = x_k - x_j,\ y'_k = y_k - y_j,$$

2. compute $T_{ij} = \cot(\varphi_j - \varphi_i)$,

3. compute the modified circle centre coordinates:

$$x'_{ij} = x'_i + T_{ij}\, y'_i,\ y'_{ij} = y'_i - T_{ij}\, x'_i,$$

$$x'_{jk} = x'_k + T_{ij}\, y'_k,\ y'_{jk} = y'_k - T_{ij}\, x'_k,$$

$$x'_{ki} = (y'_k - y'_i),\ y'_{ki} = (x'_i - x'_k),$$

4. compute $k'_{ki} = (x'_i y'_k - x'_k y'_i)$,
5. compute D (if $D = 0$, return with an error):

$$D = (x'_{jk} - x'_{ij})(y'_{ki}) + (y'_{ij} - y'_{jk})(x'_{ki}),$$

6. compute the robot position $\{x_R,\ y_R\}$ and return:

$$x_R = x_j + \frac{k'_{ki}(y'_{ij} - y'_{jk})}{D}, y_R = y_j + \frac{k'_{ki}(x'_{jk} - x'_{ij})}{D}$$

Algorithm 3 Special case $\varphi_{ki} = 0 \vee \varphi_{ki} = \pi$.

Appendix 4: Code Implementation of the ToTal Algorithm

```
# ToTal algorithm in the case when the subtraction of two of the angles = 0 or pi
def ToTal_Algorithm_Special_Cases(refCoords):
    # compute modified beacon coordinates
    x_i = refCoords[0][0] - refCoords[1][0]
    y_i = refCoords[0][1] - refCoords[1][1]
    x_k = refCoords[2][0] - refCoords[1][0]
    y_k = refCoords[2][1] - refCoords[1][1]

    # compute Tij
    Tij = mpmath.cot(refCoords[1][2] - refCoords[0][2])

    # compute the modified circle center coordinates
    x_ij = x_i + Tij*y_i
    y_ij = y_i - Tij*x_i
    x_jk = x_k + Tij*y_k
    y_jk = y_k - Tij*x_k
    x_ki = y_k - y_i
    y_ki = x_i - x_k

    # compute k_ki
    k_ki = x_i*y_k - x_k*y_i

    #compute D
    D=(x_jk - x_ij)*y_ki + (y_ij - y_jk)*x_ki

    # compute the robot position
    xR = refCoords[1][0] + (k_ki*(y_ij - y_jk))/D
    yR = refCoords[1][1] + (k_ki*(x_jk - x_ij))/D
    return xR, yR
```

Appendix 5: z-Depth testing Results

Depth Testing															
Angle	Actual Position			Estimated Position				Error in			3d Euclidean Error	Reference point Accuracy			
	X	Y	z	x	y		z	x	y	Z		1	2	3	Average Reference Point confidence Score
0	150	100	3.5	141.6569	98.95266	234.1556	-2.65559	8.343081	1.047345	6.155586	10.4209	0.933331	0.896893	0.725664	0.85196278
45	150	100	3.5	149.4761	100.4047	229.9948	1.505225	0.523942	0.404732	1.994775	2.101773	0.931577	0.882993	0.582037	0.79886925
90	150	100	3.5	144.5197	97.43747	230.6319	0.8681	5.480346	2.562528	2.6319	6.597548	0.937137	0.895293	0.774171	0.868867003
0	150	100	33	145.9237	95.48401	203.8741	27.62593	4.076253	4.515991	5.374066	8.117302	0.930295	0.921528	0.906354	0.91939225
45	150	100	33	146.2419	99.78377	203.0771	28.42294	3.758078	0.216226	4.57706	5.926161	0.934911	0.9022	0.887937	0.9083495
90	150	100	33	152.0474	99.30257	199.4324	32.06756	2.04739	0.697433	0.932441	2.355349	0.917161	0.909583	0.90902	0.911921467
0	150	100	40	144.7956	99.89291	191.6697	39.83028	5.204386	0.107092	0.169715	5.208253	0.941291	0.885387	0.875766	0.9008145
45	150	100	40	148.4152	99.79992	194.8874	36.61262	1.584797	0.200078	3.387379	3.745124	0.92555	0.905585	0.882551	0.90456227
90	150	100	40	148.08	97.16359	193.1226	38.3774	1.920005	2.836413	1.622601	3.790052	0.922484	0.906346	0.905857	0.911562
0	150	100	60	146.3409	103.1504	171.0842	60.41578	3.659138	3.150436	0.415784	4.846382	0.959279	0.902989	0.867004	0.909757347
45	150	100	60	149.5951	101.9029	174.3336	57.16639	0.404856	1.902919	2.833615	3.437205	0.957549	0.919058	0.90515	0.927252143
90	150	100	60	150.6341	99.56931	173.4566	58.04339	0.634086	0.43069	1.95661	2.1014	0.963108	0.876682	0.875746	0.905178903
0	150	100	71.5	146.8355	99.52519	161.6564	69.84357	3.164466	0.47481	1.65643	3.6032	0.936421	0.912146	0.84085	0.896472417
45	150	100	71.5	149.3516	101.8972	162.4515	69.0485	0.648364	1.897171	2.451498	3.166935	0.932169	0.873079	0.850809	0.8853523
90	150	100	71.5	149.4297	98.86975	161.4131	70.08693	0.570259	1.130253	1.413069	1.897217	0.933539	0.876019	0.866996	0.89218457

Appendix 6: X, Y Route 1 test Results table

ROUTE 1														
Actual Position			Estimated Position								Reference point Accuracy			
x	y	z	x	y		z	Error in x	Error in y	Error In Z	3d Euclidean Error	1	2	3	Reference Point confidence
160	25	3.5	165.9513	24.5461	235.8037	-4.303707431	5.951344	0.453896	7.803707	9.824579859	0.920592	0.822788	0.80964	0.851006497
160	25	3.5	148.182	10.42916	233.6738	-2.17379754	11.81798	14.57084	5.673798	19.60015007	0.855849	0.556107	0.366245	0.592733747
160	25	3.5	158.6713	28.72952	234.7739	-3.27388173	1.328703	3.729517	6.773882	7.846032203	0.84587	0.580324	0.524223	0.65013911
160	50	3.5	161.3437	42.41682	231.3329	0.167128472	1.343687	7.583183	3.332872	8.391554745	0.812763	0.805483	0.784611	0.80095251
160	50	3.5	159.4734	51.31132	235.2979	-3.797856738	0.526635	1.311323	7.297857	7.433412787	0.898128	0.522694	0.444009	0.621610227
160	50	3.5	159.7073	50.62992	234.738	-3.237964914	0.292664	0.629923	6.737965	6.773671564	0.89261	0.59828	0.508464	0.666451443
160	75	3.5	162.0896	76.71092	233.4617	-1.961694837	2.089618	1.710921	5.461695	6.092935609	0.954212	0.827935	0.684732	0.822292843
160	75	3.5	160.3279	74.48203	233.4524	-1.952414028	0.327946	0.51797	5.452414	5.486771369	0.83651	0.833672	0.669019	0.779733973
160	75	3.5	160.3093	73.87424	233.1635	-1.663461341	0.309327	1.125759	5.163461	5.293802897	0.860258	0.827806	0.657357	0.78180715
160	100	3.5	162.5449	106.2298	225.7094	5.790587758	2.544945	6.229849	2.290588	7.108766172	0.923573	0.815006	0.451993	0.730190293
160	100	3.5	159.7958	101.2148	230.2664	1.233577853	0.204239	1.214849	2.266422	2.579581362	0.92291	0.915433	0.798279	0.878873833
160	100	3.5	159.6371	100.3836	230.7577	0.742264637	0.362857	0.383618	2.757735	2.807834136	0.918654	0.871161	0.809499	0.866437923
160	125	3.5	161.8988	127.362	232.8667	-1.366718315	1.898822	2.36198	4.866718	5.733185845	0.896445	0.890995	0.404345	0.73059492
160	125	3.5	160.0063	125.6261	229.8359	1.664060568	0.006269	0.626108	1.835939	1.939774263	0.884149	0.841833	0.822147	0.84937651
160	125	3.5	159.6645	123.4992	237.0241	-5.524100292	0.335525	1.50077	9.0241	9.154194334	0.898872	0.83032	0.531131	0.753440887
160	150	3.5	151.4832	155.7524	236.3498	-4.849780713	8.516771	5.75239	8.349781	13.2417607	0.895621	0.893128	0.862126	0.883625023
160	150	3.5	160.1613	152.6201	229.2524	2.247575333	0.161275	2.620094	1.252425	2.908516433	0.815863	0.772131	0.58534	0.724444763
160	150	3.5	161.7112	152.7482	228.7084	2.79164667	1.711169	2.748215	0.708353	3.313992973	0.72892	0.653547	0.516477	0.632981323
160	175	3.5	164.3379	172.8765	228.8177	2.682278329	4.33788	2.123479	0.817722	4.898472796	0.837813	0.779922	0.705751	0.77449528
160	175	3.5	163.8221	176.5157	229.2584	2.241598155	3.82211	1.51567	1.258402	4.299925092	0.881554	0.671903	0.630334	0.727930447
160	175	3.5	163.8019	177.6458	239.3925	-7.892466866	3.801884	2.645781	11.39247	12.29808047	0.854533	0.781835	0.598534	0.74496718
160	200	3.5	165.0609	187.7669	235.586	-4.085962954	5.060889	12.23306	7.585963	15.25802269	0.935667	0.797877	0.195564	0.643035867
160	200	3.5	165.7731	189.6623	239.6827	-8.182698625	5.773099	10.33772	11.6827	16.63377676	0.929446	0.766674	0.197761	0.631293543
160	200	3.5	165.7243	189.5872	239.7609	-8.260941263	5.724335	10.41285	11.76094	16.71870709	0.934814	0.756037	0.200727	0.630525857
					Anomolous Average		2.843749	3.930823	5.481163	8.151562592				0.740372548
					Anomoly Free Average		2.843749	3.930823	5.481163	8.151562592				0.740372548

Appendix 7: X, Y Route 2 test Results table

ROUTE 2														
Actual Position			Estimated Position							Reference point Accuracy				
x	y	z	x	y		z	Error in x	Error in y	Error In Z	3d Euclidean Error	1	2	3	Average Reference Point confidence Score
89	25	3.5	91.57648	29.42673	231.6943	-0.194348819	2.57648	4.426727	3.694349	6.315248936	0.909293	0.480859	0.176788	0.522313327
89	25	3.5	91.1924	29.12985	232.5238	-1.023838629	2.192398	4.129854	4.523839	6.505952633	0.886997	0.45064	0.316911	0.55151611
89	25	3.5	90.05274	29.30434	231.7061	-0.206104564	1.052742	4.304343	3.706105	5.77675047	0.907791	0.452798	0.112875	0.491154855
89	50	3.5	90.15652	53.45928	230.4194	1.080584923	1.15652	3.459279	2.419415	4.376952958	0.860201	0.79919	0.250065	0.636485203
89	50	3.5	90.06924	53.23583	229.8479	1.652124341	1.069242	3.235832	1.847876	3.8766654	0.854714	0.749486	0.318985	0.641061633
89	50	3.5	90.26938	53.62258	229.884	1.616023549	1.269379	3.622585	1.883976	4.275957297	0.890489	0.78796	0.324481	0.667643463
89	50	3.5	90.39366	53.37009	230.7768	0.723176055	1.393661	3.370091	2.776824	4.583726926	0.904935	0.7807	0.348213	0.67794919
89	75	3.5	90.03563	76.88143	233.7139	-2.213881421	1.035634	1.881432	5.713881	6.104159836	0.925922	0.902723	0.849509	0.892718313
89	75	3.5	90.2195	77.09091	233.7255	-2.225481401	1.2195	2.090914	5.725481	6.216127362	0.913221	0.885196	0.851481	0.88329931
89	75	3.5	90.14345	77.09575	233.0886	-1.588580817	1.143454	2.095752	5.088581	5.620793372	0.918248	0.895661	0.874128	0.896012387
89	100	3.5	89.99378	103.7892	227.9791	3.520913688	0.993775	3.789161	0.020914	3.91736785	0.901501	0.8074	0.798649	0.835850147
89	100	3.5	91.69238	101.689	229.364	2.135953954	2.692384	1.689013	1.364046	3.458658283	0.920765	0.823919	0.799008	0.84789737
89	100	3.5	90.0243	103.2305	229.5065	1.993522954	1.024298	3.230451	1.506477	3.708702497	0.925135	0.798951	0.798847	0.84097728
89	125	3.5	91.30333	127.1552	229.0466	2.453361438	2.303335	2.155226	1.046639	3.32352249	0.88082	0.832907	0.726713	0.813480333
89	125	3.5	91.41389	127.2044	230.0449	1.455072172	2.413887	2.204426	2.044928	3.855914495	0.915173	0.811899	0.705695	0.81092229
89	125	3.5	152.6015	144.1515	162.3222	69.17775478	63.60146	19.15147	65.67775	93.41034298	0.893357	0.859886	0.743135	0.832126063
89	150	3.5	93.52917	149.5686	228.8625	2.637490532	4.529171	0.43141	0.862509	4.630704997	0.9464	0.791938	0.319976	0.686104723
89	150	3.5	92.88372	149.5243	229.2626	2.237398131	3.883721	0.475718	1.262602	4.111417931	0.918975	0.819102	0.404784	0.71428695
89	150	3.5	93.27885	149.6472	224.9959	6.504124005	4.278853	0.352779	3.004124	5.240018905	0.947223	0.488136	0.012112	0.482490218
89	175	3.5	92.73628	175.4744	228.8166	2.683403693	3.736282	0.474435	0.816596	3.853793324	0.934722	0.863604	0.600616	0.799647257
89	175	3.5	92.39505	175.4655	228.9325	2.567464823	3.395047	0.46548	0.932535	3.551427629	0.93694	0.897005	0.688686	0.840877017
89	175	3.5	91.57184	174.9915	228.0239	3.476128553	2.571838	0.008451	0.023871	2.571962729	0.952487	0.900751	0.60938	0.820872617
					Anomolous Average		4.978776	3.047492	5.270151	8.603916786				0.735713003
					Anomoly Free Average		2.187219	2.280636	2.393598	4.565515539				0.731121904

Appendix 8: X, Y Route 3 test Results table

ROUTE 3

Actual Position			Estimated Position							Reference point Accuracy				
x	y	z	x	y		z	Error in x	Error in y	Error In Z	3d Euclidean Error	1	2	3	Average Reference Point confidence Score
25	40	3.5	48.81358	36.17422	235.2184	-3.718429733	23.81358	3.825776	7.21843	25.17595604	0.872504	0.766712	0.709543	0.782919713
25	40	3.5	51.98269	35.98049	235.7397	-4.239734365	26.98269	4.019515	7.739734	28.35710694	0.881804	0.817498	0.696639	0.7986467
25	40	3.5	54.88427	35.93399	235.4468	-3.946763352	29.88427	4.066013	7.446763	31.0653581	0.918323	0.782579	0.668974	0.7899586
45	40	3.5	44.58827	48.34315	234.9316	-3.431606854	0.411726	8.343154	6.931607	10.85471865	0.93206	0.917146	0.803511	0.884239287
45	40	3.5	44.73319	48.36793	235.538	-4.037971685	0.266812	8.367925	7.537972	11.26562829	0.940296	0.900459	0.795847	0.878867193
45	40	3.5	44.98735	48.08931	235.7357	-4.235730923	0.012646	8.089313	7.735731	11.19279567	0.944288	0.920478	0.729555	0.86477365
65	40	3.5	64.78234	45.26088	233.625	-2.124957734	0.217663	5.26088	5.624958	7.704828578	0.938106	0.93702	0.583104	0.819409917
65	40	3.5	64.27089	45.01774	229.6669	1.833072678	0.729114	5.017741	1.666927	5.337412998	0.936175	0.92826	0.572114	0.812183233
65	40	3.5	65.52232	46.45475	228.9583	2.541681374	0.522324	6.454749	0.958319	6.546371223	0.952482	0.913153	0.664712	0.843449093
85	40	3.5	79.93063	45.57448	229.617	1.882986131	5.069367	5.574483	1.617014	7.706366144	0.934447	0.933212	0.806051	0.891236867
85	40	3.5	80.25361	46.36435	229.5616	1.938383602	4.746389	6.364354	1.561616	8.091468011	0.931887	0.908145	0.850302	0.8967778
85	40	3.5	79.87561	46.36362	229.277	2.223012804	5.124389	6.363616	1.276987	8.269562927	0.938222	0.935501	0.7815	0.885074153
105	40	3.5	100.5798	46.27413	229.9622	1.537848861	4.420158	6.274132	1.962151	7.921651819	0.92943	0.911809	0.89288	0.91137284
105	40	3.5	100.5511	46.39538	231.186	0.313968776	4.448947	6.395379	3.186031	8.416935354	0.930429	0.928466	0.918585	0.9258267
105	40	3.5	100.2372	46.4968	229.8835	1.616501231	4.762843	6.496804	1.883499	8.272889754	0.934381	0.933345	0.901668	0.923131383
105	40	3.5	100.2784	45.87057	230.7313	0.768732218	4.721582	5.870572	2.731268	8.013537292	0.934748	0.920512	0.919901	0.92505388
125	40	3.5	118.8881	46.16664	230.7612	0.738762688	6.111882	6.166643	2.761237	9.110818181	0.94548	0.942252	0.918284	0.935338867
125	40	3.5	118.9758	46.23609	230.1053	1.394653106	6.024189	6.236091	2.105347	8.922564538	0.950178	0.929971	0.923508	0.934552137
125	40	3.5	118.9032	46.25839	229.9312	1.568824848	6.096848	6.258387	1.931175	8.94809473	0.934172	0.920633	0.917575	0.9241264
125	40	3.5	118.8348	46.3925	229.4583	2.041749883	6.165165	6.392501	1.45825	8.999989665	0.944339	0.920828	0.911341	0.925502867
145	40	3.5	138.8709	45.85108	236.1503	-4.65032172	6.12911	5.851078	8.150322	11.75707615	0.923851	0.894322	0.772049	0.86340753
145	40	3.5	138.1517	46.45979	234.0448	-2.54479795	6.848277	6.459789	6.044798	11.18782187	0.944705	0.935109	0.789366	0.889726667
145	40	3.5	137.7058	46.71744	232.2135	-0.713473452	6.977726	6.097677	4.213473	10.77417817	0.940733	0.916821	0.758638	0.872063867
165	40	3.5	160.5978	43.63894	234.3188	-2.818810744	7.085161	6.108492	6.318811	8.517568052	0.884844	0.874155	0.616895	0.7919646
165	40	3.5	160.6411	43.57577	234.0302	-2.530163477	6.217249	6.202579	6.030163	8.255223922	0.877205	0.857293	0.552498	0.762331833
165	40	3.5	160.1364	43.78126	234.9417	-3.441661726	4.86358	3.781263	6.941662	9.28111148	0.883872	0.882003	0.692125	0.8193335
					Anomolous Average		6.871296	6.013035	4.347471	11.15180902				0.867356511
					Anomoly Free Average		4.259702	6.279461	3.940405	8.928200585				0.877380185

Appendix 9: X, Y Route 4 test Results table

ROUTE 4														
Actual Position			Estimated Position							Reference point Accuracy				
x	y	z	x	y		z	Error in x	Error in y	Error In Z	3d Euclidean Error	1	2	3	Average Reference Point confidence Score
25	98.5	3.5	22.67489	109.2679	231.0438	0.456240299	2.325112	10.76786	3.04376	11.42879423	0.942688	0.898056	0.777054	0.872599093
25	98.5	3.5	22.55407	109.868	231.2248	0.275160221	2.445929	11.36804	3.22484	12.06708208	0.924923	0.900109	0.752075	0.859035267
25	98.5	3.5	22.6622	110.2004	230.9307	0.569312062	2.337802	11.7004	2.930688	12.28631976	0.912546	0.910326	0.752086	0.858319433
45	98.5	3.5	45.62169	103.203	230.0604	1.439555474	0.62169	4.703013	2.060445	5.172065251	0.959857	0.931983	0.895151	0.928996997
45	98.5	3.5	45.69438	103.1562	230.1874	1.312643833	0.694377	4.656228	2.187356	5.191063846	0.962408	0.927607	0.895324	0.928446377
45	98.5	3.5	45.45277	102.94	230.1145	1.385539767	0.452773	4.43997	2.11446	4.93855046	0.95942	0.932734	0.904199	0.93211745
45	98.5	3.5	45.49179	103.031	230.5656	0.934428426	0.49179	4.531006	2.565572	5.230107746	0.960345	0.917829	0.909295	0.929156363
65	98.5	3.5	64.64163	102.4001	229.9566	1.543355072	0.358368	3.900124	1.956645	4.378110525	0.955062	0.943376	0.888294	0.9289105
65	98.5	3.5	64.77631	102.4726	229.624	1.875973329	0.223694	3.972582	1.624027	4.297546444	0.957795	0.92856	0.890869	0.925741593
65	98.5	3.5	64.86336	102.1868	229.5282	1.971845722	0.136645	3.686814	1.528154	3.993309849	0.951293	0.924951	0.896847	0.9243638
85	98.5	3.5	81.45613	102.3305	230.6196	0.880406545	3.54387	3.830455	2.619593	5.838978137	0.952092	0.92023	0.920017	0.930779633
85	98.5	3.5	80.47601	102.0818	232.3555	-0.855478217	4.523988	3.581766	4.355478	7.229502488	0.952583	0.921036	0.919939	0.931185977
85	98.5	3.5	81.49361	102.2709	231.9244	-0.424393562	3.506391	3.770861	3.924394	6.474182074	0.951463	0.924417	0.912903	0.929594363
105	98.5	3.5	100.4322	102.2377	231.1196	0.38040715	4.567811	3.737732	3.119593	6.675882134	0.912199	0.888282	0.880476	0.893652333
105	98.5	3.5	101.5072	101.2194	230.9414	0.558599698	3.492801	2.719395	2.9414	5.314753331	0.925589	0.885525	0.860267	0.89046059
105	98.5	3.5	100.3024	102.34	230.7915	0.708487144	4.697632	3.840024	2.791513	6.678777954	0.929292	0.902255	0.875989	0.902512033
125	98.5	3.5	121.9543	104.1677	232.742	-1.241983979	3.045705	5.66768	4.741984	7.992829861	0.917599	0.872052	0.871789	0.887146767
125	98.5	3.5	122.0884	103.8007	232.9175	-1.417467973	2.911568	5.300724	4.917468	7.794638652	0.935428	0.875744	0.874354	0.895175223
125	98.5	3.5	122.1677	104.1008	233.2856	-1.785582781	2.832293	5.600766	5.285583	8.205354961	0.935821	0.901956	0.888562	0.908779567
145	98.5	3.5	138.9793	105.8713	233.9276	-2.427642215	6.020744	7.371254	5.927642	11.21256811	0.9071	0.898835	0.884514	0.896816067
145	98.5	3.5	138.7433	105.8516	233.6034	-2.103399576	6.256725	7.351601	5.6034	11.16202157	0.911253	0.908549	0.906076	0.908625933
145	98.5	3.5	139.0907	105.9691	234.3904	-2.890363963	5.909289	7.469123	6.390364	11.46927374	0.915343	0.901649	0.877338	0.898110233
165	98.5	3.5	158.3882	104.4743	231.1312	0.368806381	2.790773	5.634882	3.131194	9.445237127	0.920119	0.917544	0.854082	0.897248043
165	98.5	3.5	157.9637	104.2432	231.142	0.358016003	2.69997	5.720352	3.141984	9.61069495	0.919606	0.914074	0.869266	0.90098183
165	98.5	3.5	157.9757	104.3186	231.0767	0.423270757	2.800807	5.388893	3.076729	9.626221377	0.925755	0.911776	0.819685	0.885738637
					Anomolous Average		2.787542	5.628462	3.408171	7.748554666				0.905779764
					Anomoly Free Average		2.787542	5.628462	3.408171	7.748554666				0.905779764

Appendix 10: X, Y Route 5 test Results table

ROUTE 5

Actual Position			Estimated Position							Reference point Accuracy				
x	y	z	x	y		z	Error in x	Error in y	Error In Z	3d Euclidean Error	1	2	3	Average Reference Point confidence Score
40	40	3.5	44.43449	38.68222	235.3855	-3.885529813	4.434487	1.317781	7.38553	8.714772946	0.933331	0.896893	0.725664	0.85196278
40	40	3.5	44.52938	38.56658	235.0114	-3.511430776	4.529379	1.433421	7.011431	8.469364438	0.931577	0.882993	0.582037	0.79886925
40	40	3.5	44.22876	38.96611	233.9831	-2.483131297	4.228758	1.03389	5.983131	7.399269146	0.937137	0.895293	0.774171	0.868867003
52	56	3.5	45.86913	58.90805	233.1947	-1.694651933	6.130871	2.908046	5.194652	8.545684545	0.930295	0.921528	0.906354	0.91939225
52	56	3.5	46.3442	59.0152	233.2217	-1.721715455	5.655803	3.015199	5.221715	8.267154498	0.934911	0.9022	0.887937	0.9083495
52	56	3.5	46.04633	58.89588	233.0356	-1.535620831	5.953671	2.895884	5.035621	8.318041609	0.917161	0.909583	0.90902	0.911921467
64	72	3.5	62.23805	75.78917	232.6054	-1.105376739	1.761945	3.789166	4.605377	6.218659391	0.941291	0.885387	0.875766	0.9008145
64	72	3.5	62.24192	76.27612	232.5137	-1.013675079	1.758077	4.276116	4.513675	6.461367211	0.92555	0.905585	0.882551	0.90456227
64	72	3.5	62.36387	76.1404	232.2059	-0.705931506	1.636127	4.140402	4.205932	6.124515907	0.922484	0.906346	0.905857	0.911562
76	88	3.5	73.58054	89.0981	231.1891	0.31094192	2.419459	1.098102	3.189058	4.150867682	0.959279	0.902989	0.867004	0.909757347
76	88	3.5	73.43528	89.18964	231.0405	0.459481721	2.564717	1.189637	3.040518	4.151838228	0.957549	0.919058	0.90515	0.927252143
76	88	3.5	73.54998	89.0303	231.2844	0.215559125	2.450018	1.030297	3.284441	4.225121481	0.963108	0.876682	0.875746	0.905178903
88	104	3.5	82.43245	104.7333	230.1803	1.31967713	5.56755	0.733306	2.180323	6.024048521	0.936421	0.912146	0.84085	0.896472417
88	104	3.5	81.88667	104.4645	229.229	2.270974598	6.113326	0.464466	1.229025	6.252917807	0.932169	0.873079	0.850809	0.8853523
88	104	3.5	82.81867	104.3975	229.1221	2.377933802	5.181333	0.397478	1.122066	5.316317908	0.933539	0.876019	0.866996	0.89218457
100	120	3.5	91.93839	123.799	233.3885	-1.888547839	8.061608	3.798992	5.388548	10.41433237	0.95411	0.8732	0.862398	0.896569157
100	120	3.5	91.91777	124.6413	235.1067	-3.606718623	8.082226	4.641344	7.106719	11.72049071	0.946564	0.861706	0.824308	0.877526
100	120	3.5	92.39591	122.7437	229.2862	2.213849586	7.604088	2.743655	1.28615	8.185595967	0.960403	0.876641	0.824246	0.887096637
112	136	3.5	106.9317	142.8082	233.3328	-1.832766996	5.068252	6.808154	5.332767	10.02379931	0.928699	0.887028	0.847477	0.887734417
112	136	3.5	105.6437	140.4468	232.2377	-0.737685868	6.356322	4.446753	4.237686	8.839367513	0.938298	0.856214	0.832984	0.875832107
112	136	3.5	107.0053	143.0096	233.2991	-1.799124819	4.994671	7.009626	5.299125	10.10753812	0.925821	0.893984	0.868795	0.89619991
124	152	3.5	115.8717	158.8917	227.3537	4.146335951	8.128321	6.891744	0.646336	10.67630529	0.904742	0.798444	0.638669	0.7806183
124	152	3.5	105.9968	98.46278	235.0701	-3.570115654	18.00319	53.53722	7.070116	56.92394433	0.913483	0.823864	0.652146	0.79649729
124	152	3.5	115.6957	159.0703	227.7984	3.701594519	8.304261	7.070274	0.201595	10.90826085	0.914331	0.811714	0.778046	0.834697243
136	168	3.5	115.7723	159.0488	227.4692	4.030835175	20.22773	8.951204	0.530835	22.12615935	0.932464	0.797686	0.661716	0.79728848
136	168	3.5	132.088	167.8849	229.3776	2.122363901	3.911976	0.115053	1.377636	4.14905707	0.916823	0.862833	0.835916	0.87185726
136	168	3.5	131.7918	167.935	228.4776	3.022352775	4.208216	0.065013	0.477647	4.235735564	0.901054	0.869931	0.823827	0.864937503
136	168	3.5	132.4079	167.8982	229.6716	1.82840424	3.592119	0.101844	1.671596	3.96332224	0.905037	0.84785	0.83048	0.861122117
148	184	3.5	142.7234	182.4981	246.0158	-14.51578836	5.276618	1.501863	18.01579	18.83260269	0.914966	0.868284	0.63737	0.806873067
148	184	3.5	142.9625	182.3233	246.161	-14.66100622	5.03755	1.676745	18.16101	18.92116612	0.919234	0.861843	0.791843	0.857640233
148	184	3.5	142.8471	182.4537	246.1359	-14.63588953	5.152868	1.546257	18.13589	18.91701475	0.933838	0.862857	0.545181	0.780625233
160	200	3.5	152.2759	203.6785	253.6379	-22.13785242	7.72409	3.678496	25.63785	27.02762262	0.892048	0.551518	0.344534	0.596033077
160	200	3.5	152.3584	206.8296	253.0914	-21.59142494	7.64161	6.829574	25.09142	27.10381689	0.854964	0.479293	0.373587	0.56928102
160	200	3.5	152.9313	203.6074	252.358	-20.85801535	7.068657	3.607387	24.35802	25.61819799	0.855886	0.397297	0.355371	0.536184827
					Anomolous Average		6.024409	4.551306	6.859683	11.98041974				0.84315037
					Anomoly Free Average		4.971194	2.6945	5.191068	8.570156203				0.874890617