



TO DEVELOP TECHNIQUES TO MAKE A REAL-TIME 3D MAPPING SYSTEM USING A UAV

Oscar Simons



Contents

Contents	2
1. Summary/Abstract	4
2. Acknowledgements.....	4
3. Introduction	4
4. Design Brief	4
4.1 Getting the quadcopter to fly using a computer	4
4.1.1 Introduction to the problem	4
4.1.2 Putting the problem into context.....	5
4.2 Getting the drone to detect an obstacle when moving to a desired location	5
4.2.1 Introduction to the problem	5
4.2.2 Putting the problem into context.....	5
4.3 Localising in a pre-built scene including orientation of the drone	5
4.3.1 Introduction to the problem	5
4.3.2 Putting the problem into context.....	5
4.4 Creating a 3D scene with an accurate scale to fly in without crashing	6
4.4.1 Introduction to the problem	6
4.4.2 Putting the problem into context.....	6
4.5 Using a 3D scene to allow the drone to update itself as it flies	6
4.5.1 Introduction to the problem	6
4.5.2 Putting the problem into context.....	6
4.6 Getting the drone to detect something has moved in order to not get a blurred or mismatched image	7
4.6.1 Introduction to the problem	7
4.6.2 Putting the problem into context.....	7
4.7 Using OpenCV tools to get as much out of the camera as possible	7
4.7.1 Introduction to the problem	7
4.7.2 Putting the problem into context.....	7
5. Method for the solution.....	8
5.1 Getting the quadcopter to fly using a computer	8
5.2 Getting the drone to detect an obstacle in its way when moving to a desired location	8
5.3 Localising itself in a pre-built scene including orientation of the quadcopter	9
5.4 Creating a 3D scene with an accurate scale to be able to fly the quadcopter in without crashing	9
5.5 Using a 3D scene to allow the quadcopter to update itself as it flies	10

5.6 Getting the drone to detect something has moved in order to not get a blurred or mismatched image	11
5.7 Using OpenCV tools to get as much out of the camera as possible	11
6 Results	11
6.1 Getting the quadcopter to fly using a computer	11
6.2 Getting the drone to detect an obstacle in its way after the computer wants to move to a desired location	12
6.3 Localising itself in a pre-built scene including orientation of the quadcopter	12
6.4 Creating a 3D scene with an accurate scale to be able to fly the quadcopter without crashing	12
6.5 Using a 3D scene to allow the quadcopter to update itself as it flies	12
6.6 Getting the drone to detect something has moved in order to not get a blurred or mismatch image	12
6.7 Using OpenCV tools to get as much out of the camera as possible	13
7. Summary of Results	13
8 Testing Methods	13
8.1 Getting the quadcopter to fly using a computer	13
8.2 Getting the drone to be able to detect an obstacle in its way after the quadcopter wants to move to a desired location	13
8.3 Localising itself in a pre-built scene including orientation of the drone with respect to the scene.....	14
8.4 Creating a 3D scene with an accurate scale to be able to fly the drone without crashing	14
8.5 Updating a 3D scene to allow the drone to update itself as it flies.....	14
8.6 Getting the drone to detect something has moved in order to not get a blurred or mismatched image in the scene.....	14
8.7 Using OpenCV tools to get as much information from the scene as possible	15
9 Project Management	15
9.1 Project Cost (all figures in pounds).....	15
9.2 Time Management	16
10 Conclusions	16
11 Further work	18
12 References	19
13 Appendices.....	22
13.1 Table of abbreviations and key terms	22
13.2 MBED code:	22
13.3 Diagrams and photos.....	23

1. Summary/Abstract

This report describes a method to produce a simultaneous localization and mapping (SLAM) map using distance measurements and angular rotations of time of flight (TOF) sensors. Although the budgets for many light detection and ranging (LIDAR) systems are over £50,000 [1] the system being described in this report is low cost and efficient. This report will show how to produce a quadcopter-based mapping system. It will highlight challenges and problems the system has overcome by completing the various objectives of the project. Through recording the measurements of time of flight sensors and accelerometer readings this system is able to produce a map of an environment which is accurate in scale.

2. Acknowledgements

I'd like to thank Plymouth University staff, especially the Lab Technicians, for their help.

3. Introduction

The problem of building a map and also localising yourself in a map has been around for many years. The problem can be broken down into two separate problems - localization and mapping although to some extent the problems are intertwined. This is because the most common solutions use a combination of distance measurements and cameras [2]. Distance measurements can be by far the best way to produce detailed maps, however the map is limited in the sense of colour. Cameras are a useful tool and with the rise of high speed processing and better tools such as OpenCV, the usefulness of making a 3D scene to scale with this setup is getting better by the year. Much research has gone into the functionality of using cameras for SLAM [2] [3] [4] [5] [6]. Not only does the use of cameras bring to life the raw data of the maps created, but they also allow for a more interactive experience with people to really see what the process is all about. The approach shown in this article is inspired by these links [8] [9] using the time of flight sensors which are used for mapping on the quadcopter (also referred to as a drone).

4. Design Brief

4.1 Getting the quadcopter to fly using a computer

4.1.1 Introduction to the problem

In order to solve the problem of quadcopter-based SLAM it was broken down into a series of objectives or steps. They include the following:

- I. Getting the quadcopter to fly using a computer
- II. Getting the quadcopter to detect an obstacle in its way when moving to a desired location
- III. Localising itself including orientation of the quadcopter with respect to the scene
- IV. Create a 3D scene with accurate scale to be able to fly the quadcopter without crashing
- V. Using the 3D scene to allow the quadcopter to update the scene as it flies
- VI. Getting the drone to detect something blurred or mismatched image in the scene

VII. Using OpenCV tools to get as much information out of the cameras as possible.

This section aims to highlight the proposed solutions to these problems previously stated and how they were solved.

4.1.2 Putting the problem into context

For a mapping system, being able to control the quadcopter effectively with a computer is essential to automate the process [10] [11] [12] as well as to maintain stable flight. There are four key controls needed in this solution - the drone – a Skyma X8c [13]. As you can see in Figure 2 the controller is broken down into two joysticks - throttle and steering. In this approach the controller is dis-assembled to enable a better automated approach to the flying using a computer and an MBED board.

4.2 Getting the drone to detect an obstacle when moving to a desired location

4.2.1 Introduction to the problem

Detecting obstacles is an important step not only in flying capability, but will be used further looking into mapping. This process is very important when looking at autonomous flying using a computer. The process of detecting obstacles in a scene is crucial when looking for landmarks in a scene [7] [17] [18] [19]. This section will explain the process of detecting an obstacle near the quadcopter as well as alerting the user in a timely manner via the SSH connection to a raspberry pi.

4.2.2 Putting the problem into context

This is crucial not only for detecting paths through an environment, but also the ability to focus on key areas. In SLAM you need to focus on individual objects in an environment so they are in essence obstacles for the robot to move around or locate in a scene in order to move on further to mapping.

4.3 Localising in a pre-built scene including orientation of the drone

4.3.1 Introduction to the problem

The problem of localising yourself in an environment has been around for centuries since the first map was created. The most widely known and used version of localisation has been around since the 1970's and is the Global Positioning System (GPS) [20]. Now GPS might be the most common, but this is fraught with problems and very high setup costs. Localisation at its heart is simply knowing where you are.

4.3.2 Putting the problem into context

Localisation and SLAM are very much intertwined in that you cannot create a map without a point of reference. Localising yourself can be easy, but it can also be extremely hard to maintain when moving. GPS is great with its ability to locate itself all over the world, but is not very good for SLAM [21]. For instance the accuracy range is generally +/- 5 meters, but this is insufficient for SLAM since many rooms are a lot less than 5 meters in length or width. In the next few sections this paper will look at how the quadcopter can localise itself within a scene without GPS and get a truly accurate position in a 3D scene that the quadcopter will create.

4.4 Creating a 3D scene with an accurate scale to fly in without crashing

4.4.1 Introduction to the problem

This is by far the most obvious and impressive aspect of the project. Creating a map can be very simple or very complex with strong ties to both localising and updating a scene. This is however also the aspect with potentially the broadest of interpretations. 3D scene generation has increased enormously with the development of computer graphics [22] [23] [24] [25]. Since the 1970's techniques have become widely used in the games industry. However the generating of accurate 3D scenes has also been going on for SLAM users as well. While not so mainstream the raw problems remain the same. Namely, how can someone create a scene that anyone can recognise?

4.4.2 Putting the problem into context

This development of creating a scene is critically important to the overall performance of the project. Everyone knows when they see a good 3D scene although not many people know how they are created from video games to Google street views. This is the most visible aspect of the project and also the most impressive. In the following sections this paper will go into further detail of how the system combines mechanical and electrical aspects to produce the map.

4.5 Using a 3D scene to allow the drone to update itself as it flies

4.5.1 Introduction to the problem

Updating a scene is important and very useful when using SLAM mapping to build up the large amount of data required to generate 3D scenes. Updating scenes is also where things go wrong whether from moving obstacles to differences in position. The minute you have two readings of the same point there will be the potential for data mismatches. This is because when you update a scene all the calibrations and setting of any sensors you have are tested to their limit because of differences in orientation, rounding errors, or time delay in taking multiple readings from sensors.

4.5.2 Putting the problem into context

The proposed solution plans to merge scenes together. This will mean that rather than a series of errors causing a distortion, this approach will allow the mapping software to look at the data as a whole and not as individual data points. This will further allow the data to form a 3D scene allowing erroneous data to be insignificant as more and more data points are collected.

4.6 Getting the drone to detect something has moved in order to not get a blurred or mismatched image

4.6.1 Introduction to the problem

Blurring in slam is a huge problem for movement of objects in a 3D scene which is a permanent feature [26] [27]. What this objective is addressing is the loop closure problem which is a major obstacle to overcome. If a robot were to be programmed to go in a square by going along for 30cm and round 90 degrees and keep on doing that till it returns to where it started from, it would not work. Most robots will have PWM controllers if they're floor robots. Even looking at the simplified structure of the floor robot every time the robot turns too much or receives an extra pulse the robot will be turning to a new position. Loop closure is when the robot returns to the same position or thinks it does and it will believe it's in the same position as before and update the scene around it [28] [29] [30] [31]. The article describes how many errors are caused by this problem and while this is a simple one to see it can be impossible to fix or identify from a computing standpoint.

4.6.2 Putting the problem into context

Data anomalies are inevitable through noise and distortion to the environment or the time delay in taking samples. In any system there will be data anomalies that cause downstream problems when being processed by algorithms. This objective aims to cure a common problem in SLAM systems and that is loop closure. Quadcopters are notorious for drifting and with incalculable degrees of freedom available to the quadcopter, the possibility of this problem arising is large. There is no one fix for the problem and conflicting data is not just an issue for SLAM systems but everywhere such quadcopters are used.

4.7 Using OpenCV tools to get as much out of the camera as possible

4.7.1 Introduction to the problem

Cameras are a widely used sensor and have been around since the 1500's [37] since when they have taken several forms and shapes. Through years of refinement, camera development continued through to 1991 when Kodak released the first disposable camera and on to the digital cameras of today. Cameras are used in almost every robotic system. With the flexibility of a camera installed on a robot, it has the ability to detect changes in the environment [32] [33] [34] [35] [36]. With cameras getting clearer images and higher pixels it is important to be able to retrieve data from them. As cameras can capture more and more data the ability to process that data is a rapidly growing industry and is the solution to using the sensor data from cameras. While not everyone has the same opinion of how successful this will be the industry will be worth over 20 billion US dollars by 2023 [38] [39] [40] [41] [42].

4.7.2 Putting the problem into context

The problem of using cameras in 3D scene generation is mostly about how to pick out the data from the camera feed. While there are many algorithms to use mono-slam and stereo vision, the plan for the proposed system is to use OpenCV functions such as contours [43]. This will allow the system to pick out key points which will be visible from the colourless 3D map in order to match the sensors data streams together into a single map.

5. Method for the solution

5.1 Getting the quadcopter to fly using a computer

The Skyma main controller works using two joysticks which both have 2 degrees of freedom. The original controller uses two potentiometers attached to each joystick which are fed into the microcontroller on the board. My approach takes a black box approach to the control mechanisms by connecting the inputs to the controller and attaching them to the MBED board STM32F429 (see figure 3 for the setup and figure 4 for the circuit). As you can see from figure 4 the system used PA_4, PA_5 using the two on-board Digital to Analogue Converters (DAC) and two pulse width modulation (PWM) outs located at D5,D6. The DAC's work in a similar way to the original controller and can provide voltage ranging from 0 to 3.3 output. This enables the controller to use the full range of voltage to control the quadcopter with a high degree of accuracy down to 0.1v.

However for steering, due to the fact that the MBED board and other microcontrollers rarely have 4 DAC's. I chose to use the two provided as well as use PWM outs [14]. Using this approach and keeping track of the controller I was able to output the control to the user using keyboard strokes (see figure 5). This enables the user to control the quadcopter effectively. The way the steering works is that instead of using the amount of voltage to directly input to the microcontroller, it turns the signal rapidly on and off as a percentage. Because the nucleo board runs at a far greater speed than the quadcopter controller the average voltage is evened out and it can steer the quadcopter effectively.

The controller for the user is very simple (see figure 5). Using these keys and treating them like arrow keys it enables the user to use the keyboard like the original controller. However an added bonus of using the computer is the stop function which can be invoked by pressing the space key. This brings the steering and forwards momentum to a 50% duty cycle and the throttle to a power neither left nor right as well as stopping the power to the controls bringing the quadcopter safely down. To help the User know that the messages from their keyboard input is getting through, the controller uses putty. This allows the User to see messages from the MBED board as well as sending inputs to it as the User (See figure 6).

5.2 Getting the drone to detect an obstacle in its way when moving to a desired location

The method of detecting obstacles in the quadcopters flight path is simple and uses the same TOF sensors you will see in the upcoming sections. It makes sense however to look at the sensor itself at this point. Time of flight sensors are something of a misnomer because in fact they use phase shifts between the outgoing and incoming beams of constant light. The deflection of the light will then be used to calculate the distance [44][46]. In this project the sensors being used are the VL53L0x [45]. With the i2c multiplexer TCA9548A [47] allowing for the ability of 8 TOF sensors the system is not at full capacity. This allows for the possibility of further development in the future. The TOF sensor has a range measurement of 2 meters. This makes it perfect for mapping most rooms as the sensors will be mounted in a cross shape enabling them to fly freely, but giving enough time to react as a wall appears. Using the library from GitHub [48] I was able to successfully extract the distance from all of the 7 TOF sensors (see figure 7). Using this you can then detect if any obstacle is near the sensor indicating that it may crash. This then will alert the user as shown in (figure 7).

5.3 Localising itself in a pre-built scene including orientation of the quadcopter

The algorithm for localisation is a straight forward process. Measurements are taken from 7 TOF readings and then compared to a list of pre-recorded data points. This involves going through the data points one by one until a possible match is found. However to find the exact position all of the sensors need to match. Diagram 2 shows the matching algorithms flowchart which is used to match the current data to the data set of the same map.

5.4 Creating a 3D scene with an accurate scale to be able to fly the quadcopter in without crashing

The generation of a 3D scene is one of the major aspects of the project. There are several parts to this section including collecting the data, saving the data to be used later, processing the data, and finally displaying the data in a 3D setting.

5.4.1. Collecting the data

Data collection for the 3D mapping system starts with setting up the sensors. The Python library uses an object orientated approach which allows easy setup for each individual sensor (see figure 8). Each TOF sensor has its own number referring to the number on the i2c buss when the object is declared in Python (see figure 12). This then allows the program to set the accuracy rating as shown in figure 13. After this the program requests a sample on the i2c buss whenever it wants. As well as collecting TOF data the program also collects readings from an accelerometer (LIS3DH) [54] through the same i2c buss.

One of the main concerns in collecting all of this data at once is that there could be a problem with noise and distortion on the i2c line. Figures 14, 15, 16 and 17 show how to determine if the sampling of data will be too noisy. Although the average pulse width for the signal has decreased when the extra sensor is added, there is a possibility of data corruption, but the difference in pulse width of $15.58\mu\text{s}$ has a fair way to go before it becomes too distorted. The main concern would be if a signal gets distorted or compressed in transit, the minimum pulse width would decrease and result in a triangular wave form. From looking at the wave forms in figures 14 and 16, as well as the data in 15 and 17, I can determine that although the accelerometer has not decreased it to the point that data corruption is common in the system, this is not something to worry about.

5.4.2. Saving the data

The data points from the 7 TOF readings [18] as well as recordings of the position set by the program and the accelerometer data (see figure [19]) are then saved directly into a worksheet which is saved to a file [20] at the end of the data collection phase. Due to the differences in how the data needs to be saved and opened, a conversion program was used to go from xls file type to csv file type [21].

5.4.3. Processing the data

The method used to process data in this project is using forward kinematics, the Denavit-Hartenberg transformations [49] [50] [51] [52]. Using this method you can find X,Y,Z, positions which can be applied to a quadcopter mapping system. The first thing to do is to define the links of the quadcopter. The links are shown in figure 10 and are as follows:

1. From the origin of the map to the base of servo one
2. From servo one to servo 2
3. From servo 2 to the point the TOF sensor is reading.

Using this model it successfully came up with the conversion formula for X,Y,Z, coordinates. Then using HS-422 [53] servo motors, figure 10 shows the model for the robot joints with the last joint becoming where the TOF sensor is placed. Figure 11 then shows the 3D printed frame for the actual implementation of the links (except for the first link which would be from the origin to the servo motor which can change as the quadcopter flies). The full equations used when the quadcopter is fully mobile are:

$$x = \cos(C) \cos(e) - \sin(C) \cos(a) \sin(e) G \cos(H) + \sin(c) \sin(A) G \sin(H) + 3.5 \cos(c) - 3.5 \sin(e) + 3.5 \sin(c) \sin(a) + b \cos(C)$$

$$y = \sin(c) \cos(e) + \cos(c) \cos(a) \sin(e) - \cos(c) \sin(a) + G \cos(h) - \cos(c) \sin(a) G \sin(h) + 3.5 \sin(c) \cos(e) + \cos(c) \cos(a) 3.5 \sin(e) - \cos(c) \sin(a) 3.5 b \sin(c)$$

$$Z = \sin(a) \sin(e) G \cos(h) + \cos(a) G \sin(H) + \sin(a) 3.5 \sin(e) + 3.5 \cos(a)$$

Where the letters a,b,c,d,e,f,g,h,m representation is shown in figure 22.

Using this method provides the ability to map any transformations possible although it does not allow for efficient real time processing because of the complex equations involved. The letters a,b,c,d all represent the first link of an imaginary line between the origin and the quadcopters base. To increase the processing speed and produce maps in a more real-time way, a stationary map generation formula was made for converting to x,y,z coordinates as follows:

$$x = g \times \sin(h) + 3.5 \times \sin(e) + 3.5$$

$$y = g \times \cos(h) + 3.5 \times \cos(e)$$

$$Z = g \times \cos(h) \times \sin(e) + 3.5 \times \sin(e)$$

As you can see these are much simpler equations to process in real time. It also allows the raspberry pi to act in a more real time way, maximising its ability to process the data. For the implementation of the formulas you can further simplify for the individual sensor due to the fact the sensors all have their own individual relation to the origin of the map.

5.4.4. Displaying the data

To display the maps the program uses an open source program call matplotlib [55]. Python was used to make a range of graphs and present the data. As you can see in the program, the data was plotted as a 3 dimensional coordinate and enabled the equations previously calculated to plot the x,y,z coordinates effectively.

5.5 Using a 3D scene to allow the quadcopter to update itself as it flies

To update a scene the program combines two files together. Previously this report has shown how programs have written and opened data files. The process of combining these maps uses the same features as shown earlier in the sense that it will open the .csv files and output them as .xls files. Figure 28 shows this process by opening both files and reading them into a common array after which the program is able to save them into a new file in the

same format as they were saved previously. This allows them to be converted to csv. This can then be displayed exactly as shown in section 5.4.4.

5.6 Getting the drone to detect something has moved in order to not get a blurred or mismatched image

When considering how to un-blur images we first need to look at how to gather images from a raspberry pi camera. In this system the camera being used is 8mp and capable of taking images that are 3280 by 2464 pixels - allowing large amounts of data to be provided [56]. In this and the next section I have chosen to use OpenCV tools which is one of the largest free computer vision libraries available. There are many tools in OpenCV which I could use to combine two blurred images. One of the things to be aware of is that in order to use the images from a raspberry pi camera there should be a way to process it. In order to take the photos I use a ssh terminal and use the command “raspistill -o cam4.jpg” to take a static image. Looking at matching images together, it is necessary to look further into stitching images together [58] [59]. This helps to combine the images to an overall scene before combining the data streams of TOF sensors and camera data.

5.7 Using OpenCV tools to get as much out of the camera as possible

Previously in the introduction section it was stated that OpenCV is a massive computer vision library to allow the system to look into matching the TOF data to the processed camera data. One of the ways to do this is to think about what the biggest differences in the TOF reading would be. For example if there was a desk then the biggest change would be on the edge of the desk. Using this method you could find the edge of the desk by applying contours to an image. The code for performing edge/contour detection is in figure 29. Using this method you are able to find and show the contours in every image it takes.

6 Results

6.1 Getting the quadcopter to fly using a computer

As you can see from the video I was able to successfully fly the drone using the ‘w’, ‘a’, ‘s’, ‘d’, ‘l’, ‘j’, ‘k’, ‘i’, keys to control the quadcopter. From this screenshot, you can see that I have also created a text based interface for the user to interact with the controller via a PC (see diagram 1). The code follows what was described previously, but you can also find it in the appendix under MBED code for the exact implementation. Below is a link showing the quadcopter being controlled by a computer <https://www.youtube.com/watch?v=f46WQp3lfPE> As shown in the video and the code snippet the Nucleo-F429ZI MBED board was able to successfully drive the quadcopter with a computer, proving the concept.

6.2 Getting the drone to detect an obstacle in its way after the computer wants to move to a desired location

Figure 7 shows the successful implementation of detecting a close obstacle with the TOF sensor. This enables the drone to warn the user of an obstacle in the way of a quadcopter before it's too late. Due to the common problems with drift in the quadcopter the fact that the system is able to look at all the data streams simultaneously (as shown in figure 7) is extremely useful to monitor all TOF sensor data in order to alert the user in a timely manner. This means that the system can alert the user before crashing into obstacles before or during the SLAM mapping process.

6.3 Localising itself in a pre-built scene including orientation of the quadcopter

Figure 8 shows that there can be a lot of individual matches for the algorithm. However this looks for the one with the exact match as shown in figure 9. This demonstrates the ability to locate a point in the scene and then find it again.

6.4 Creating a 3D scene with an accurate scale to be able to fly the quadcopter without crashing

The implementations of the key elements of this project are combined at this point to make a map as shown in figures [24] [25] [26] [27]. The maps are both accurate and to scale. Below is an image taken previously by another SLAM system to allow comparison. Figures [31] [32] [33] are screenshots from RTABmap [59] which is another 3D SLAM system. While it may initially be perceived to be better, closer inspection of the lower corners of [32] [33], it becomes apparent that the map is quite distorted. This is a problem that the map produced as part of this project does not have because of the nature of gathering the information using TOF data. It is more exact and by adding more and more data it does not distort the previously collected data. Also matching the data collected with real life measurements it appears to be the right size and shape for the 3D scene to be accurate.

6.5 Using a 3D scene to allow the quadcopter to update itself as it flies

This section will show the two maps. The first is the more basic map while the second shows the same map but with more data points. This is down to the previous steps of combining the maps to form one image allowing for more data points. This can be shown in a number of different ways ranging from the size of the csv file itself in figure 36 or as the number of columns, as in figure 37 for the original scene, and figure 38 for the updated one, with nearly double the number of entries.

6.6 Getting the drone to detect something has moved in order to not get a blurred or mismatch image

Using OpenCV could help in removing blurry images alongside other features. In the raspberry pi however in order to capture the data you use the command line interface typing "raspstill -o cam.7" shown in figure 34. The program is able to capture an image which can then be used in the next section as well.

6.7 Using OpenCV tools to get as much out of the camera as possible

Figure 30 shows contour edge detection on an image taken from a raspberry pi camera and processed on the raspberry pi using OpenCV.

7. Summary of Results

In this section you have seen how I was able to fly a quadcopter, detect an obstacle, create and update a scene, as well as integrating cameras with my design before extracting as much information as possible from them.

8 Testing Methods

In this section I will go through each of my objectives and explain in more detail how I tested the accuracy of my solutions previously provided.

8.1 Getting the quadcopter to fly using a computer

In order to test whether the quadcopter can fly the method is simple:

- 1) Connect up the circuit including the MBED board and connect the MBED board to a pc.
- 2) Turn on the drone and place it where it can fly safely
- 3) Check that the MBED board is connecting to the pc this should happen because you should see "enter an input" on the putty terminal
- 4) Turn on the controller using the switch as part of the controller previously
- 5) On the computer keyboard in the putty terminal press 'w' until the lights start to flash more slowly than when it was originally on the quadcopter and the controller.
- 6) Press 's' until the lights on the controller and quadcopter are solidly on
- 7) Next using the w key slowly move the quadcopter upwards controlling the direction of travel of the quadcopter with 'l','j','k','l'
- 8) The quadcopter has successfully been controlled via a pc.

8.2 Getting the drone to be able to detect an obstacle in its way after the quadcopter wants to move to a desired location

This objective can be tested with a bench test. This is referring to the fact that the testing method is:

- 1) Putting the PCB (printed circuit board) down on a bench with all the sensors connected and pointing upwards
- 2) One by one an object is placed in front of a sensor.
- 3) If it is working correctly a message should appear warning that the drone may crash.

8.3 Localising itself in a pre-built scene including orientation of the drone with respect to the scene

The testing of this relies on several steps:

- 1) collecting data of a surrounding to act as a database full of time of flights data and samples
- 2) collecting a sample to test from each of the time of flight sensors
- 3) The matching algorithm will go through and pick out any data which is the same as the sample. The way to make sure that the algorithm works is to do the following:
 - a. take the sample measurement and print it to the screen
 - b. copy the measurements of the 7 time of flight reading into the xls file
 - c. Run the algorithm again and see if it says that all the TOF sensors match.

8.4 Creating a 3D scene with an accurate scale to be able to fly the drone without crashing

To test the accuracy of the 3D scene there is a number of steps:

- 1) Run the algorithm to create the 3D scene which is the VL53L0X_TCA9548A_example.py
- 2) Run the converter.py program to allow the open data program to work
- 3) Run open_data.py and view the map created
- 4) Using a tape measure pick out key points in the map
- 5) Compare these results to the map created by the program and see whether they are correct

8.5 Updating a 3D scene to allow the drone to update itself as it flies

The procedure to test this is straight forward:

- 1) Take a map of the room and save it to file one
- 2) Take a second map of the room and save it to file two
- 3) Take note of the two file sizes with respect to the number of rows and the actual file size
- 4) Combine the two maps together
- 5) Compare the new file, the number of items and the file size should have doubled.

8.6 Getting the drone to detect something has moved in order to not get a blurred or mismatched image in the scene

The testing procedure for this is to:

- 1) Run the mapping program
- 2) Move an object in the map
- 3) Combine the two maps together and see if there is a data anomaly.

8.7 Using OpenCV tools to get as much information from the scene as possible

The testing procedure for this is:

- 1) Take an image from the raspberry pi camera
- 2) Import it to a Python script with OpenCV library running
- 3) Run the Python scripts and see if the resulting image shows the key contours of the image.

9 Project Management

9.1 Project Cost (all figures in pounds)

Item	Cost of one	Number of items used	Total cost of all item (£)	Running total (£)
X8C quadcopter	80	1	80	80
Raspberry pi	30	1	30	110
VL53Lox	6.33	7	44.31	151.31
I2c multiplexer	5.32	1	5.32	159.63
HS-422 servo	12.50	2	25	184.63
PCB`	5	2	10	194.63
Battery pack	14.33	2	28.66	223.29
M2 threaded rod	5	1	5	228.29
M2 washers	1	1 pack	1	229.29
M2 bolt	2	1 pack	2	233.29
Idc conector	1	1	1	234.29
Molex conector 6	0.2	14	5.6	239.89
Moxex pinout 6	0.32	7	2.24	242.13
2.45 pitch pins	0.52	2	1.04	243.17
Molex conector3	0.2	1	0.20	243.37
Molex pinout 3	0.2	1	0.20	243.57
3 axis accelerometer	4.95	1	4.95	248.52

So the total cost of this project was £248.52.

9.2 Time Management

Below is a table with the project objectives and when they were completed:

Task	Date achieved by	Over under in weeks
Getting the quadcopter to fly using a computer	29/4/18	+21
Getting the drone to detect an obstacle in its way after the computer wants to move to a desired location	5/3/18	+1
Localising itself in a pre-build scene including orientation of the quadcopter	14/5/18	+6
Creating a 3D scene with an accurate scale to be able to fly the quadcopter in without crashing	5/3/18	+0.5
Using a 3D scene to allow the quadcopter to update itself as it flies	5/3/18	-6
Getting the drone to detect something has moved in order to not get a blurred or mismatch image	Partially achieved 14/5/18	n/a
Using OpenCV tools to get as much out of the camera as possible	6/5/18	+0.14

10 Conclusions

This solution enables the user to fly the drone using a computer. Most microcontrollers rarely contain more than two DACs. To achieve a more robust control system further investigation into the exact 2.4GHz signal given off by the quadcopter controller would be needed. This would be a better way of controlling the quadcopter (bypassing the controller entirely).

In terms of the SLAM mapping, my system is able to create 3D maps which are to scale. However using single beam TOF data means that the number of points to be collected for detailed mapping is very large which represents a significant overhead.

Cameras are always an asset to the SLAM maps and edge detection used in this design makes picking out the key points easier with your own eyes. Looking for blurred images is extremely important, but this only becomes an issue when larger point clouds are produced. Ultimately the point cloud in this design needs to have 10 times the amount of data which has been collected.

Overall, this project to develop techniques to make a real time 3D mapping systems has been successful. The project has demonstrated that using low cost TOF chips, the system could create a scale model of any indoor environment.

Following on from the proof of concept highlighted the key areas to investigate further, as shown below, including other specifics of 3D mapping technologies.

11 Further work

There are a number of things highlighted in this article and a number of things which could be done better. This section looks to future projects which could be looked into for more investigation to develop this technology including:

- 1) The controlling of the quadcopter has been somewhat challenging. One of the biggest drawbacks of this system is that the controller is an open loop design. The sensors are capable of detecting obstacles, but because of the challenges in linking the raspberry pi and the MBED controller an automated approach proved highly challenging.
- 2) One key feature of the maps is the point cloud. If there were more points collected and these were combined with the camera feed, you would be able to show the room as clearly as figures [31] [32] [33].
- 3) More work could be done into combining the camera images together to form the scene. This would involve breaking down the images into segments or combining them together into a larger 3D coloured scene.
- 4) A main issue is processing time with displaying data and using the Denavit-Hartenberg transformations. A way to speed this up would be to convert it into just (x,y,z) points and use a more computer graphic approach to model the data from then onwards allowing for a wider variety of tools available.
- 5) In this project OpenCV contours were used, but due to the constraints of processing on the raspberry pi only a contour algorithm was used. OpenCV has far more uses in SLAM than was demonstrated for this project. Using more advanced features of OpenCV may help in projecting a more interactive 3D scene. This could also be a good time to look into transferring the data from an overstretched raspberry pi and use a PC to do more and faster processing for this system.
- 6) This project mainly focused on the abilities of the TOF sensors to generate maps. However the research uncovered at the start of the project showed how cameras have an important role in SLAM as much as ranging sensors. As a result, future work should look at the greater impact of cameras which could benefit a system such as this for a 3D quadcopter mapping system.

12 References

Ref number	link
1	https://www.wired.com/story/bmw-magna-innoviz-lidar-self-driving/
2	SLAM combining ToF and High-Resolution cameras Victor Castañeda Diana Mateus Nassir Navab Computer Aided Medical Procedures (CAMP) Technische Universität München, Germany
3	DTAM: Dense Tracking and Mapping in Real-Time Richard A. Newcombe, Steven J. Lovegrove and Andrew J. Davison Department of Computing, Imperial College London, UK
4	Verification of Stereo Vision Based Localization System Kenji MASUDA ^{1,2} , Simon Thompson ^{1*} , Satoshi KAGAMI ^{1,3} , and Taken KANADE ^{1,3,5} , ¹ Digital Human Research Center, AIST, Japan [*] Nara Institute Of science and Technology, Japan ³ CIIEST, JST, Japan ⁴ Japan Society Promotion of Science ⁵ Robotics Institute, Carnegie-Mellon University
5	Real-time Implementation of a Vision-aided Integrated Navigation System with a Tiny Vision Processing Camera Module Sukchang Yun ¹ , Seung Jun Lee ¹ , Byoung-Jin Lee ¹ , Young Jae Lee ¹ and Sangkyung Sung ¹ ¹ Department of Aerospace Information Engineering, Konkuk University, Seoul, Korea (Tel : +82-2-450-4176; E-mail: sksung@konkuk.ac.kr)
6	Vision-Based SLAM: Stereo and Monocular Approaches THOMAS LEMAIRE*, CYRILLE BERGER, IL-KYUN JUNG AND SIMON LACROIX LAAS/CNRS 7, Ave du Colonel Roche, F-31077 Toulouse Cedex 4, France Thomas.Lemaire@laas.fr Received November 18, 2005; Accepted January 23, 2007
7	SLAM for Dummies A Tutorial Approach to Simultaneous Localization and Mapping. By the 'dummies' Søren Riisgaard and Morten Rufus Blas
8	https://www.youtube.com/watch?v=Msd0GfT5KQc
9	Zebedee: Design of a Spring-Mounted 3-D Range Sensor with Application to Mobile Mapping Michael Bosse, Robert Zlot, and Paul Flick

10	https://ieeexplore.ieee.org/document/6919154/
11	https://arxiv.org/pdf/1703.10049.pdf
12	https://airccj.org/CSCP/vol4/csit41833.pdf
13	http://www.symatoys.com/upload/201609/13/201609131505517662.pdf `
14	https://os.mbed.com/platforms/ST-Nucleo-F429ZI/
15	http://www.i6.in.tum.de/pub/Main/Hub/Bachelorarbeit.pdf
16	http://ais.informatik.uni-freiburg.de/teaching/ss13/robotics/slides/14-slam-fastslam.pdf
17	http://eia.udg.es/~qsalvi/papers/2008-CCIAa.pdf
18	https://www.cs.utexas.edu/~kuijpers/slides/L17-FastSLAM.pdf
19	https://ieeexplore.ieee.org/abstract/document/7795967/
20	https://www.nasa.gov/directorates/heo/scan/communications/policy/GPS_History.html
21	https://www.qps.gov/systems/qps/performance/accuracy/
22	https://80.lv/articles/desert-express-creating-a-game-from-3d-scene/
23	https://80.lv/articles/making-3d-scene-your-low-poly-canvas/
24	https://gamedevelopment.tutsplus.com/tutorials/designing-3d-environments-lights-camera-polygons-action--gamedev-13910
25	https://deseng.ryerson.ca/dokuwiki/mec222:brief_history_of_computer_graphics
26	http://hrl-olddesign.informatik.uni-freiburg.de/papers/prezzo_icra09.pdf
27	https://www.kudan.eu/kudan-news/recovery-loops-slam/
28	https://ieeexplore.ieee.org/abstract/document/6942926/
29	https://www.sciencedirect.com/science/article/pii/S0921889009000876
30	https://link.springer.com/article/10.1007/s11263-006-0020-1
31	https://ieeexplore.ieee.org/abstract/document/1641869/
32	http://journals.sagepub.com/doi/abs/10.1177/1729881416657746
33	http://journals.sagepub.com/doi/abs/10.1177/0278364911410755
34	https://pdfs.semanticscholar.org/5dc2/df860751a4e339098804d94cc48a26e09098.pdf
35	https://www.sciencedirect.com/science/article/pii/S1877050915001258
36	http://theconversation.com/how-do-robots-see-the-world-51205
37	http://photodoto.com/camera-history-timeline/

38	https://www.prnewswire.com/news-releases/computer-vision-market-worth-1738-billion-usd-by-2023-676550403.html
39	https://www.prnewswire.com/news-releases/global-industrial-machine-vision-market-2017-2023-growing-demand-for-application-specific-machine-vision-systems-driving-the-12-billion-industry-300581403.html
40	https://www.tractica.com/newsroom/press-releases/computer-vision-hardware-and-software-market-to-reach-48-6-billion-by-2022/
41	https://theexpertconsulting.com/global-computer-vision-market-is-growing-cagr-of-57-6-synopsis-growth-factors-industry-segmentation-regional-analysis-and-competitive-analysis-forecast-from-2018-to-2024/
42	https://www.tractica.com/newsroom/press-releases/computer-vision-hardware-and-software-market-to-reach-48-6-billion-by-2022/
43	https://docs.opencv.org/3.4/d3/d05/tutorial_py_table_of_contents_contours.html
44	https://www.acuitylaser.com/measurement-principles
45	http://www.st.com/en/imaging-and-photonics-solutions/vl53l0x.html
46	https://www.mouser.com/pdfdocs/Time-of-Flight-Basics-Application-Note-Melexis.pdf
47	https://cdn-shop.adafruit.com/datasheets/tca9548a.pdf
48	https://github.com/johnbryanmoore/VL53L0X_rasp_python
49	https://users.cs.duke.edu/~brd/Teaching/Bio/asmb/current/Papers/chap3-forward-kinematics.pdf
50	https://onlinelibrary-wiley-com.plymouth.idm.oclc.org/doi/full/10.1002/cae.21656
51	https://ieeexplore-ieee-org.plymouth.idm.oclc.org/document/4252158/
52	https://www.sciencedirect-com.plymouth.idm.oclc.org/science/article/pii/S1474667017358184?via%3Dihub
53	https://cdn.sparkfun.com/datasheets/Robotics/hs422-31422S.pdf
54	https://www.adafruit.com/product/2809
55	https://matplotlib.org/
56	https://thepihut.com/products/raspberry-pi-camera-module?variant=758603005&qclid=Cj0KCQjwlv_XBRDrARIsAH-iRJQDB7NdtVqcgsu28kyqRqCqn0qmCoCGaiiMcfMj5TcSMsRUFv9emf0aA-vFpEALw_wcB#fo_c=1889&fo_k=2b686ab56474e4ce56fa1e8eb397b767&fo_s=gplauk
57	http://answers.opencv.org/question/75905/stitch-images-with-opencv-3-contrib-and-python-27/
58	https://www.pyimagesearch.com/2016/01/11/opencv-panorama-stitching/
59	http://wiki.ros.org/rtabmap_ros

13 Appendices

13.1 Table of abbreviations and key terms

Term	Full name
SLAM	Simultaneous localization and mapping
TOF	Time of Flight
DAC	Digital to analogue converter
PWM out	Pulse with modulation output
GPS	Global positioning system
Open loop design	A human is used at some point in the control

13.2 MBED code:

```
#include "mbed.h"
Serial pc(USBTX, USBRX); // tx, rx
PwmOut led(LED1);
AnalogOut signal_one(PA_4); //setup of analog power output
AnalogOut signal_two(PA_5); //setup for analog power steering output
PwmOut steering_up_down(D5); //setup of pwm out for forwards and back moutions
PwmOut steering_left_right(D6); //setup of left and right steering/moution controls
int i=0;
char* testinput;
int Left_right=0.5, Forwads_back=0.5;
float j=0, udpower=0, lrpower=0;
int main() {
    while(1){
        steering_up_down.period(0.001f);
        steering_left_right.period(0.001f);

        pc.printf("enter an input  ");
        char c = pc.getc();
        if(c == 'w') {
            pc.printf("incresing power\r\r\n");
            udpower=udpower+0.1; //increases power
        }
        if(c == 'a') {
            pc.printf("left power\r\r\n");
            lrpower=lrpower-0.1; //decreases power moveing the quadcopter to move left
        }
        if(c == 's') {
            pc.printf("decreasing power\r\r\n");
            udpower=udpower-0.1; //lowers the quadcopter
        }
        if(c == 'd') {
            pc.printf("right power\r\r\n");
            lrpower=lrpower+0.1; //moves the quadcopter to the right by increasing the left power
        }
        if(c == 'i'){
            pc.printf("forwads \r\r\n");
            Forwads_back=Forwads_back+1; //moves the quadcopter to the forwads by increasing the forwads pwm duty cycle
        }
        if(c == 'j') {
            pc.printf("left\r\r\n");
            Left_right=Left_right-1; //moves the quadcopter to the left by decreasing the left/right pwm duty cycle
        }
    }
}
```

```

    if(c == 'k') {
        pc.printf("back\r\r\n");
        Forwads_back=Forwads_back-1;//moves the quadcopter to the left by decreasing the forwads pwm duty cycle
    }
    if(c == 'l') {
        pc.printf("right\r\r\n");
        Left_right=Left_right+1;//moves the quadcopter to the right by increase the left/right pwm duty cycle
    }
    if(c == ' ') {
        pc.printf("stop\r\r\n");
        udpower=0; //gives no power to the quadcopter
        lrpower=1.65;//centers the power of the quadcopter to help stop it crashing
        Forwads_back=0.5;//gives half power to hold the quadcopter still
        Left_right = 0.5;//gives half power to hold the quadcopter still
    }

    signal_two = lrpower;
    signal_one = udpower;
    stering_up_down.pulsewidth(Forwads_back);
    stering_left_right.pulsewidth(Left_right);
}
}

```

13.3 Diagrams and photos

Diagram 1

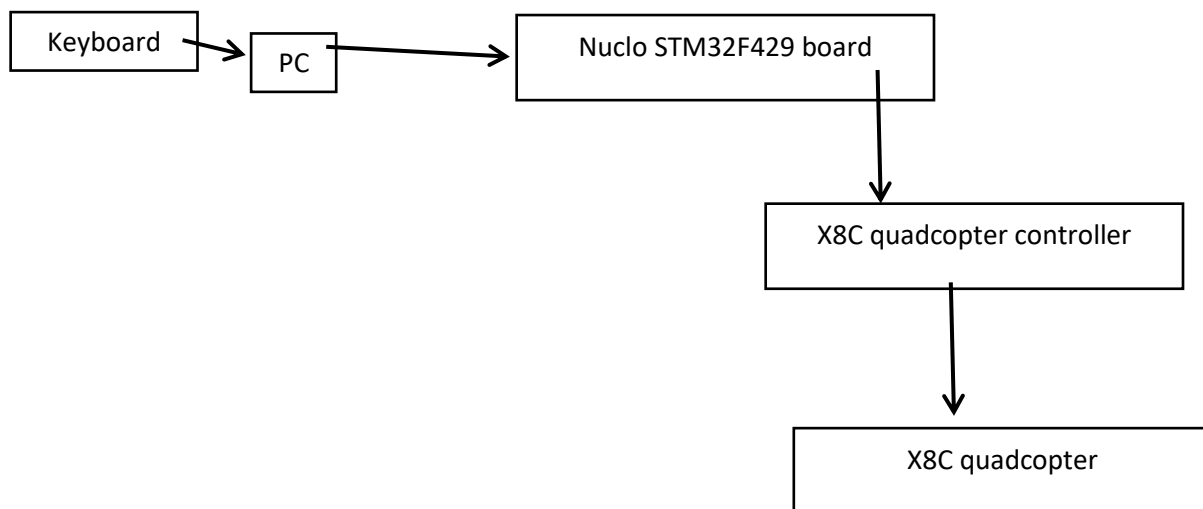


Figure 2 [13]

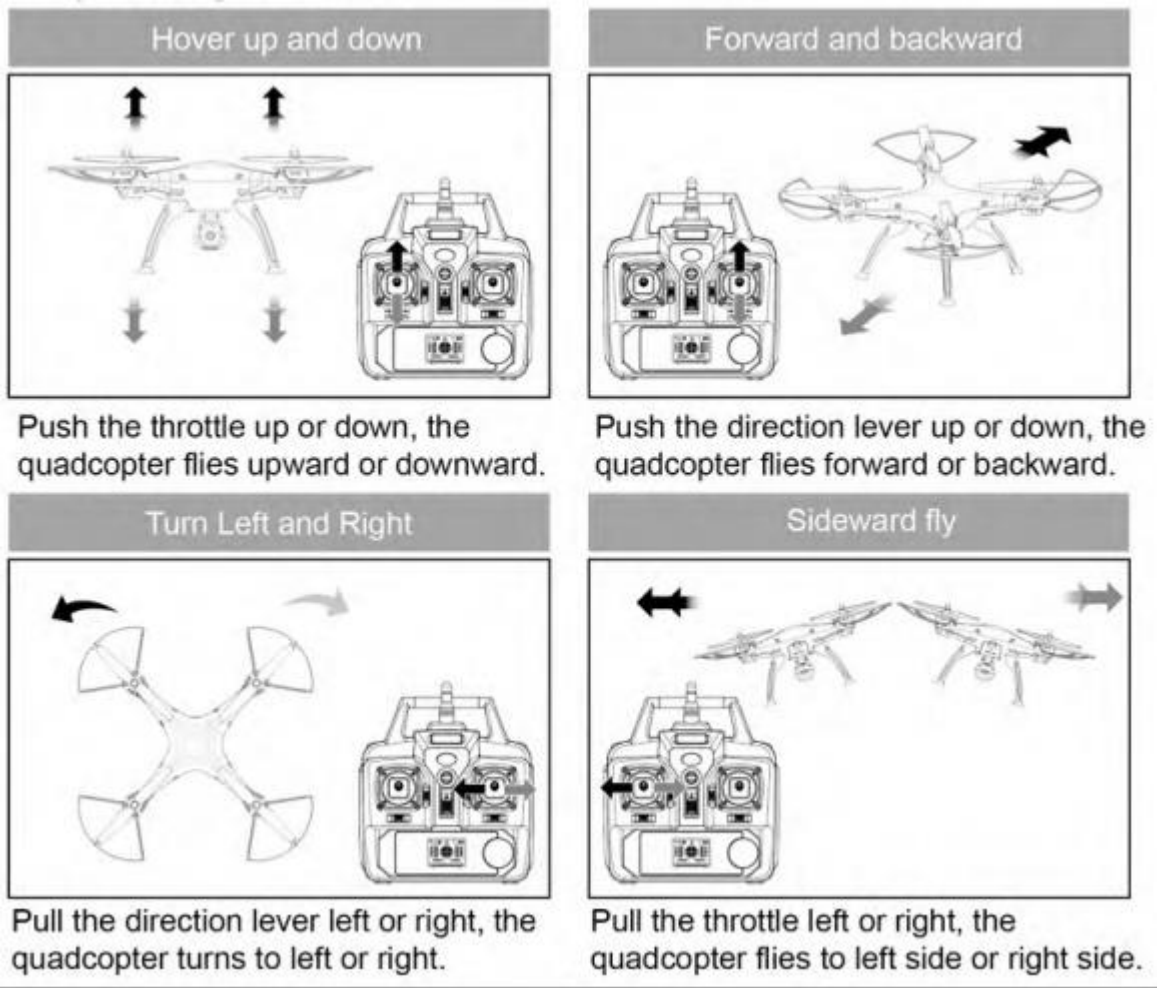


Figure 3

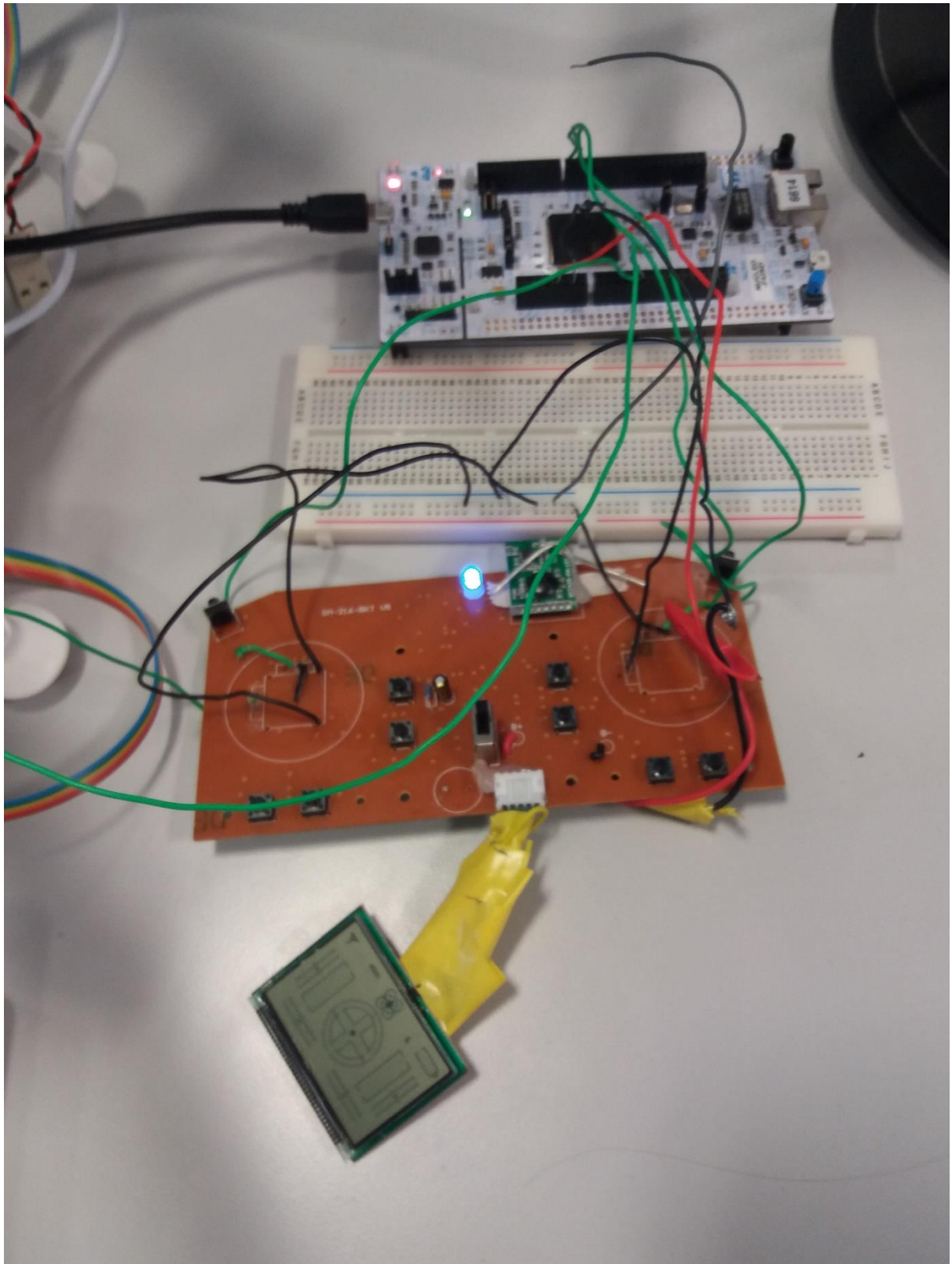


Figure 4

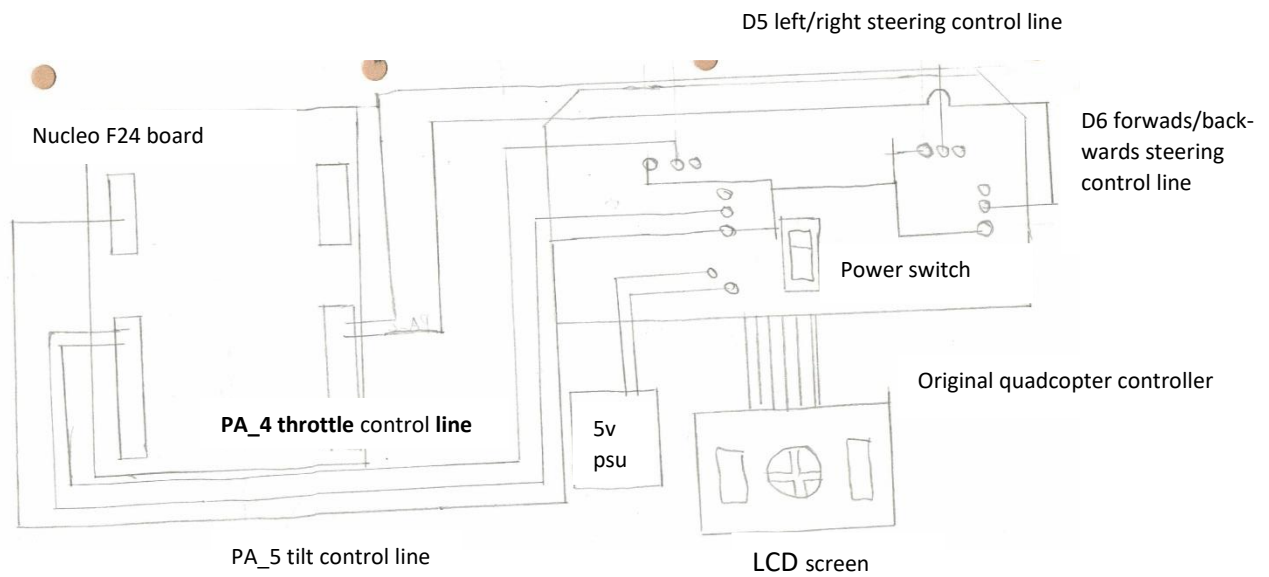
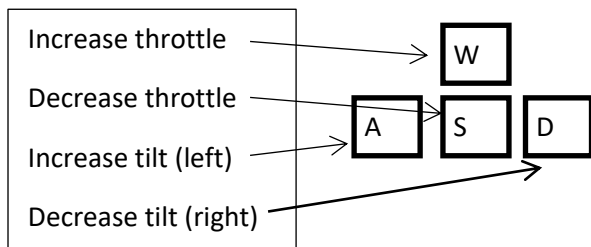


Figure 5

For throttle controls



For steering controls

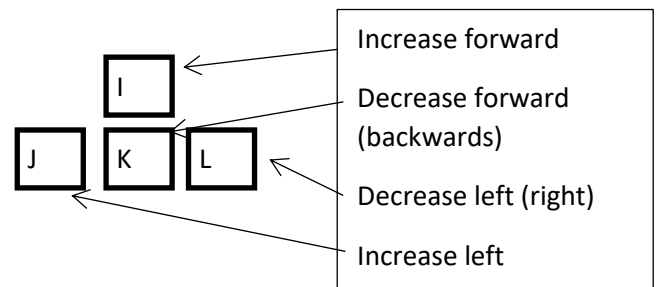
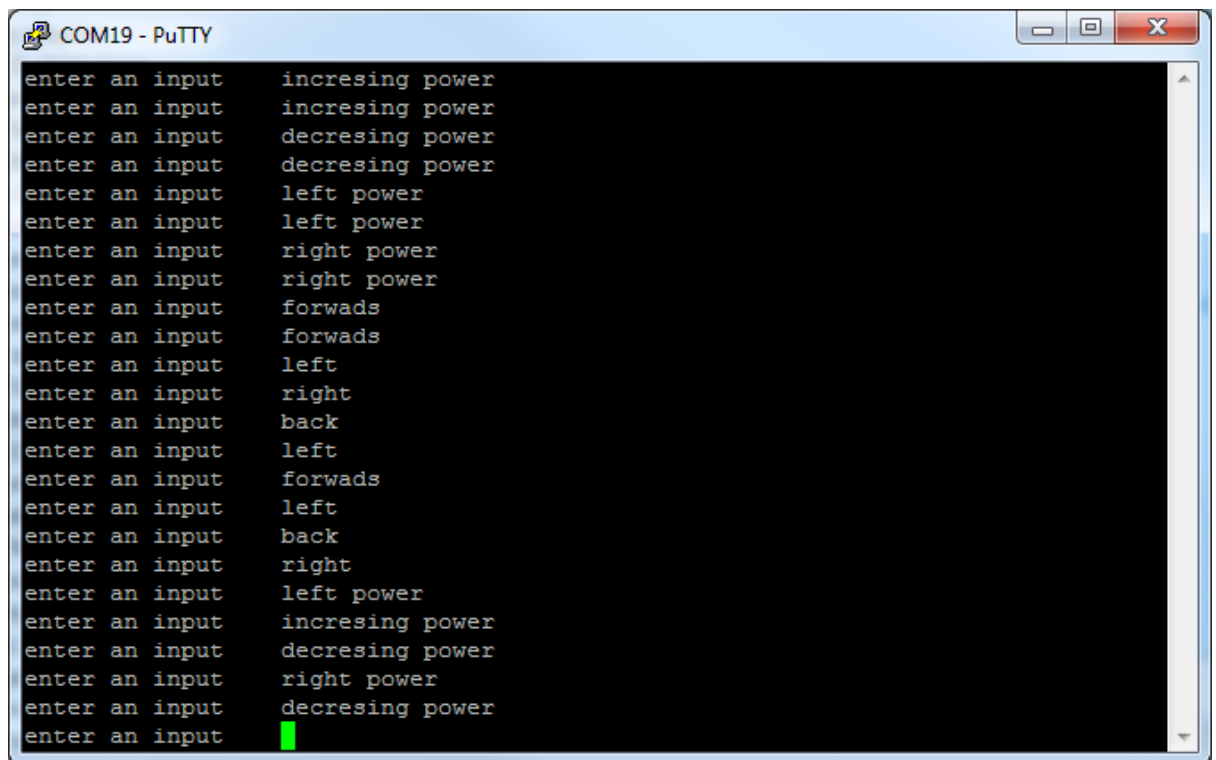
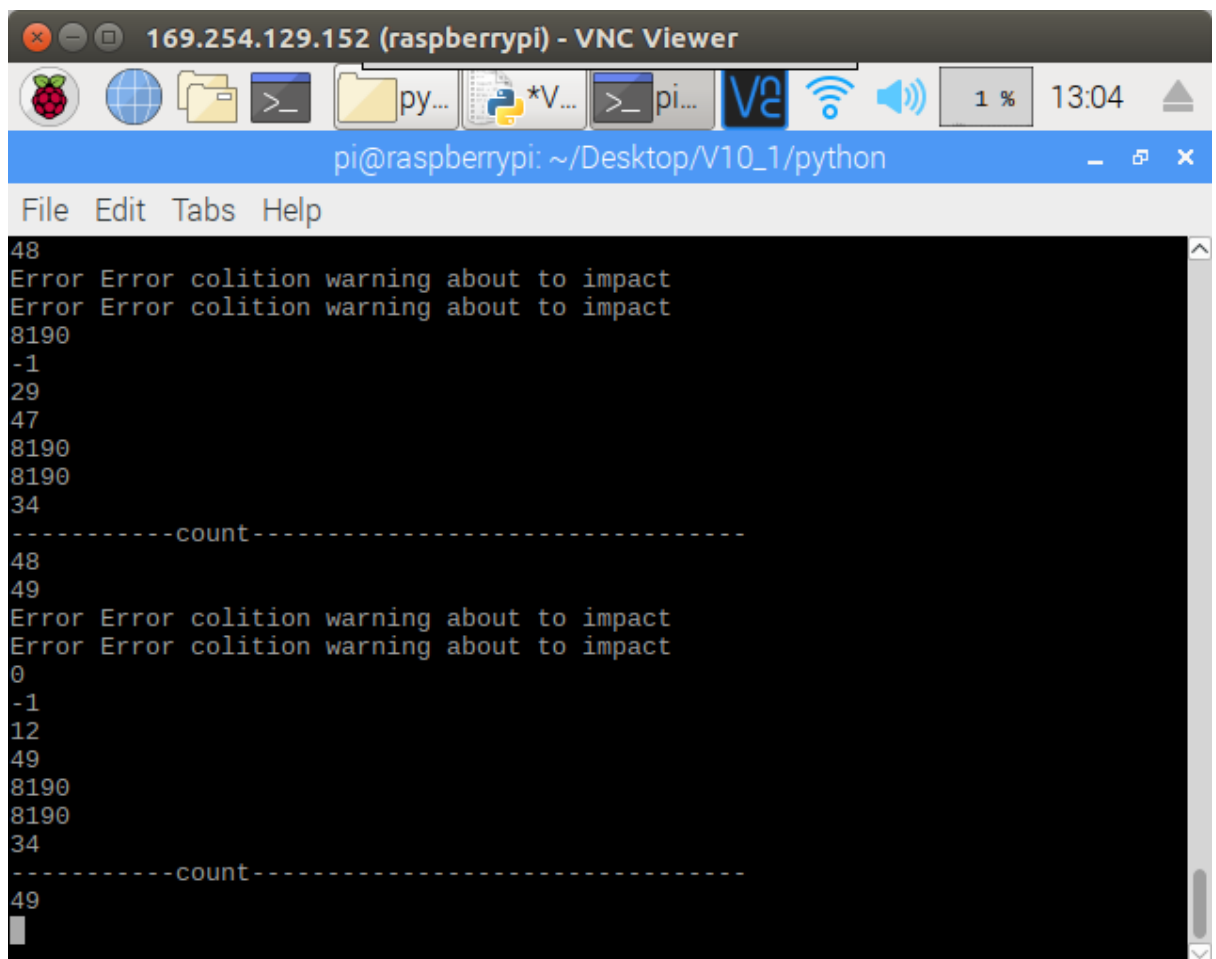


Figure 6



```
enter an input      increasing power
enter an input      increasing power
enter an input      decreasing power
enter an input      decreasing power
enter an input      left power
enter an input      left power
enter an input      right power
enter an input      right power
enter an input      forwads
enter an input      forwads
enter an input      left
enter an input      right
enter an input      back
enter an input      left
enter an input      forwads
enter an input      left
enter an input      back
enter an input      right
enter an input      left power
enter an input      increasing power
enter an input      decreasing power
enter an input      right power
enter an input      decreasing power
enter an input      
```

Figure 7



```
48
Error Error colition warning about to impact
Error Error colition warning about to impact
8190
-1
29
47
8190
8190
34
-----count-----
48
49
Error Error colition warning about to impact
Error Error colition warning about to impact
0
-1
12
49
8190
8190
34
-----count-----
49
```

Figure 8

```
pi@raspberrypi: ~/Desktop/V10_1/python
opened data file
the following number represents the number of datapoints in the map
each datapoint contains 7 time of flight sensor readings and 2 servo values
909
position0 matches for Time of Flight sensor D
position0 matches for Time of Flight sensor F
position9 matches for Time of Flight sensor D
position36 matches for Time of Flight sensor F
position63 matches for Time of Flight sensor F
position90 matches for Time of Flight sensor F
position99 matches for Time of Flight sensor D
position117 matches for Time of Flight sensor F
position135 matches for Time of Flight sensor D
position162 matches for Time of Flight sensor F
position171 matches for Time of Flight sensor F
position207 matches for Time of Flight sensor D
position243 matches for Time of Flight sensor F
position261 matches for Time of Flight sensor F
position270 matches for Time of Flight sensor F
position297 matches for Time of Flight sensor C
position297 matches for Time of Flight sensor F
position351 matches for Time of Flight sensor F
position369 matches for Time of Flight sensor F
position378 matches for Time of Flight sensor F
position396 matches for Time of Flight sensor F
position405 matches for Time of Flight sensor F
position441 matches for Time of Flight sensor F
position468 matches for Time of Flight sensor F
position486 matches for Time of Flight sensor F
position495 matches for Time of Flight sensor F
position504 matches for Time of Flight sensor F
position531 matches for Time of Flight sensor F
position549 matches for Time of Flight sensor F
position639 matches for Time of Flight sensor F
position648 matches for Time of Flight sensor F
position675 matches for Time of Flight sensor F
position693 matches for Time of Flight sensor F
position702 matches for Time of Flight sensor F
position720 matches for Time of Flight sensor D
position720 matches for Time of Flight sensor F
position729 matches for Time of Flight sensor F
position738 matches for Time of Flight sensor F
```

Diagram 2

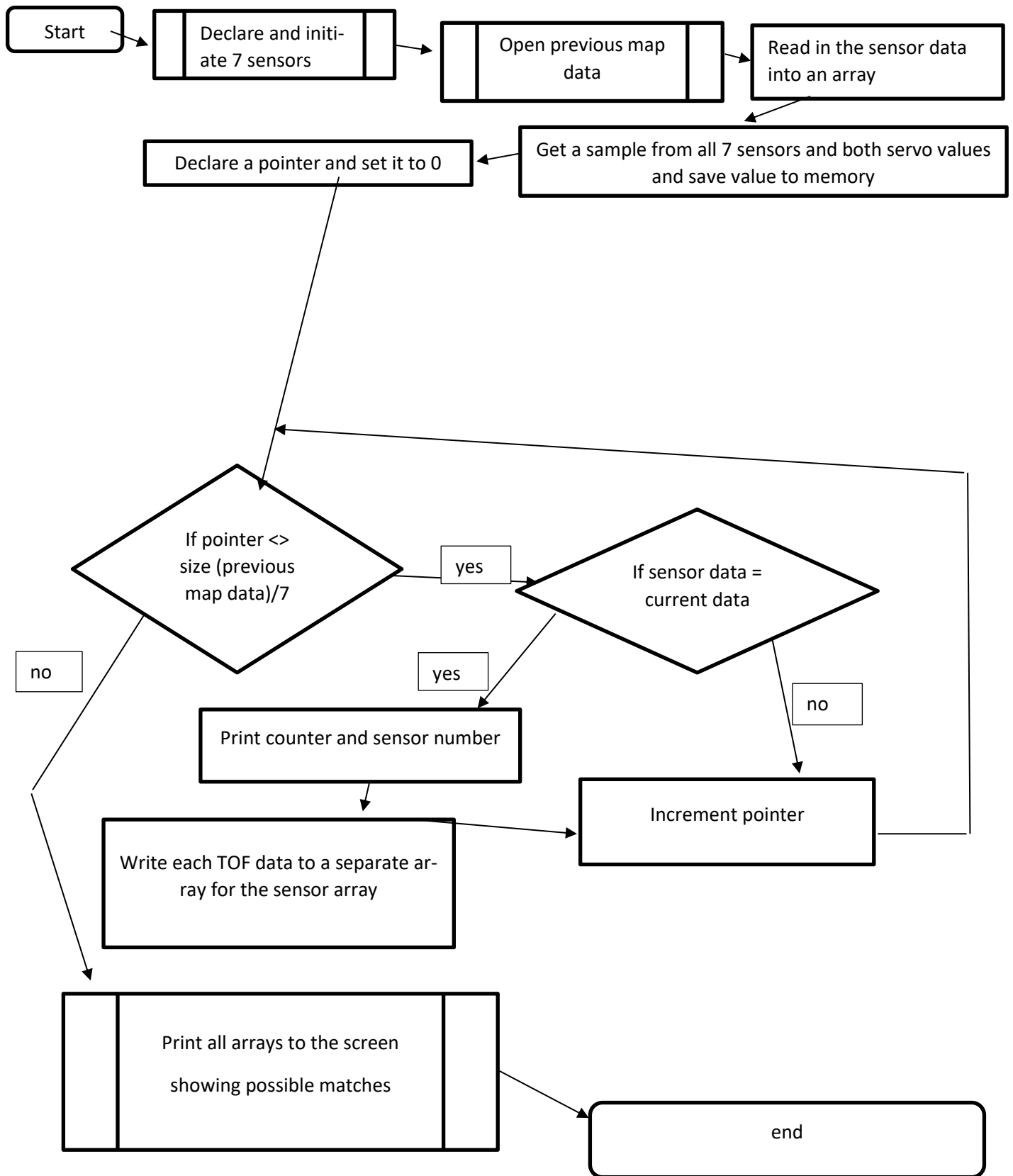


Figure 9

```
pi@raspberrypi: ~/Desktop/V10_1/python
opened data file
the following number represents the number of datapoints in the map
each datapoint contains 7 time of flight sensor readings adn 2 srvo valus
999
position0 matches for Time of Flight sensor D
position0 matches for Time of Flight sensor F
position9 matches for Time of Flight sensor D
position36 matches for Time of Flight sensor F
position63 matches for Time of Flight sensor F
position90 matches for Time of Flight sensor F
position99 matches for Time of Flight sensor D
position117 matches for Time of Flight sensor F
position135 matches for Time of Flight sensor D
position162 matches for Time of Flight sensor F
position171 matches for Time of Flight sensor F
position207 matches for Time of Flight sensor D
position243 matches for Time of Flight sensor F
position261 matches for Time of Flight sensor F
position270 matches for Time of Flight sensor F
position297 matches for Time of Flight sensor C
position297 matches for Time of Flight sensor F
position351 matches for Time of Flight sensor F
position369 matches for Time of Flight sensor F
position378 matches for Time of Flight sensor F
position396 matches for Time of Flight sensor F
position405 matches for Time of Flight sensor F
position441 matches for Time of Flight sensor F
position468 matches for Time of Flight sensor F
position486 matches for Time of Flight sensor F
position495 matches for Time of Flight sensor F
position504 matches for Time of Flight sensor F
position531 matches for Time of Flight sensor F
position549 matches for Time of Flight sensor F
position639 matches for Time of Flight sensor F
position648 matches for Time of Flight sensor F
position675 matches for Time of Flight sensor F
position693 matches for Time of Flight sensor F
position702 matches for Time of Flight sensor F
position720 matches for Time of Flight sensor D
position720 matches for Time of Flight sensor F
position729 matches for Time of Flight sensor F
position738 matches for Time of Flight sensor F
position738 matches for Time of Flight sensor F
position495 matches for Time of Flight sensor F
position504 matches for Time of Flight sensor F
position531 matches for Time of Flight sensor F
position549 matches for Time of Flight sensor F
position639 matches for Time of Flight sensor F
position648 matches for Time of Flight sensor F
position675 matches for Time of Flight sensor F
position693 matches for Time of Flight sensor F
position702 matches for Time of Flight sensor F
position720 matches for Time of Flight sensor D
position720 matches for Time of Flight sensor F
position729 matches for Time of Flight sensor F
position738 matches for Time of Flight sensor F
position756 matches for Time of Flight sensor F
position792 matches for Time of Flight sensor F
position828 matches for Time of Flight sensor F
position837 matches for Time of Flight sensor F
position891 matches for Time of Flight sensor F
opened data values
plotted data
-arr 1--
[]
--arr 2-----
[]
--arr 3-----
['29']
-arr 4-----
['54', '54', '54', '54', '54']
--arr 5-----
[]
--arr 6-----
['8190', '8190', '8190', '8190', '8190', '8190', '8190', '8190', '8190', '8190',
'8190', '8190', '8190', '8190', '8190', '8190', '8190', '8190', '8190', '8190',
'8190', '8190', '8190', '8190', '8190', '8190', '8190', '8190', '8190', '8190',
'8190', '8190', '8190', '8190', '8190', '8190']
-----arr 7-----
[]
--arr 8-----
[]
--arr 9-----
[]
pi@raspberrypi:~/Desktop/V10_1/python $
```


Figure 10

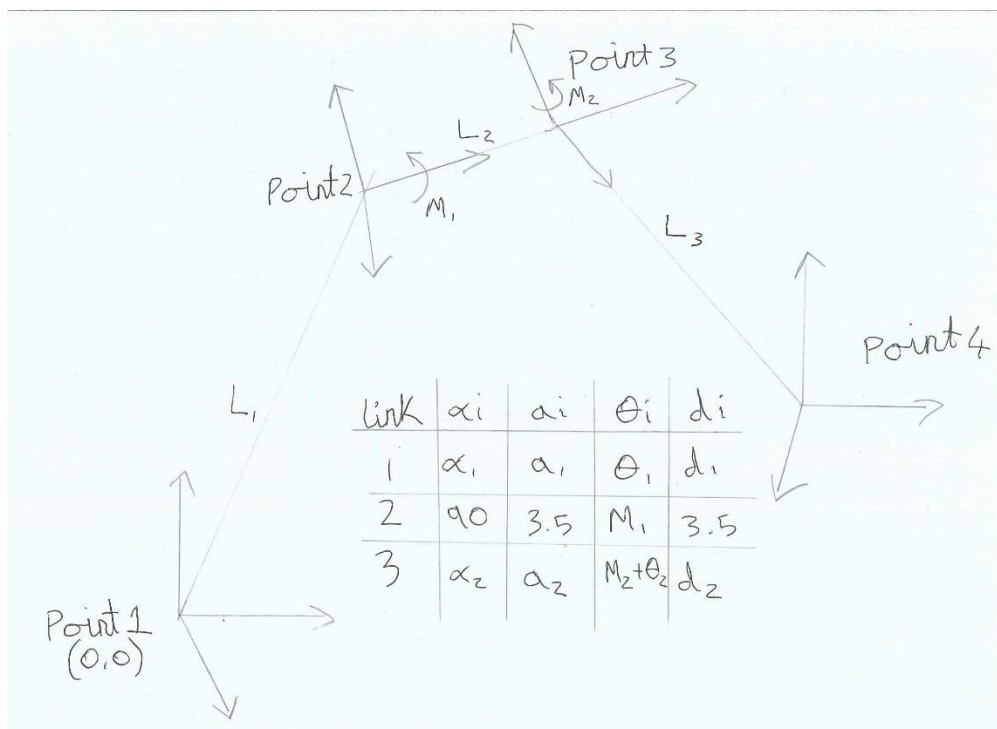


Figure 11

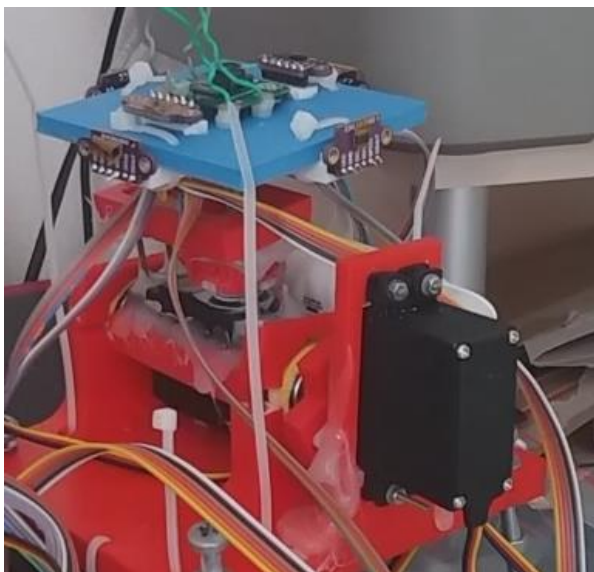


Figure 12

```
tof1 = VL53L0X.VL53L0X(TCA9548A_Num=1, TCA9548A_Addr=0x70)
```

Figure 13

```
tof1.start_ranging(VL53L0X.VL53L0X_BETTER_ACCURACY_MODE)
```

Figure 14

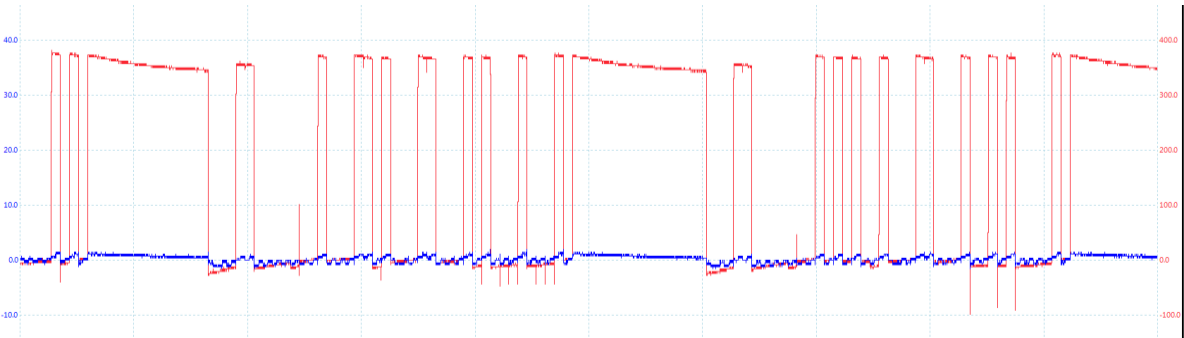


Figure 15

Channel	Name	Value	Min	Max	Average	σ	Capture Count	Span
B	Frequency	12.84 kHz	12.84 kHz	12.84 kHz	12.84 kHz	0 Hz	1	Whole trace
B	High Pulse Width	37.54 μ s	37.54 μ s	37.54 μ s	37.54 μ s	0 s	1	Whole trace

Stopped Trigger None A 0 V 50 % 0 s

Figure 16

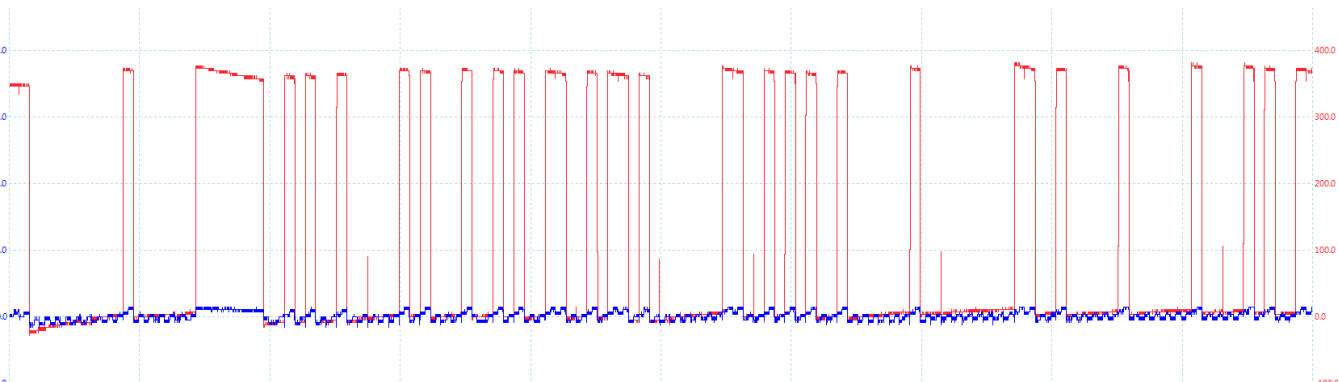


Figure 17

Channel	Name	Value	Min	Max	Average	σ	Capture Count	Span
B	Frequency	14.27 kHz	14.27 kHz	14.27 kHz	14.27 kHz	0 Hz	1	Whole trace
B	High Pulse Width	21.96 μ s	21.96 μ s	21.96 μ s	21.96 μ s	0 s	1	Whole trace

Stopped Trigger None A 0 V 50 %

Figure 18


```

for count in range(1, samples_taken):
    print(count)
    distance1 = tof1.get_distance()
    if (distance1 > 0):
        arr_0.insert(count, distance1)
        if (distance1 < 10):
            print("Error Error colition warning about to impact")
    distance2 = tof2.get_distance()
    if (distance2 > 0):
        arr_1.insert(count, distance2)
        if (distance1 < 10):
            print("Error Error colition warning about to impact")
    distance3 = tof0.get_distance()
    if (distance3 > 0):
        arr_2.insert(count, distance3)
        if (distance1 < 10):
            print("Error Error colition warning about to impact")
    distance4 = tof3.get_distance()
    if (distance4 > 0):
        arr_3.insert(count, distance4)
        if (distance1 < 10):
            print("Error Error colition warning about to impact")
    distance5 = tof4.get_distance()
    if (distance5 > 10):
        arr_4.insert(count, distance5)
        if (distance1 < 10):
            print("Error Error colition warning about to impact")
    distance6 = tof5.get_distance()
    if (distance6 > 0):
        arr_5.insert(count, distance6)
        if (distance1 < 10):
            print("Error Error colition warning about to impact")
    distance7 = tof6.get_distance()
    if (distance7 > 0):
        arr_6.insert(count, distance7)
        if (distance1 < 10):
            print("Error Error colition warning about to impact")

```

Figure 19

```

x = sensor.getX()
y = sensor.getY()
z = sensor.getZ()
print("\rX: %.6f\rY: %.6f\rZ: %.6f" % (x, y, z))

ws.write(count, 1, distance1)
ws.write(count, 2, distance2)
ws.write(count, 3, distance3)
ws.write(count, 4, distance4)
ws.write(count, 5, distance5)
ws.write(count, 6, distance6)
ws.write(count, 7, distance7)
ws.write(count, 8, str(Position_array_servo_two[c_ang_arr]))
ws.write(count, 9, str(Position_array_servo_one[b_ang_arr]))
ws.write(count, 10, x)
ws.write(count, 11, y)
ws.write(count, 12, z)

```

Figure 20

```
wb.save(n1 + ".xls")
```

Figure 21

```
import xlrd
import csv
import sys
n1 = sys.argv[1]
n2 = sys.argv[2]
def csv_from_excel():
    wb = xlrd.open_workbook(n1 + ".xlsx")
    sh = wb.sheet_by_name('Sheet1')
    your_csv_file = open(n2 + ".csv", 'w')
    wr = csv.writer(your_csv_file, quoting=csv.QUOTE_ALL)

    for rownum in range(sh.nrows):
        wr.writerow(sh.row_values(rownum))

    your_csv_file.close()

# runs the csv_from_excel function:
csv_from_excel()
```

Figure 22

Letter	Symbol and link number for table conventions	notes
A	α_1	Represents the angle change around z axis
B	a_1	Represents the height from the origin to the quadcopter
C	θ_1	Represents the angle change around y axis
D	d_1	Represents the horizontal offset from the origin to the quadcopter
E	m_1	Represents the first servo position
F	$\alpha_2 = 90$	Represents the fact the servos are 90 degrees apart
G	a_2	Can be either -5 for the vertical sensors or 2 for the horizontal
H	$m_2 + \theta_2$	represents the sensors position and the second servo position in degrees
M	d_2	This is the TOF reading +horizontal offset with is 35

Figure 23

```

for x in range(0, k, 9):
    initial_x=0
    initial_y=0
    initial_z=1
    b=(initial_x**2+initial_y**2)**(1/2)## has to be done out of order because a relies on a b value
    a=math.cos(b**2+initial_x**2-initial_z)/(2*b*initial_z)
    d=(initial_y**2+initial_z**2)**(1/2)
    c=1.0
    #a, b = divmod(j, 10)
    e=int(arr_8[i])
    f=1.0
    g=1.0
    #print("angles_one[i]")
    #print(angles_one[i])
    temp =float(arr_7[i])
    e=((float(angles_one[i])*10*math.pi)/180)
    m=3.5+temp## m becomes a baseline from scree to point
    temp=(float(angles_two[j]))
    h=temp+0 ## this can change due to srvo position 0, 90, 180, 270
    temp1=float(math.sin(h))
    temp2=float(math.sin(e))
    temp3=float(math.cos(h))
    temp4=float(math.cos(e))
    x_l=(g*temp1+35*temp2+3.5)*5
    #print(x)
    x_values.append(x_l)
    y=(g*temp3+35*temp4)*5
    y_values.append(y)
    #print(y)
    z=(g*temp3*temp2+35*temp2)*20
    z_values.append(z)
    #print(z)
    ax.scatter(x_l, y, z, c=car, marker=mark)

#####
temp =float(arr_6[i])
m=3.5+temp## m becomes a baseline from scree to point

temp =float(arr_3[i])##distance
m=3.5+temp## m becomes a baseline from scree to point
temp=(float(angles_two[j]))
e=((float(angles_one[i])*math.pi)/180)+90
h=temp+270 ## this can change due to srvo position 0, 90, 180, 270
temp1=float(math.sin(h))
temp2=float(math.sin(e))
temp3=float(math.cos(h))
temp4=float(math.cos(e))
x_l=(g*temp1+35*temp2+35)*5
#print(x)
x_values.append(x_l)
y=(g*temp3+35*temp4)*5
#print(y)
y_values.append(y)
z=(g*temp3*temp2+35*temp2)*20
#print(z)
z_values.append(z)
ax.scatter(x_l, y, z, c=car, marker=mark)

#####
temp =float(arr_0[i])##distance
m=3.5+temp## m becomes a baseline from scree to point
temp=(float(angles_two[j])+90)
e=((float(angles_one[i])*math.pi)/180)+90
h=temp+0 ## this can change due to srvo position 0, 90, 180, 270
temp1=float(math.sin(h))
temp2=float(math.sin(e))
temp3=float(math.cos(h))
temp4=float(math.cos(e))
x_l=(g*temp1+35*temp2+35)*5
#print(x)
x_values.append(x_l)
y=(g*temp3+35*temp4)*5
#print(y)
y_values.append(y)
z=(g*temp3*temp2+35*temp2)*20
#print(z)
z_values.append(z)
ax.scatter(x_l, y, z, c=car, marker=mark)

temp=(float(angles_two[j]))
e=((float(10*angles_one[i])*math.pi)/180)
h=temp+90 ## this can change due to srvo position 0, 90, 180, 270
temp1=float(math.sin(h))
temp2=float(math.sin(e))
temp3=float(math.cos(h))
temp4=float(math.cos(e))
x_l=(g*temp1+35*temp2+35)*5
#print(x)
x_values.append(x_l)
y=(g*temp3+35*temp4)*5
#print(y)
y_values.append(y)
z=(g*temp3*temp2+35*temp2)*20
#print(z)
z_values.append(z)
ax.scatter(x_l, y, z, c=car, marker=mark)

#####
temp =float(arr_5[i])
m=3.5+temp## m becomes a baseline from scree to point
temp=(float(angles_two[j]))
e=((float(angles_one[i])*math.pi)/180)
h=temp+180 ## this can change due to srvo position 0, 90, 180, 270
temp1=float(math.sin(h))
temp2=float(math.sin(e))
temp3=float(math.cos(h))
temp4=float(math.cos(e))
x_l=(g*temp1+35*temp2+35)*5
#print(x)
x_values.append(x_l)
y=(g*temp3+35*temp4)*5
#print(y)
y_values.append(y)
z=(g*temp3*temp2+35*temp2)*20
#print(z)
z_values.append(z)
ax.scatter(x_l, y, z, c=car, marker=mark)

#####
ax.scatter(x_l, y, z, c=car, marker=mark)
temp =float(arr_1[i])##distance
m=3.5+temp## m becomes a baseline from scree to point
temp=(float(angles_two[j])+90)
e=((float(angles_one[i])*math.pi)/180)+90
h=temp+0 ## this can change due to srvo position 0, 90, 180, 270
temp1=float(math.sin(h))
temp2=float(math.sin(e))
temp3=float(math.cos(h))
temp4=float(math.cos(e))
x_l=(g*temp1+35*temp2+35)*5
#print(x)
x_values.append(x_l)
y=(g*temp3+35*temp4)*5
#print(y)
y_values.append(y)
z=(g*temp3*temp2+35*temp2)*20
#print(z)
z_values.append(z)
ax.scatter(x_l, y, z, c=car, marker=mark)

if (i==9):
    i=0
    j=j+1
if (i<9):
    i=i+1
if (x>99):
    j=0

#####
print("xvalues")
print(x_values)
print("yvalues")
print(y_values)
print("zvalues")
print(z_values)
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')
plt.show()
print("plot now active")

```

Figure 24

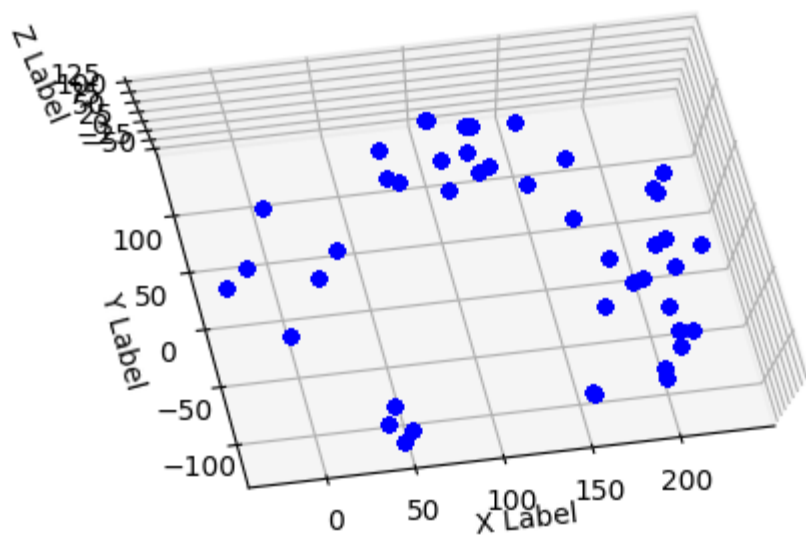


Figure 25

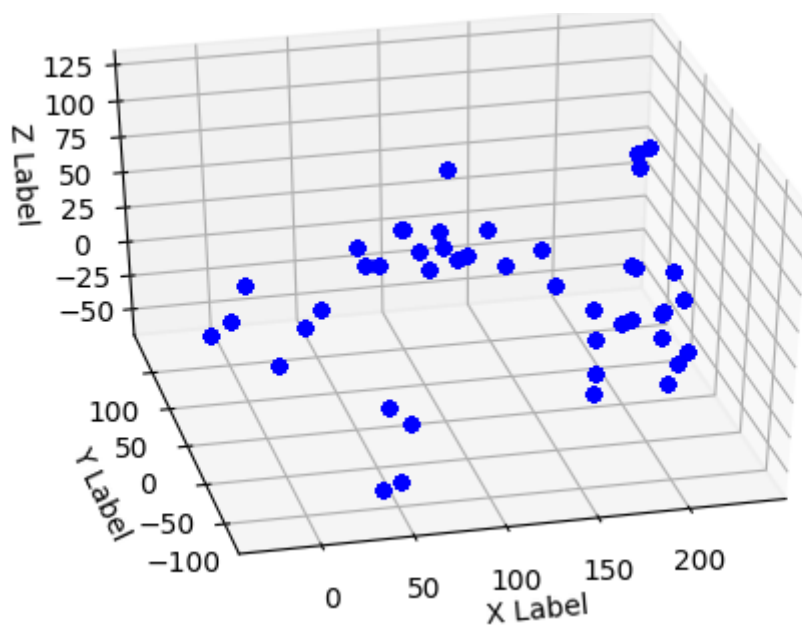


Figure 26

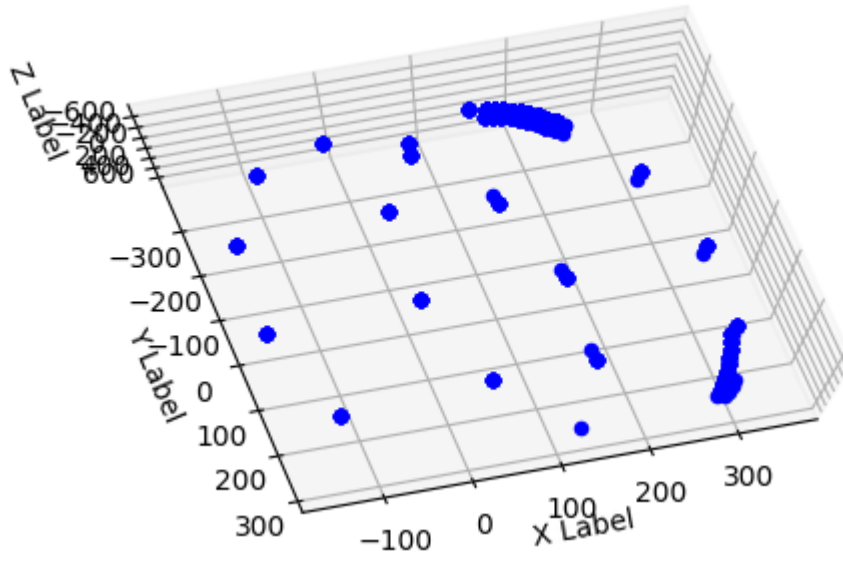


Figure 27

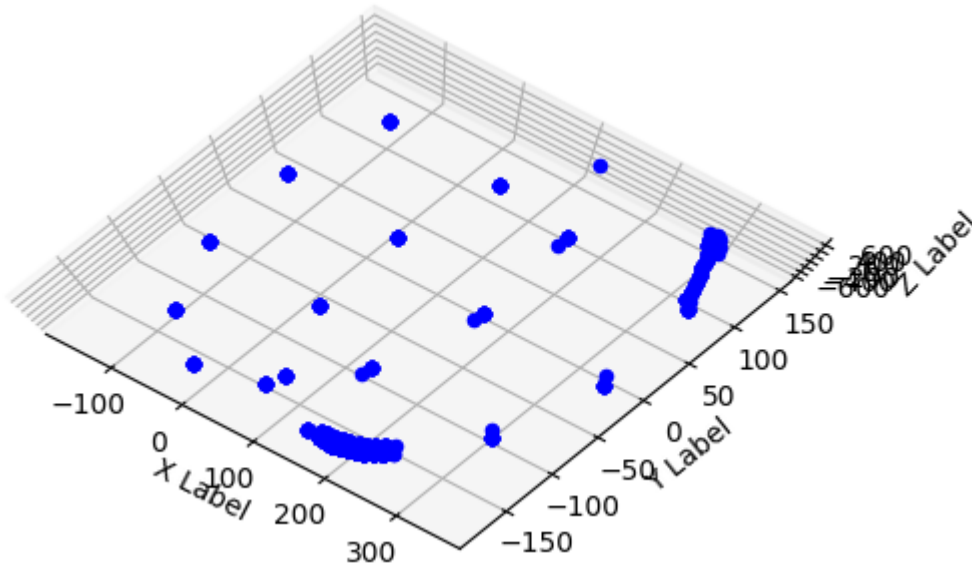
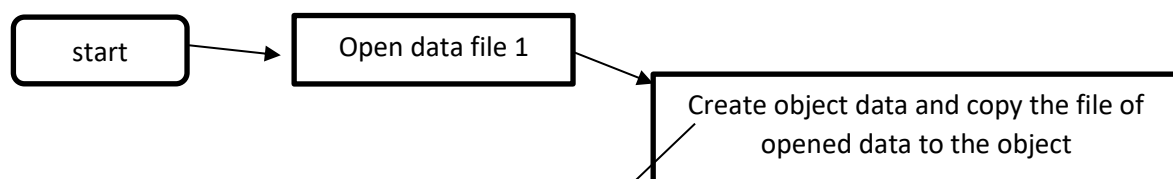
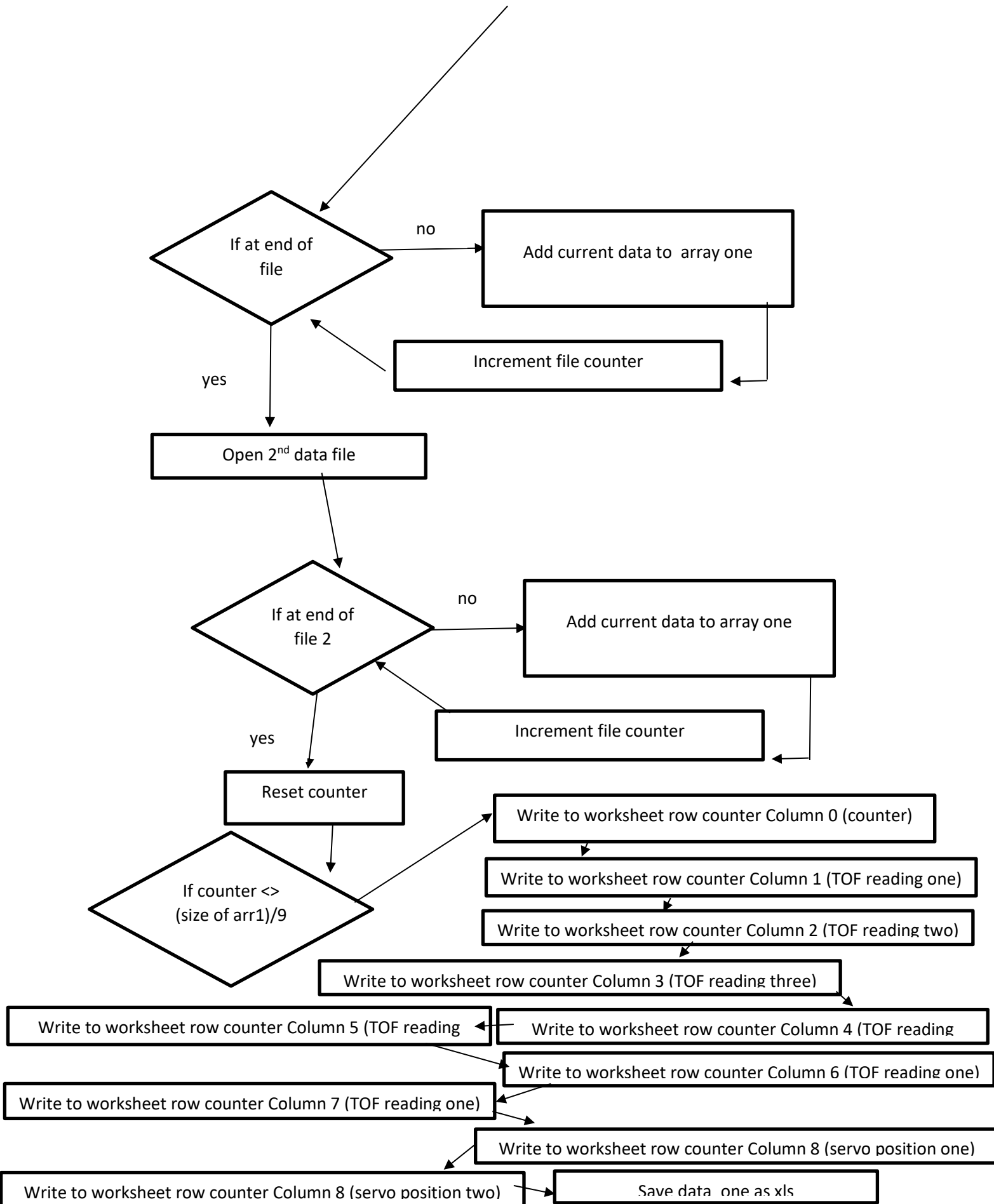


Figure 28





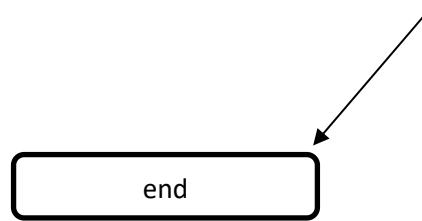


Figure 29

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
# Load an color image in grayscale
img = cv2.imread('cam6.jpg',0)

plt.imshow(img, cmap = 'gray', interpolation = 'bicubic')

plt.xticks([], plt.yticks([])  # to hide tick values on X and Y axis

edges = cv2.Canny(img,100,200)

plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([], plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([], plt.yticks([])

plt.show()
```

Figure 30

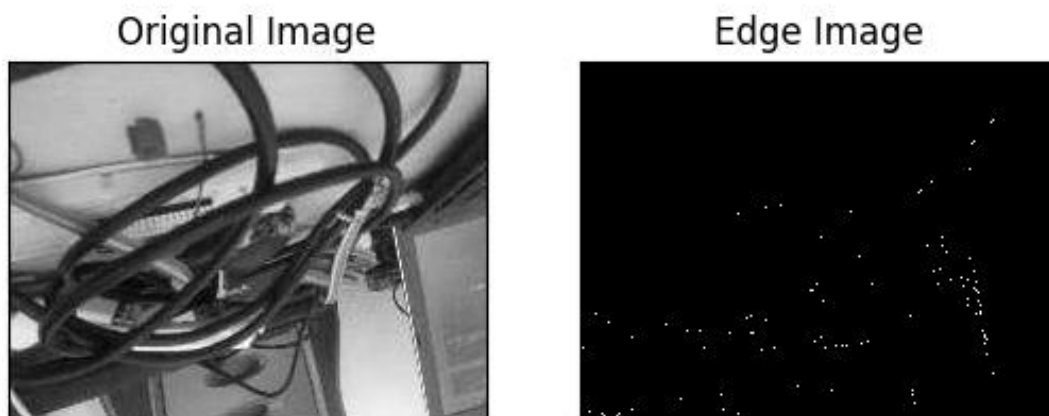


Figure 31(NOT CREATED BY THE ALGORITHM SHOWN IN THIS REPORT)

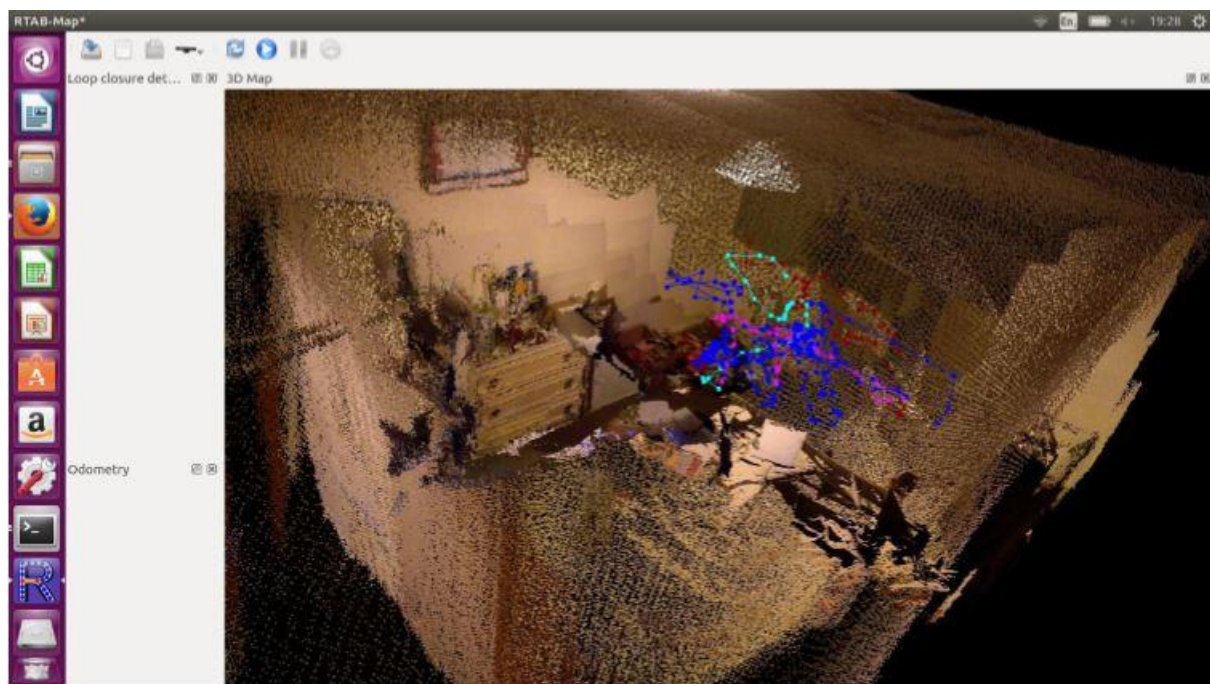


Figure 32(NOT CREATED BY THE ALGORITHM SHOWN IN THIS REPORT)

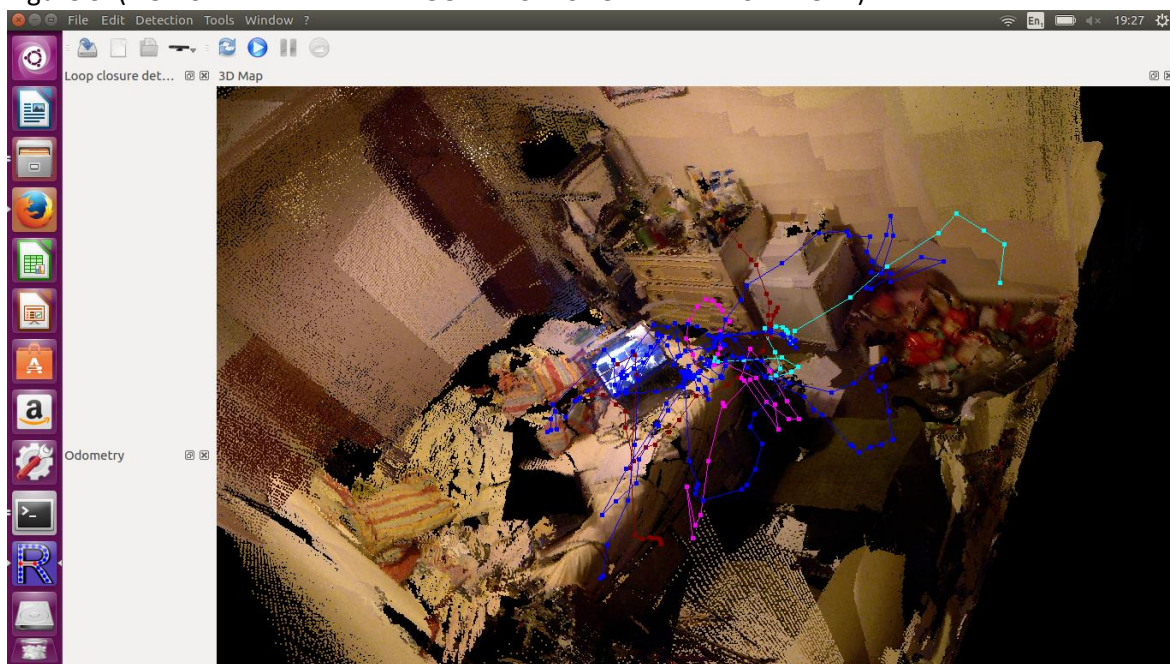


Figure 33(NOT CREATED BY THE ALGORITHM SHOWN IN THIS REPORT)

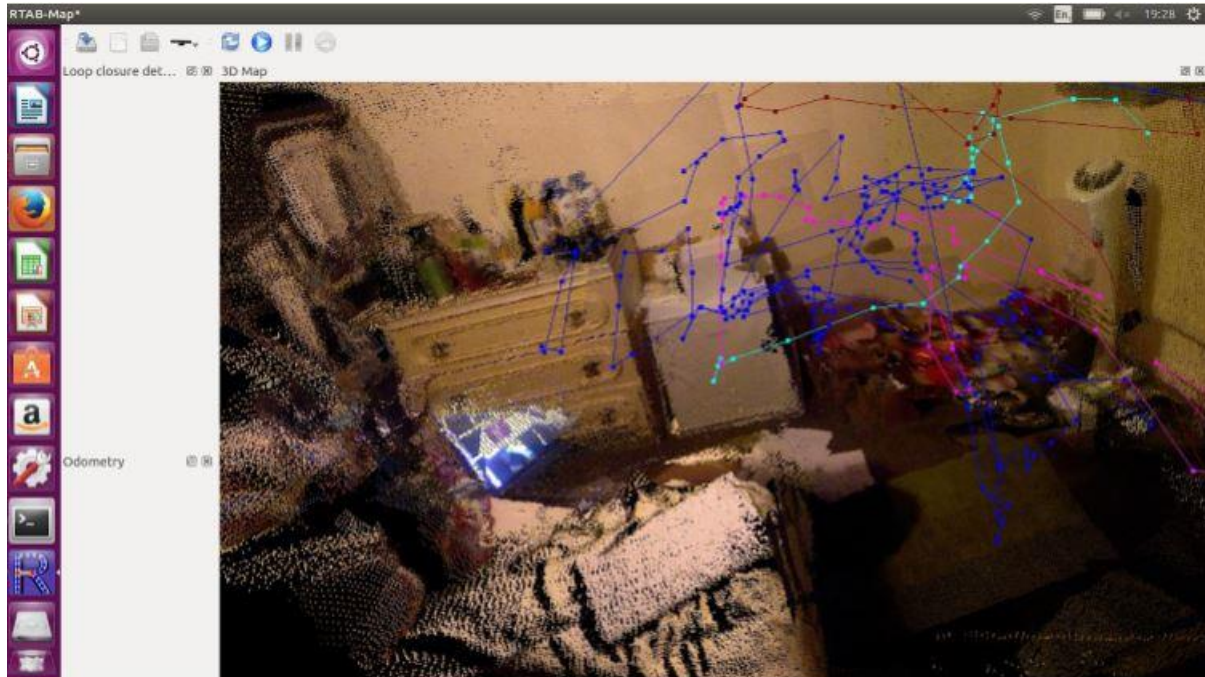


Figure 34

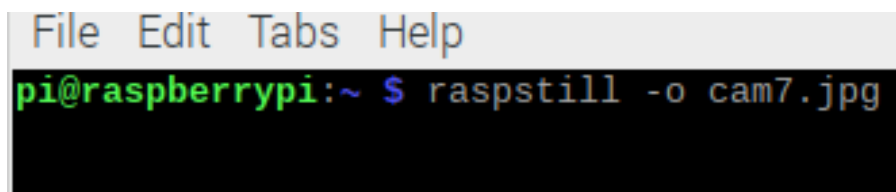
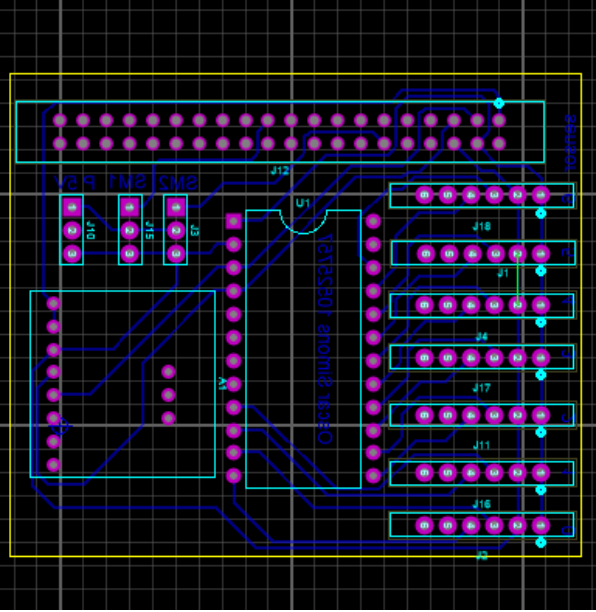
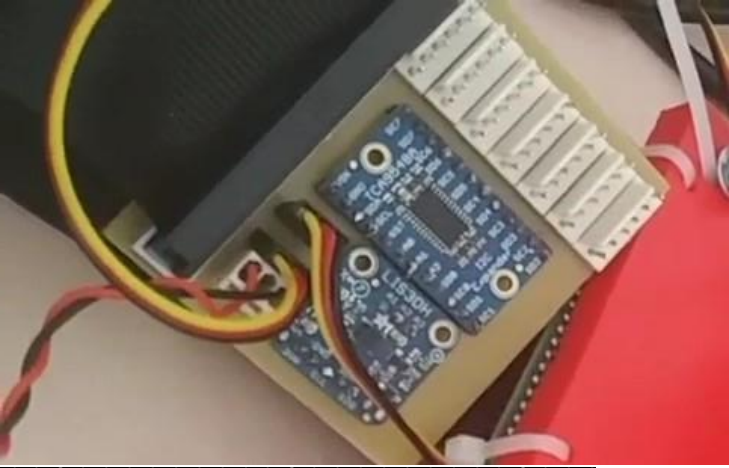


Figure 35



File Name	Size	Modified	Type
~lock.test_sun_7.csv#	1 bytes	13/05/18 20:39	csv#
test_sun_7.csv	2.8 KiB	13/05/18 15:40	csv
test_sun_6.csv	11.4 KiB	13/05/18 13:35	csv
test_sun_5_combined.csv	5.6 KiB	13/05/18 13:06	csv
test_sun_4.csv	2.9 KiB	13/05/18 12:53	csv

2 items selected (8.4 KiB) Free space: 3.7 GiB (Total: 12.5 GiB)

[illegible][illegible]