

University of Southern Queensland
Faculty of Health, Engineering & Sciences

MM-Wave Assisted Automatic Camera Tracking System

A dissertation submitted by

Andrew Jawney

in fulfilment of the requirements of

ENG4111 Research Project

towards the degree of

Bachelor of Mechatronic Engineering

Submitted: October, 2023

Abstract

Each day there are a great deal of presentations or performances occurring in front of audiences all around the world. These events are often recorded by a person aiming a camera or in other cases the camera will have an automatic tracking system. Having a dedicated camera person will usually achieve excellent tracking performance, however this task is tedious and focus can easily be diverted. For this reason camera tracking systems have been developed to automate this process. These existing tracking systems can often get distracted with movements in the audience or poor lighting conditions. Usually, machine vision is the main mechanism used for this camera tracking. However, these methods are computationally expensive and cannot operate effectively in all lighting conditions.

To address these issues, the idea of pairing a mm-wave sensor with a camera tracking system was conceived. These sensors are immune from any visual obstructions like low light, smoke or glare. Previous research has successfully used a mm-wave sensor to detect, count and monitor the positions of people in software (Huang et al. 2021). However the final step in pairing this to a physical camera tracking system has never been done. This research gap is where the final year project will be focused. The aim is that an automatic camera tracking system will be developed which is able to successfully track a speaker, with lower computation requirements while performing better in more extreme lighting conditions. This project will also attempt to answer the question of whether the idea of pairing a mm-wave sensor with a camera tracking system is advantageous when compared to existing methods.

To determine the answer to this question, a new type of camera tracking system which uses a mm-wave sensor will be developed and compared to an existing system. This project requires many steps to achieve these goals; these first few steps will be similar to the methods used in the research identified. Building on this, more algorithms need to be developed to interface with the motors inside the camera gimbaling hardware to allow smooth tracking. Once a

system is operational, some degree of software and hardware refinement will be performed to improve the design. Finally the developed camera tracking system will be compared head to head with another camera tracking system and some key performance characteristics will be measured. These results will be interpreted and the findings will be discussed in the dissertation.

University of Southern Queensland
Faculty of Health, Engineering & Sciences

ENG4112 <i>Research Project</i>

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled “Research Project” is to contribute to the overall education within the student’s chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Dean

Faculty of Health, Engineering & Sciences

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Andrew Jawney



Acknowledgments

I am extremely grateful to Dr Craig Lobsey and Dr Tobias Low, my project supervisors, without their friendly and patient guidance, this dissertation would not have been possible. I would also like to thank the University of Southern Queensland for providing the resources, funding and the knowledge needed to complete this dissertation. Lastly, I am truly gratefully to my loving family, especially my wife and children. Their continued support and encouragement over this process, has been has been truly amazing and has helped me to stay positive and focussed.

ANDREW JAWNEY

Contents

Abstract	i
Acknowledgments	viii
List of Figures	xiv
List of Tables	xvi
Chapter 1 Introduction	1
1.1 Background	2
1.2 Mm-Wave Sensors	4
1.3 Robot Operating System (ROS)	8
1.4 Literature Review	10
1.4.1 Indoor Detection And Tracking Of People Using Mm-Wave Sensor	10
1.4.2 mID: Tracking and Identifying People with Millimeter Wave Radar	12
1.4.3 MmWave Radar and Vision Fusion for Object Detection in Autonomous Driving: A Review	14
1.5 Project Specification	17
1.5.1 Scope	17
1.5.2 Objectives	18
1.5.3 Research Questions	19
1.5.4 Methodology	20
1.5.5 Feasibility	22
Chapter 2 System Design	23

2.1	Chapter 2 Overview	24
2.2	Initial Setup & Testing	25
2.3	Electronic Design	26
2.4	Motor Control.....	33
2.5	Angle Sensing	35
2.6	Overall Configuration	38
2.6	Chapter 2 Summary.....	40
Chapter 3 Mechanical Design		41
3.1	Chapter 3 Overview	42
3.2	Conceptualisation	43
3.3	CAD Design	44
3.4	Manufacture & Assembly	49
3.5	Chapter 3 Summary.....	52
Chapter 4 Software Development		53
4.1	Chapter 4 Overview	54
4.2	Implementation Strategy With ROS	55
4.3	ROS Setup.....	57
4.4	ROS Node Fundamentals	61
4.5	Modular ROS Node Creation Strategies	63
4.5.3	Adoption Strategy	64
4.5.4	Break-Down Strategy.....	65
4.5.5	Build-Up Strategy	68
4.6	Alternate Development Strategies.....	73
4.7	Chapter 4 Summary.....	75
Chapter 5 Testing & Conclusions		76

5.1	Chapter 5 Overview	77
5.2	System Testing	78
5.3	Tracking Performance	79
5.4	Resistance To Disturbances	85
5.5	Conclusions	89
5.5.1	Test Conclusions	89
5.5.2	General Conclusions	91
5.7	Chapter 5 Summary	93
References		94
Appendix A Risk Assessment		97
Appendix B Ethical & Environmental Considerations		102
Appendix C Resource Planning		104
Appendix D Project Phases		106
D.1	Gathering	107
D.2	Setup	108
D.3	Development	109
D.4	Testing	110
D.5	Documentation	111
Appendix E Project Timelines		112
Appendix F Original Stepper Control Code		115
Appendix G Command Node Script		121
Appendix H Angles Node Script		126
Appendix I Sliders Node Script		130
Appendix J Horizontal Stepper Node Script		135
Appendix K Vertical Stepper Node Script		141

List of Figures

Figure 1.1: Mm-Wave On The Electromagnetic Spectrum (Mcgrath 2021).....	4
Figure 1.2: Chirp Signal, With Frequency As A Function Of Time. (Lovescu & Rao 2020)...	4
Figure 1.3: Angle Detection Using A Phased Array Antenna (Lovescu & Rao 2020)	5
Figure 1.4: Point Cloud Generated By A Mm-Wave Sensor	6
Figure 1.5: Simple ROS System Layout.....	8
Figure 1.6: Point Cloud Data From The IWR1642 (Huang et al., 2021)	10
Figure 1.7: Data Processing Steps Used In Mm-Wave Tracking System (Zhao et al., 2019).	13
Figure 2.1: Jetson Nano Developer Kit (NVIDIA 2023).....	26
Figure 2.2: Modified PC Power Supply With 3×2.5mm Barrel Connectors (1×12V, 2×5V) .	27
Figure 2.3: Drv8825 Pinout Diagram (Last Minute Engineers, 2018)	28
Figure 2.4: Pinout Diagram For Jetson Nano (Alvarez 2022)	30
Figure 2.5: Electrical Schematic Of Camera Tracking System	31
Figure 2.6: Prototype Of Camera Tracking System Under Development	32
Figure 2.7: CAD Model Of 3382G Rotary Potentiometer (Bourns 2015)	35
Figure 2.8: Adc1115 Integrated Circuit Board	37
Figure 2.9: Block Diagram Of Mm-Wave Camera Tracking System	38
Figure 2.10: Overall Schematic Of Mm-Wave Camera Tracking System	39
Figure 3.1: CAD Model Of Mm-Wave Sensor (IWR1642BOOST)	44
Figure 3.2: CAD Model Of Camera Module	45
Figure 3.3: CAD Model Of Stepper Motor.....	45

Figure 3.4: CAD Model Of Stepper Driver Circuit Board (DRV8825 \times 2)	45
Figure 3.5: CAD Model Of Jetson Nano	46
Figure 3.6: CAD Assembly Of Mm-Wave Camera Tracking System	47
Figure 3.7: Photograph Of Mm-Wave Tracking System (1)	51
Figure 3.8: Photograph Of Mm-Wave Tracking System (2)	51
Figure 4.1: ROS System Architecture Of Mm-Wave Camera Tracking System	56
Figure 4.2: GUI For Manual Motor Controller.....	65
Figure 4.3: Corner Reflector	70
Figure 4.4: Importance Of Object ‘Y’ Coordinate For Motor Control.....	71
Figure 4.5: Mm-Wave Camera Tracing System Horizontal Range Of Motion.....	72
Figure 5.1: Maximum Tracking Range Test.....	80
Figure 5.2: Maximum Tracking Speed Test	81
Figure E.1: Gantt Chart Of Timeline For Final Year Project	114

List of Tables

Table 1.1: Comparison Of Mmwave Radar, Lidar, And Camera. (Wei et al. 2022).....	15
Table 2.1: DRV8825 Pin Names And Description (Last Minute Engineers, 2018).....	29
Table 2.2: Pin Interconnections Between 2×DRV8825's And Jetson Nano	31
Table 2.3: Mode Pin Configurations For DRV8825 (Last Minute Engineers, 2018).....	33
Table 2.4: ADS1115 Pin Names And Description (Texas Instruments 2018)	37
Table 3.1: - Comprehensive List Of Parts To Build Mm-Wave Camera Tracking System	48
Table 4.1: Required ROS Nodes.....	55
Table 4.2: Development Strategies & Node Origins	63
Table C.1: Resource Requirements & Acquisition Details	105

Chapter 1

Introduction

1.1 Background

Automatic camera tracking technology has been around for a number of years, with military camera tracking systems being developed as far back as 1976 (Redish 1976). More recently these camera tracking techniques have utilised machine vision, but these techniques are not without their disadvantages (Jaiswal & Pandey 2021). Even with the huge technological advances in recent years, the seeming simple task of tracking a speaker in front of an audience still has no perfect solution. This is evident by evaluating the camera tracking solutions which are currently available, with all systems having one or more disadvantages. With some systems requiring the user to wear a tracking tag, to other systems easily being distracted by small disturbances, a more robust solution is needed. These apparent shortcomings can be overcome by utilising the relatively new addition to the commercially available sensors which operate using mm-wave radar.

Originally, sensors which utilised mm-wave radar were developed for use in the automotive industry for self-driving applications, however the uses for this type of sensor can be much broader (Texas Instruments 2020). The technology behind mm-wave sensors has been proven to be extremely robust for object detection in all weather conditions (Chen et al. 2022). While these sensors can work well in all conditions, its sensitivity is fine enough to detect a human heartbeat or breathing from across a room (Gupta et al. 2022). According to Zhao et al. (2019), mm-wave radar uses the principle of frequency modulated continuous wave (FMCW) radar, which has the ability to measure the range and speed of the target simultaneously. Traditionally to achieve this kind of sensing performance the use of Lidar sensors would be essential, however this is no longer the case. While Lidar still has many advantages like 360° point mapping in very high resolution in nearly all weather conditions, its disadvantages make it unusable for a number of applications. With Lidar being relatively expensive, bulky and less reliable due to multiple moving parts, mm-wave sensors can be a better option as they do not have any of these disadvantages.

Despite the versatility and usefulness of mm-wave sensors, they have yet to be implemented as the main sensing technology in automatic camera tracking systems. With this gap in research literature and the need for more robust camera tracking systems, it is proposed that the inclusion of a mm-wave sensor in a camera tracking system would improve its performance significantly. It is expected that a system of this nature will be able to effectively track a speaker without

getting distracted by other objects, people or obscurities like smoke or glare. In addition to the tracking system being more robust, its cost, size and responsiveness should also be improved.

The use of a mm-wave sensor in a camera tracking system is a unique approach to improving the performance and robustness while remaining relatively inexpensive and compact. During the remainder of the year, the plan is to concentrate on developing a system of this nature for the final year project. Once the required hardware for this system is acquired, testing and development can commence. This dissertation will comprehensively outline all phases of progress for this system including research, planning, methodology, development and testing.

1.2 Mm-Wave Sensors

With the phrase ‘mm-wave’ being frequently used, it is important to have a decent understanding of what this term actually means as well as how this kind of sensor works. The mm-wave class of sensors make use electromagnetic waves (radio waves) to interpret the spatial environment at hand. The term ‘millimetre’ refers to the wavelengths used for these radar signals, this typically ranges between 1 to 10 mm, which corresponds to frequencies between 30 GHz to 300 GHz. According to McGrath (2021), this range of frequencies is also used in 5G wireless communication technology.

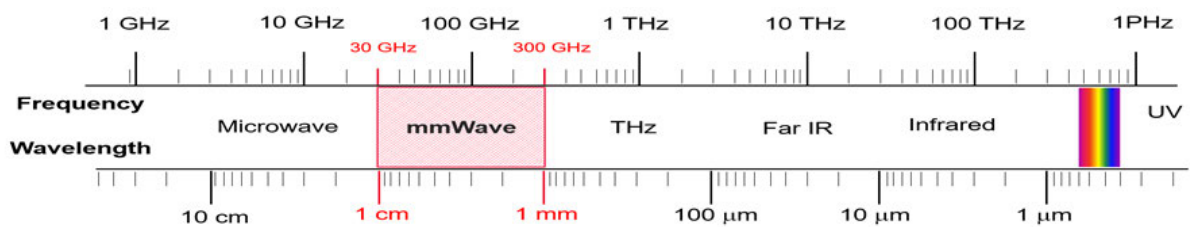


Figure 1.1: Mm-Wave On The Electromagnetic Spectrum (Mcgrath 2021)

Millimetre wave sensors operate in a similar principle to how bats use echo location to navigate in the dark, the difference being that instead of sound waves, these sensors use radio waves. The signals emitted by the sensor are called frequency-modulated continuous waves (FMCW) these are also known as ‘chirps’ (Lovescu & Rao 2020). These ‘chirp’ signals are emitted in quick succession and interact with the physical environment. When a FMCW encounters a solid object, it gets reflected back to a receiving antenna on the sensor. By measuring these reflected signals, many spatial aspects about the object can be determined.

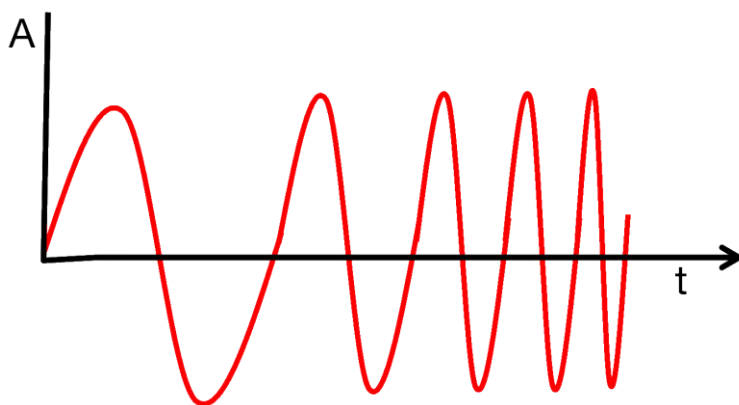


Figure 1.2: Chirp Signal, With Frequency As A Function Of Time. (Lovescu & Rao 2020)

According to Lovescu & Rao (2020) these mm-wave sensing devices are able to provide ‘high-accuracy object data including range, velocity and angle’. The range of an object is able to be determined by measuring the time it takes for the signal to return to the receiving antenna. Since the speed of the radio waves is known, the distance of the object is simple calculated by:

$$d = \frac{\tau c}{2} \quad \text{Equation 1}$$

Where d is the distance of the object, τ is the time delay between the signal transmission and receipt and c is the speed of light.

Interestingly the velocity of the measured object can also be determined by analysing the received signals. This is possible due to the Doppler effect, where the radio signals frequencies and phase are shifted due to an objects motion. Recall that phase can be described by the time difference between two waves peaks. When the object is moving closer or further away from the sensor, the frequency in the reflected radio signal is shifted. Similarly, when an object moves left or right relative to the sensor, the phase of the radio signal is shifted. By comparing the frequency and phase of the transmitted signal to the received signal, the objects velocity on a horizontal plane can be determined.

In order for the mm-wave sensor to determine the angle of a detected object, very advanced phase analysis techniques are used. This is achieved through using an array of receiving antennas which is also known as a ‘phased array antenna’. By analysing the phase changes and timing of received signals across the antenna array, the mm-wave sensor is able to calculate an objects angle relative to the centre axis (Lovescu & Rao 2020). An illustration of how this is achieved is shown in Figure 1.3 below:

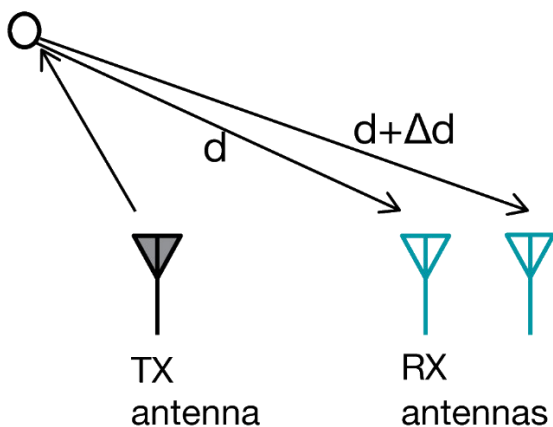


Figure 1.3: Angle Detection Using A Phased Array Antenna (Lovescu & Rao 2020)

By measuring the time it takes for a signal to travel from transmission to when it is received at each individual antenna, the total distance travelled by the radio signal can be determined for each receiver. With known dimensions for the transmitter and antennas spacing's, trigonometric calculations can be used to ascertain the objects relative angle.

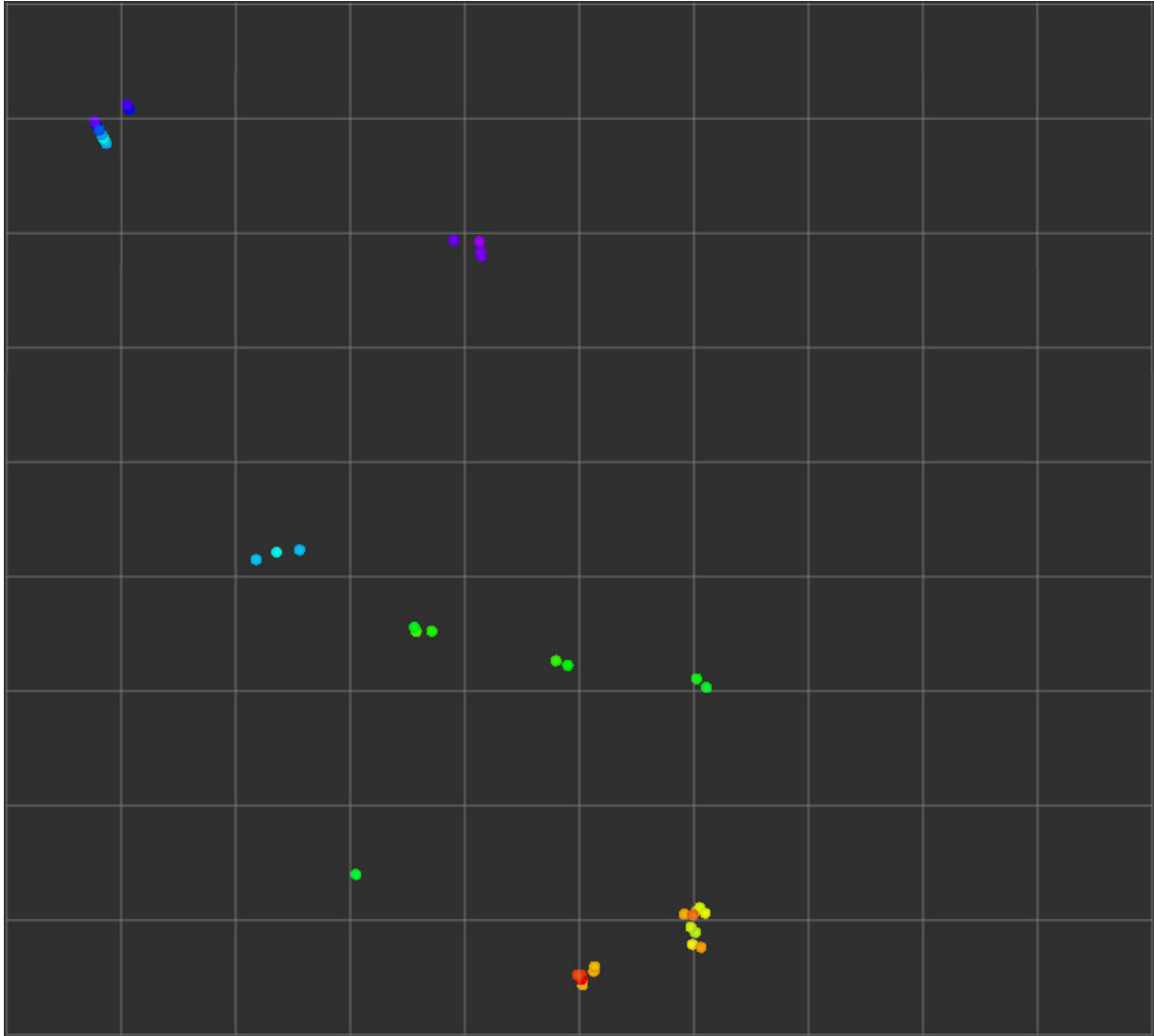


Figure 1.4: Point Cloud Generated By A Mm-Wave Sensor

By using advanced radio transmission and receiving techniques, mm-wave sensors are able to reliably and accurately measure the spatial environment. They are able to accurately measure the distance, velocity and relative angle of objects, with immunity to visual obfuscation such as poor lighting, glare or smoke. While there are many advantages of mm-wave sensors, some limitations are still evident. According to Wei et al. (2022, p. 2), mm-wave sensors ‘cannot

provide the outline information of an object', it is 'difficult to distinguish relatively stationary targets' and they suffer from 'sparseness of radar features'. Despite these drawbacks, mm-wave sensors still possess several desirable attributes which make them a intelligent choice for a wide variety of engineering applications.

1.3 Robot Operating System (ROS)

In the development of complex systems that use advanced sensors and devices, effective intercommunication can become a significant challenge. However, this problem can be effectively managed by utilising the capabilities of ROS (Robot Operating System). ROS is essentially a framework which facilitates robust and simple communication between devices or ‘nodes’. It allows the simple communication of the fundamental building blocks of a robotic system like sensors, actuators and computers. Perhaps one of the greatest aspects about ROS is that a large number of ROS packages which already exist for these kinds of devices, so it removes the need to develop specialised software to interface to devices. This makes developing complex systems a much easier experience since it can be built from readily available ‘plug and play’ ROS nodes. To understand the operation of how a ROS system works, several concepts need to be understood. The most basic ROS systems consists of the following:

- Nodes: The most basic building block of a ROS system which will perform a specific task and usually communicate with other nodes.
- Topics: Message categories used to distinguish between different messages.
- Publishers: A node that creates messages under a certain topic.
- Subscribers: A node that received messages under a certain topic

There are other ROS concepts like services, actions, parameters and packages but these are outside the scope of this dissertation. To better understand the basic operation a ROS system, a graphical representation of a simple ROS system is shown in Figure 1.4 Below:

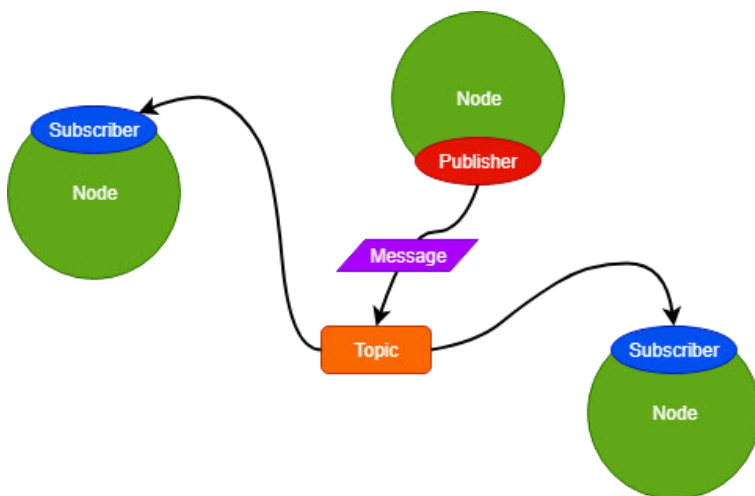


Figure 1.5: Simple ROS System Layout

Figure 1.4 shows that the publisher node produces a message, which is received by any node which is subscribed to this particular message topic. These messages can be published at any rate which is dictated by the publisher and the subscribers will receive the messages at the same rate automatically. What is not shown in Figure 1.4 is the flexible nature of a ROS system since any node can publish and/or subscribe to any number of topics simultaneously and there can be any number of nodes. To streamline the process of starting individual nodes within a system, ROS also has the capability to create a ‘launch’ file which will simultaneously start multiple nodes with a single executable file. This flexibility, modularity and simplicity allows for any number of devices to seamlessly communicate with each other, even when the system becomes very complex.

1.4 Literature Review

1.4.1 Indoor Detection And Tracking Of People Using Mm-Wave Sensor

There are many research papers that are related to this topic, which can aid in the development of methodologies for this area of research. One of the most closely related research papers in this area is from Huang et al. (2021) where it was proven that mm-wave sensors are an effective way of tracking people in an indoor environment. The tracking of individuals was achieved purely inside software, and no link to hardware through camera gimbaling was undertaken. Their research used the Texas instruments IWR1642 sensor paired with a raspberry pi 4 and tested numerous algorithms to count and track people in software. The raw data provided by the sensor needed to be interpreted, this was handled by several different algorithms which achieved different steps in processing. The majority of their research centred around comparing similar algorithms and determining the best performer at a particular task. This methodology was repeated numerous times for different tasks that resulted in a series of algorithms which produced the greatest results in relation to speed and accuracy. An example of the data generated by the mm-wave sensor is shown in Figure 1.1 Below.

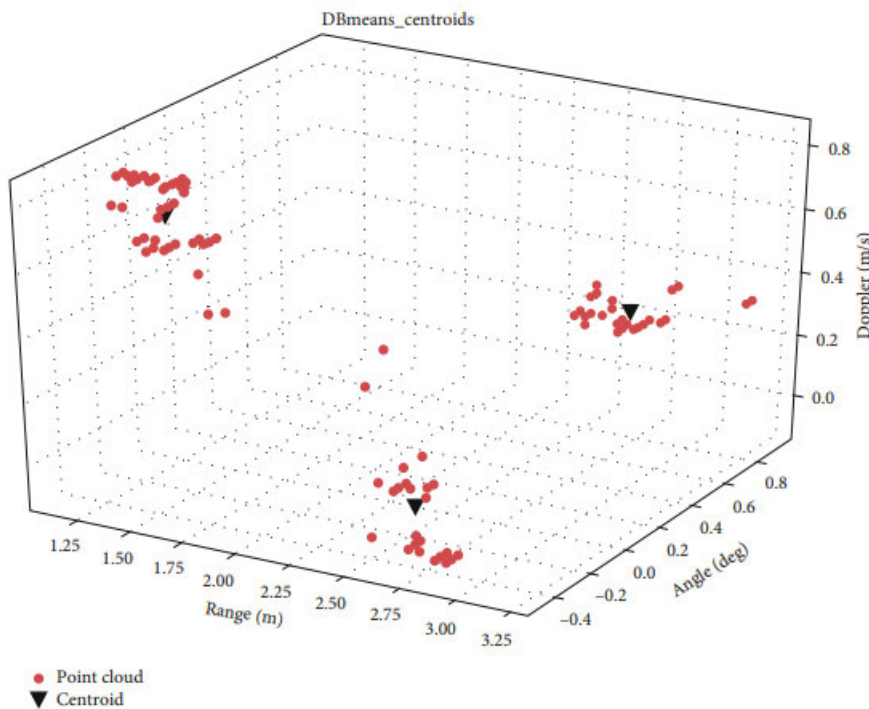


Figure 1.6: Point Cloud Data From The IWR1642 (Huang et al., 2021)

Before any algorithms were compared, the data was ‘cleaned up’ by removing any static points by use of the static clutter removal algorithm. The first comparison of algorithms was for data point clustering and it was determined that the DBmeans clustering method was faster and more accurate than the DBmedoids method. Next, the evaluation of the tracking algorithm was undertaken. Here the Extended Kalman filter (EKF) was pitted against the recursive Kalman filter (RKF), which showed that the RKF method was more than 4.5 time faster while actually providing more accurate results. Overall this research paper outlined an improved method for processing the data which is produced by the IWR1642 sensor, with improved accuracy and less than 50ms between frames.

The algorithms and methodology used in the paper by Huang et al. (2021) will be highly useful in the development of the proposed camera tracking system. It is likely that the sensor data will need to be filtered in some way to extract valuable information about the tracking targets position and speed. The methods of comparing several algorithms and selecting the best performer is a useful approach to improving the performance of the overall system, therefore the same approach will be implemented in developing the camera tracking system. It is possible that the exact methods identified in this paper are directly applicable to the camera tracking system which is to be developed. If this is the case, then a major focus will be to integrate this technology into the final design of the camera tracking system.

1.4.2 mID: Tracking and Identifying People with Millimeter Wave Radar

Another highly relevant study was completed by Zhao et al. (2019) which used a mm-wave sensor to track and identify people in software. Their research resulted in a system that could identify and track up to 12 people with an accuracy of 89%, they dubbed this system ‘mID’ which means mm-wave Identification system. The mID system was able to reach tracking accuracies of up to 95% when the group sizes were 6 or less people. An interesting finding of this study was that mm-wave sensors can be used for more than simply detecting and tracking targets; they are also capable of differentiating between individuals. A comparison between the developed mID system and the popular Xbox Kinect (V2) sensor was conducted through a number of different tests. These tests demonstrated the superiority of the mID system, with a median tracking error of 0.16m, which was more than a 5-fold improvement compared to the Kinect V2's error of 0.9m. Furthermore, the tests also showcased the improved effective tracking distance of the mID system, which was able to track targets at distances greater than 5.5m, whereas the Kinect was limited to a maximum range of 4.5m. This research paper has certainly demonstrated the benefits associated with using a mm-wave sensor.

To achieve the outstanding outcomes, the researchers devised a specific combination of algorithms to process the data generated by the mm-wave sensor which is depicted below in Figure 1.2. Upon receiving the data from the sensor, it is organised into a 3D ‘point cloud’. Any points corresponding to stationary objects are removed from the point cloud. The point cloud is then analysed using a DBScan algorithm, which merges individual points into clusters, focusing on identifying vertically-oriented objects. These clusters are subsequently tracked using the Hungarian algorithm and a Kalman filter. Targets are identified by implementing a recurrent neural network called a Long Short-Term Memory (LSTM) network. The paper further elaborates on the optimisation of sensor parameters such as frequency, bandwidth, chirp cycle time, and frequency slope to achieve optimal results.

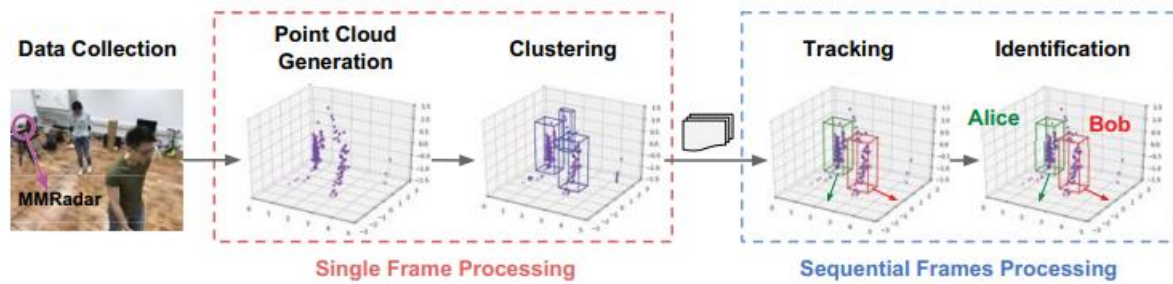


Figure 1.7: Data Processing Steps Used In Mm-Wave Tracking System (Zhao et al., 2019)

While the focus was mainly on the benefits involved with using the mm-wave sensor and the mID system, several limitations were also identified and discussed. One such disadvantage was the limited number of users the system could track. While the mID system could accurately identify and track up to 12 people, increasing this number further became troublesome. Secondly, the monitoring range seemed to be another limiting factor. The mID system had a range of 5m and had excellent performance, however the mm-wave sensor utilised had specified maximum range of 30m, but this would be at the expense of spatial precision and increased signal noise. Finally, the researchers identified an issue which occasionally occurred when the mm-wave sensor was used in proximity to extremely flat planes such as glass windows. These flat surfaces would sometimes cause ‘reflection’ objects to appear when using the mm-wave sensor.

The research undertaken by Zhao et al. (2019) has uncovered some extremely useful information and techniques in regards to developing a system that utilises a mm-wave sensor. One useful feature that the mID system possesses is the ability to identify and distinguish between different people being tracked. This feature would be particularly valuable in the upcoming camera tracking system as it would effectively eliminate disruptions. In addition to re-enforcing the appeal in utilising a mm-wave sensor, this article also provided useful techniques in extracting the data, and processing it in a way in which users can be tracked and identified. The author also identified several limitations and drawbacks with using a mm-wave sensor and serves as a valuable cautionary guide moving forward. Considering the excellent performance achieved by the mID system, the intent is to incorporate many of the useful techniques outlined in this research.

1.4.3 MmWave Radar and Vision Fusion for Object Detection in Autonomous Driving: A Review

The research conducted by Wei et al. (2022) focussed on the current state of sensor fusion techniques for object detection in autonomous driving. Most current self-driving methods will employ different combinations of cameras, LIDAR, mm-wave radars and ultrasonic radars, however this research paper mainly focusses on the first three. Each sensor type has its unique advantages and disadvantages, however with sensor fusion techniques the strengths of each sensor type can be leveraged to mitigate the specific drawbacks of other sensor types. This paper also compared the various algorithms which have been developed for object recognition and discussed the key advantages that each technique possesses. With these findings, the researchers were able to identify current trends in these sensor fusion techniques as well as make predictions on future trends. Several key areas for future research were also identified which showed potential for great improvements in this field.

Several industry leaders in autonomous driving such as Tesla, Baidu, NIO, Xpeng, Audi and Mercedes Benz have developed their own self-driving solutions which utilise different sensors and different data processing strategies (Wei et al. 2022). While a variety of other sensors are also used, all manufactures have utilised cameras and mm-wave radar sensors in their systems. This suggests this sensor combination produces the greatest combination of cost to performance when sensing the surroundings of an autonomous vehicle. LIDAR is another popular sensor which is used in autonomous driving, but companies such as Tesla and Xpeng have opted not to use this technologies. This may be due to the fact that the complimentary nature of mm-wave sensors and cameras can make up for the lack of a LIDAR system. The strengths of these three sensor types are shown in Table 1.1 below, here the complimentary nature of cameras and mm-wave radar can also be seen.

Table 1.1: Comparison Of Mmwave Radar, Lidar, And Camera. (Wei et al. 2022)

Sensor Type	mmWave Radar	Lidar	Camera
Range resolution	4	5	2
Angle resolution	4	5	6
Speed detection	5	4	3
Detection accuracy	2	5	6
Anti-interference performance	5	5	6
Requirements for weather conditions	1	4	4
Operating hours	All weather	All weather	Depends on light conditions
Cost and processing overhead	2	4	3

* ‘1’–‘6’ denote the levels from ‘extremely low’ to ‘extremely high’.

This research also identified three key sensor fusion strategies (data level, feature level & decision level) which centre around how far into the fusion algorithm the sensor data is combined. The data level fusion approach combines the sensor data very early in the fusion algorithm, this method is well established and is highly reliable, but it is dependent on the number of radar points (Wei et al. 2022). Decision level fusion combines the sensor data near the end of the algorithm, typically after objects have already been identified. According to Wei et al. (2022), this is the most common strategy for data fusion currently and makes use of all the sensing data however combining the data can be challenging due to the potential of conflicting information. Between these fusion approaches lies feature level fusion, where some level of pre-processing before the data is fused and processed further. These feature level fusion techniques are being explored further with current research. This research indicates that feature level fusion produces the best results when compared to the other approaches, but is significantly more difficult to fuse the data seamlessly. Generally the approach to overcome this challenge is to utilise some type of neural network, however this greatly increases the computation overhead that these algorithms require to operate in real-time.

Through the research conducted by Wei et al. (2022), several current trends have been identified as well as promising avenues for future research on this topic. Traditionally sensor fusion techniques will detect and sense objects in 2D, however to more accurately perceive the real world, 3D object detection is required. Some of the more recent developments in this space have begun to explore 3D object detection, however this required much more processing power. This trend to use 3D object detection is complimented by the reducing cost of LIDAR technologies which has driven the increase use of LIDAR sensors in autonomous driving.

Another recent trend which has begun development is the idea of multimodal information fusion. While sensor fusion is focussed on combining sensor data with complimentary types, multimodal information fusion will combine sensor data from vastly different sensor types. The types of sensor data that can be combined include visual, auditory, spatial, chronological, biometric, environmental and more. By combining these vastly different sensor data types with neural networks, the situational awareness is dramatically improved. These latest trends in sensor fusion methods demand significantly more processing power yet they offer superior environmental perception and robustness.

This research paper offered many valuable insights which may be applicable to this dissertation. Several key advantages of mm-wave sensors were identified in this article such as improved speed detection, good range, cost effectiveness and usability in all weather conditions. Interpreting the surrounding environment is a technical challenge which is common to both autonomous vehicles and camera tracking systems. In order to improve the accuracy, robustness and reliability of object detection and tracking, several sensor fusion techniques were analysed. The complimentary nature of combining camera data and mm-wave radar data seemed to be commonly employed in the autonomous vehicle industry, making it a potentially beneficial addition to this projects design.

1.5 Project Specification

1.5.1 Scope

The scope of this dissertation project encompasses the development and testing of software and hardware components for an automatic camera tracking system. It involves conducting a comprehensive review of relevant research to gather valuable knowledge applicable to the system's development. The project will encompass various aspects, including hardware design, fabrication, and assembly, interfacing with and testing components, as well as writing control code and iteratively improving the system's design. Advanced data analysis techniques such as Kalman filters, clustering algorithms, and possibly machine vision or machine learning techniques may also be explored.

Furthermore, the performance of the operational camera tracking system will be evaluated by comparing it to another commercially available camera tracking solution using various key metrics. This comparative analysis will provide valuable insights into the system's accuracy, robustness under different lighting conditions, and its ability to withstand environmental factors.

To ensure comprehensive documentation, all procedures followed throughout the development process will be carefully recorded in this dissertation. This documentation will also include considerations of ethical and safety concerns, timeline planning, resource requirements, and project planning.

1.5.2 Objectives

This project has a number of objectives to achieve which can broadly be categorised into main objectives, and secondary objectives. The main objectives are to:

- Develop a camera tracking system which utilises a mm-wave sensor that can steer a camera to follow a person.
- Complete this project within the allocated time including all testing, comparisons and documentation required.

Whilst maintaining focus on the two main objectives, several secondary objectives are important to ensure the overall success and effectiveness of the camera tracking system. These secondary objectives include:

- The person being tracked remains in the centre of the camera frame at all times.
- System is highly responsive, with smooth motions.
- Excellent performance in difficult lighting conditions like smoke, dust, glare or low-light scenarios.
- Immunity from external distractions such as others moving in the detection field.
- Provide method of target selection.

1.5.3 Research Questions

The purpose of this project is to investigate the idea of using a mm-wave sensor in a camera tracking system. To adequately explore this idea, several questions should be asked. These questions identify areas which require investigation in this dissertation:

- What benefits does the integration of a mm-wave sensor add to a camera tracking system?
- How feasible is the development of a camera tracking system within the allocated time and budget constraints?
- What challenges are involved with integrating a mm-wave sensor into a camera tracking system?
- What steps are involved with creating a tracking system which is highly responsive and maintains smooth tracking so a target person remains in the centre of the frame?
- What affect does modifying the lighting conditions have on the ability of the camera tracking system to maintaining its focus on the target?
- How well does the developed system respond to external distractions like other people moving around the tracked target?
- What methods and algorithms are needed to develop a system where an object can be selected for tracking?

1.5.4 Methodology

The methodologies utilised throughout this project will be highly dependent on the particular task which is being completed. In the early phases of project, the main task is centred around developing a suitable idea along with planning appropriate resources and timelines. The initial concept for the project was presented to the project supervisor, and through a process of refinement and feedback, the final project idea was formulated. This idea was further developed during the research phase where several potential approaches and techniques for achieving the project objectives were identified. Once the idea had developed enough it became possible to formulate projects timelines ([Appendix E](#)) and resource requirements ([Appendix C](#)).

Upon commencement of the project there is a notable shift in methodologies employed, transitioning from a theoretical focus to a more practical approach. In this phase, the methodology will involve testing specific components of the system to gain a better understanding of their functionality. This process will also help in identifying techniques that are effective and those that are not. The insights acquired through this process will play a pivotal role in shaping the methodologies to be employed. It is anticipated that this evolving project strategy will be engaged in both the hardware and software development stages. Additionally, in this phase it will be necessary to conduct research using various sources such as component data sheets, wiring schematics, user guides, forum posts, research papers and help documents. These techniques will be used to iteratively improve the system until the camera tracking system is operational and has achieved the project [objectives](#).

At this stage, the focus will shift to testing the developed camera tracking system in order to determine its performance characteristics. The methodologies employed during this phase will include devising and carrying out specific tests which are designed to acquire different attributes of the camera tracking system. Each test will be conducted multiple times to ensure accurate and consistent results. Subsequent tests will involve modifying the environmental conditions to assess the performance characteristics of each system in different scenarios, including low-light conditions and high glare. Additional tests will be conducted to evaluate the ability of each system to handle external distractions, assess system responsiveness, and evaluate tracking smoothness. These results will be carefully analysed in order to determine whether this new method of camera tracking has any particular advantages over traditional methods.

Upon completion of the testing phase, the full focus will then shift to writing the dissertation, which utilises other methodologies. These methods are primarily concerned with word processing, which utilises Microsoft word. Generally the strategy is to brainstorm thoughts quickly, then organise the ideas into a logical flow. Next, the ideas are transformed into refined sentences by enhancing sentence structure and word choice. This is achieved through the use of editing tools like spelling and grammar checkers, as well as consulting a thesaurus for suitable alternative for certain words. Throughout the writing process, various methods such as brainstorming, memory recollection, and research are commonly employed to develop the subject matter utilised in the dissertation.

At each stage of progression, it is crucial to employ appropriate strategies tailored to the specific task at hand. These methods may vary significantly from one another, but each plays a critical role in achieving the objective of developing an effective camera tracking system.

1.5.5 Feasibility

While the advantages of implementing a camera tracking system of this nature are clear, it is important to ensure the project remains feasible. This means that all aspects need to be completed within the allocated timeframe, while maintaining a low budget. Fortunately, the timing and budget are likely to be achievable since the project objectives have been carefully considered and are not overly ambitious. However, one particular concern arises in relation to the utilisation of machine vision techniques. Ideally, these techniques should be avoided to prevent increasing the burden on the processor and potentially fail to meet the project objectives related to system responsiveness. Nevertheless, some of these techniques may be indispensable for achieving operational functionality of the camera tracking system.

Chapter 2

System Design

2.1 Chapter 2 Overview

This chapter will outline the procedures taken to set up the components required for the camera tracking system. The initial setup and testing which was undertaken will be discussed, as well as the outcomes and lessons learned during this process. The knowledge gained was then used to design, build and assemble the electronic circuits and components that were required for the camera tracing system to function. Once the electronic design was constructed it allowed further development to occur, which ultimately saw the successful control of the stepper motors to occur.

2.2 Initial Setup & Testing

Upon receiving the necessary components from UniSQ, the initial setup and testing began. This first step included connecting Jetson Nano to a monitor, keyboard, mouse, camera and providing it with 5V via USB. Once successfully booted and logged into the Linux environment, some initial testing could take place. The first test was to make use of the connected camera, this was achieved by following the guide provided by Collins (2021). This process included writing a simple python script and executing it within the terminal in Linux; which immediately opened a window with the camera feed running. With this milestone completed, the next step was to learn how to control the output pins on the Jetson Nano. This was achieved by following a tutorial written by Kumar (2020) on how to use the GPIO pins on the Jetson Nano. This was tested by creating a simple script which toggled on and off a specific pin on the Jetson board. In order to verify this, a ground pin and the output pin were connected to a multimeter which showed the voltage changing between 0V and 3.3V periodically, as expected.

During this first phase of testing, numerous valuable lessons were learned that significantly influenced the direction taken in the electronic design process. By successfully operating the camera and an output pin, a deeper understanding of python and the Jetson Nano hardware was achieved. This also proved it was possible to interface directly with external devices such as motors, cameras and sensors. These tests also made it very clear that work was needed to allow more devices to connect to the Jetson Nano. With intentions to connect the IWR1642 sensor and two additional motors, it became evident that a unified power supply was needed to accommodate all of these devices.

2.3 Electronic Design

To progress further with the project, a suitable way of gimbaling the camera would be needed, therefore an appropriate choice of motor was necessary. Since the camera gimbaling needs fine control in angle, and does not need to rotate more than 360° , the two obvious choices become: a servo motor or a stepper motor. Having already possessed several stepper motors, along with adequate prior experience in controlling them, the decision was straight-forward, stepper motors would be used. In order to greatly reduce the complexity in controlling the stepper motors, several DRV8825 stepper driver boards were also purchased.

After finalising the hardware selection, the subsequent concern centred around the power supply for these devices. Upon inspection of the Jetson Nano and the IWR1642 board, each device is powered by 5V DC, via a 2.5mm barrel jack. Next, the datasheet for the DRV8825 stepper driver was acquired from Texas Instruments (2014), which identified the supply voltage to be between 8.2V and 45V. Ideally all of these components would be powered by one device. Conveniently, a PC power supply offers the simultaneous provision of 3.3V, 5V, and 12V with ample wattage to power these devices. Fortunately a spare PC power supply was already in possession. When AC power was supplied to the PC power supply to test the output voltages, it became apparent that there was no output voltages being produced. This issue was fixed by first dis-connecting AC power then following the advice given by McDaniel (2004), which included connecting the green wire (enable) to a black wire (ground).



Figure 2.1: Jetson Nano Developer Kit (NVIDIA 2023)

With a suitable working power supply, the connection of each device to its required voltage still needed to be achieved. As the Jetson Nano and the IWR1642 board are both supplied by 2.5mm barrel jack, the decision was made to use this type of connection for the DRV8825 stepper driver boards also. These 2.5 barrel connectors and the PC power supply are shown in [Figure 2.1](#) below. The 12V connector has yellow and black wires with red heat shrink, and the 5V connectors have red and black wires with black heat shrink.

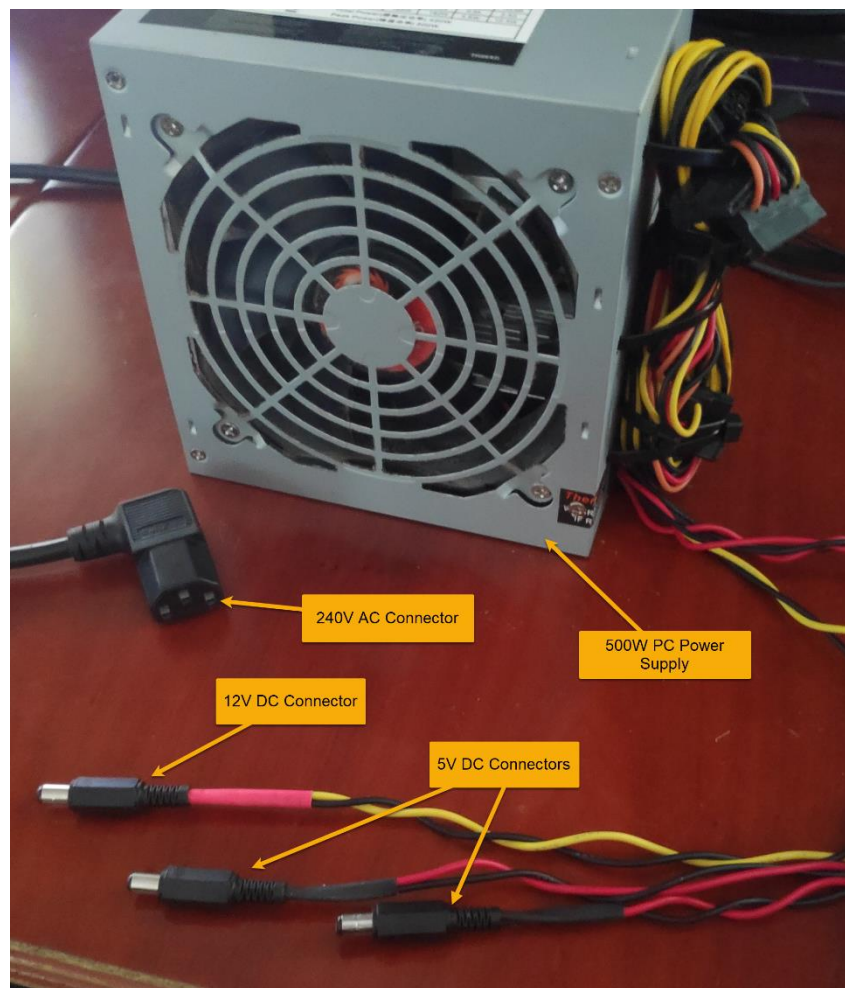


Figure 2.2: Modified PC Power Supply With 3×2.5mm Barrel Connectors (1×12V, 2×5V)

With the power supply covered, the next task was to build a circuit which would accommodate two DRV8825 driver boards. In order to do this, the driver board pinout diagram needs to be considered, this is shown in [Figure 2.2](#) below. Last minute Engineers (2018) also provides a

description of what each pin is used for, this is summarised in [Table 2.1](#) below. In order to control the operation of the DRV8825, only pins 5 to 9 are required to be connected to the Jetson Nano. Pins 5 and 6 (RST and SLP) will be connected to a single output pin of the Jetson Nano, to effectively enable the DRV8825 with a HIGH signal. The step pin (7) is also connected to the Jetson Nano and will be used to control the speed of the motor, this is achieved by regulating the frequency of HIGH pulses to this pin. The direction pin (8) is also controlled by the Jetson Nano and, as the name suggests, it controls the spin direction of the motor (HIGH = clockwise). Finally, pin 9 (GND LOGIC) needs to be connected to a GND pin on the Jetson Nano.



Figure 2.3: Drv8825 Pinout Diagram (Last Minute Engineers, 2018)

Table 2.1: DRV8825 Pin Names And Description (Last Minute Engineers, 2018)

Pin Number	Pin Name	Description	Connection
1	EN	Enable pin (active low)	Not used
2	M0	Step resolution selector pin 1	Jetson Nano
3	M1	Step resolution selector pin 2	Jetson Nano
4	M2	Step resolution selector pin 3	Jetson Nano
5	RST	Reset pin (active low)	Jetson Nano
6	SLP	Sleep Pin (active low)	Jetson Nano
7	STEP	Step Pin, each HIGH pulse will spin the motor one step.	Jetson Nano
8	DIR	Direction pin.	Jetson Nano
9	VMOT	Supply voltage pin for motor (8.2-45V)	Jetson Nano
10	GND MOT	Ground pin for motor	Jetson Nano
11	B2	Stepper output pin 1 (coil B)	Stepper
12	B1	Stepper output pin 2 (coil B)	Stepper
13	A1	Stepper output pin 3 (coil A)	Stepper
14	A2	Stepper output pin 4 (coil A)	Stepper
15	FAULT	Fault detection pin	Not used
16	GND LOGIC	Ground pin for microcontroller	Jetson Nano

In regards to the powering the motors, only pins 11 to 16 are needed. The VMOT pin (16) is used to provide adequate power to drive the stepper motor, this will be connected to +12V via the power supply. The GND MOT pin (15) is simply the ground pin for the external power supply. It is generally good practice to connect a capacitor between the positive and negative terminals on a power supply to smooth out any variations that may be present in the supply. Finally, pins 11 to 14 (A2, A1, B1 and B2) are connected directly to the stepper motor.

The DRV8825 is also capable of reducing its step size to allow for more precise motor control. The inclusion of this feature is highly valuable for a camera gimbal system, as it enables precise adjustments to the camera angle, which are often necessary. In order to make use of this feature, the mode pins (M0, M1, M2) will also need to be connected to the Jetson Nano. Since the camera gimbaling requires two stepper motors (vertical and horizontal control), there will be

a significant amount of pin connections to the Jetson Nano to accommodate this. Fortunately, the two stepper boards can share an ‘enable’ pin and a ‘ground’ pin. This configuration requires the use of 12 pins from the Jetson Nano, with 11 being output pins and 1 being a ground pin. To determine which pins are suitable to use for this purpose, the pinout diagram for the Jetson Nano is shown below (Alvarez 2022):

SoC GPIO	Linux GPIO #	Alternate Function	Default Function			Default Function	Alternate Function	Linux GPIO #	SoC GPIO
			3.3 VDC	1	2	5 VDC			
PJ.03	75	GPIO	I2C1_SDA	3	4	5 VDC			
PJ.02	74	GPIO	I2C1_SCL	5	6	GND			
PBB.00	216	AUD_CLK	GPIO	7	8	UART1_TXD	GPIO	48	PG.00
			GND	9	10	UART1_RXD	GPIO	49	PG.01
PG.02	50	UART1_RTS	GPIO	11	12	GPIO	I2S0_SCLK	79	PJ.07
PB.06	14	SPI1_SCK	GPIO	13	14	GND			
PY.02	194		GPIO	15	16	GPIO	SPI1_CS1	232	PDD.00
			3.3 VDC	17	18	GPIO	SPI1_CS0	15	PB.07
PC.00	16	SPI0_MOSI	GPIO	19	20	GND			
PC.01	17	SPI0_MISO	GPIO	21	22	GPIO	SPI1_MISO	13	PB.05
PC.02	18	SPI0_SCK	GPIO	23	24	GPIO	SPI0_CS0	19	PC.03
			GND	25	26	GPIO	SPI0_CS1	20	PC.04
PB.05	13	GPIO	I2C0_SDA	27	28	I2C0_CLK	GPIO	18	PC.02
PS.05	149	CAM_MCLK	GPIO	29	30	GND			
PZ.00	200	CAM_MCLK	GPIO	31	32	GPIO	PWM	168	PV.00
PE.06	38	PWM	GPIO	33	34	GND			
PJ.04	76	I2S0_FS	GPIO	35	36	GPIO	UART1_CTS	51	PG.03
PB.04	12	SPI1_MOSI	GPIO	37	38	GPIO	I2S0_DIN	77	PJ.05
			GND	39	40	GPIO	I2S0_DOUT	78	PJ.06

Figure 2.4: Pinout Diagram For Jetson Nano (Alvarez 2022)

The corresponding pin connections which were decided are shown in [Table 2.2](#) below:

Table 2.2: Pin Interconnections Between 2×DRV8825's And Jetson Nano

	Stepper Board Pin Name & Number		Jetson Nano Pin Number
Shared pins	GND	16	9
	ENABLE	5&6	7
Stepper 1	M0	2	11
	M1	3	13
	M2	4	15
	STEP	7	19
	DIR	8	21
Stepper 2	M0	2	29
	M1	3	31
	M2	4	33
	STEP	7	35
	DIR	8	37

The colours in Table 2.2 indicate the wire colour used to interconnect the two DRV8825's to the Jetson Nano. To further clarify how this system is connected together, a full system schematic is shown in Figure 2.4:

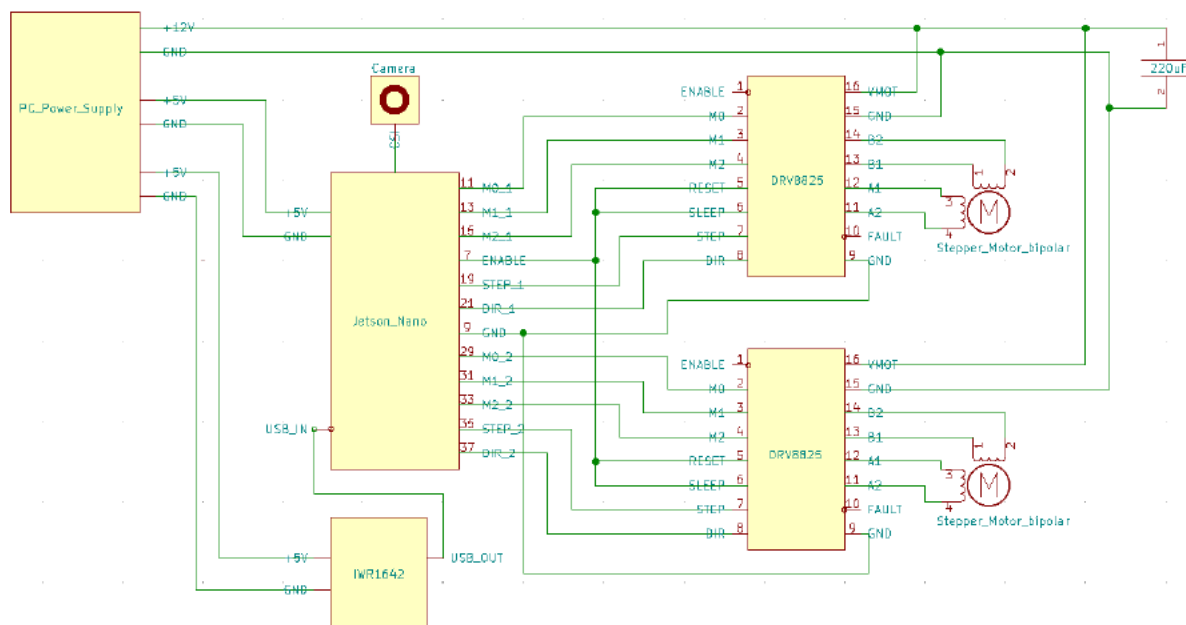


Figure 2.5: Electrical Schematic Of Camera Tracking System

Using this schematic and the acquired hardware components, the system could finally be assembled, this is shown in [Figure 2.5](#):

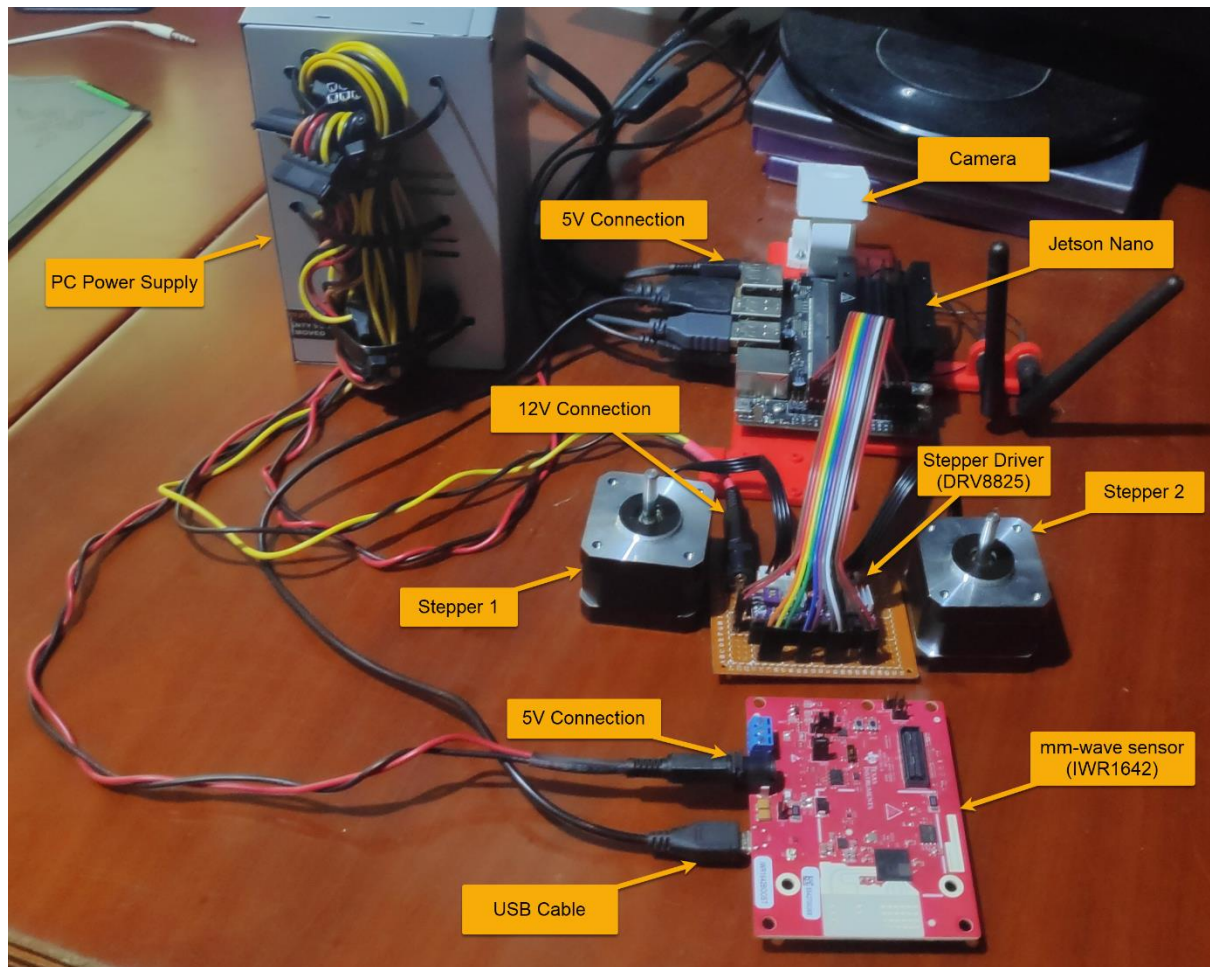


Figure 2.6: Prototype Of Camera Tracking System Under Development

2.4 Motor Control

Following the initial assembly and connection of all the necessary equipment, the software which is responsible for controlling the stepper motors can begin development. The first step was to write some test code in a python script to verify the hardware connections are correct. This test code consisted of assigning pin numbers to suitable variables, setting each pins output accordingly and cycling the step pin between HIGH and LOW repetitively. This initial test successfully verified that the hardware connections were correct and that the steppers could be controlled within software. This code was modified numerous times to test aspects like rotation speed (by decreasing the delay between step pulses), spin direction (by changing the direction pin). Altering the micro-step resolution was also tested, this was achieved by setting the mode pins in accordance with Table 2.3 below (Last Minute Engineers, 2018). These pins allow the stepper motor to operate in up to 6 different modes of step resolution.

Table 2.3: Mode Pin Configurations For DRV8825 (Last Minute Engineers, 2018)

M0	M1	M2	Micro-step Resolution
LOW	LOW	LOW	Full Step
HIGH	LOW	LOW	1/2 Step
LOW	HIGH	LOW	1/4 Step
HIGH	HIGH	LOW	1/8 Step
LOW	LOW	HIGH	1/16 Step
HIGH	LOW	HIGH	1/32 Step
LOW	HIGH	HIGH	1/32 Step
HIGH	HIGH	HIGH	1/32 Step

The test code used up until this point were able to verify the capability of the current apparatus in numerous ways, however this method of motor control was very limiting, and needed to be refined further. One major refinement was to develop a method to run the stepper motors in the ‘background’ while other tasks (like reading sensors) could be completed simultaneously. This was implemented by utilising the ‘time’, and ‘threading’ libraries which provided the capability to run a section of code at consistent time intervals. This method also allowed for easy throttling

of the stepper motor, however it became apparent that this control code would become very messy and unorganised when trying to scale up to two steppers. For this reason, it was decided to create a 'class' to handle all aspects concerned with controlling a stepper motor. The advantage of using a class is that it only needs to be defined once and can be reused multiple times, similar to a function but with enhanced capabilities. The class which was created performs many actions such as: initialising a stepper motor, sending the pulse signals, storing all relevant values for the stepper, and manipulating these values in real time. After a long process of testing and improvement, a class called 'Stepper' was created which could adequately control a stepper motor in the 'background'.

This was a major step in the software development, however more functionality needed to be added. The disadvantage of the code at this point was that the stepper speed and direction could only be set once at the beginning of the class running, a method was needed to update the stepper motor in real time. A convenient method was devised which used a pop-up window with a sliding bar that corresponds to a steppers rotational direction and speed. This pop-up window was achieved by making use of the 'tkinter' library in python. Conveniently, the functionality of the sliding bar could be added into the already existing 'Stepper' class created earlier. This also made it easier to use the values generated by the sliding bar inside the class in order to manipulate the values responsible for controlling the steppers.

The final step in developing the code for controlling the stepper motors was to modify the 'Stepper' class so it could accommodate two stepper motors. This included adding another attribute for the 'Stepper' class to distinguish between two different stepper motors which use different pins. In addition to this, the code responsible for the sliding bar needed modification to accommodate another slider that controlled the second stepper. The successful implementation of simultaneously controlling two stepper motors represented the final milestone of code development for this stage.

2.5 Angle Sensing

During the development and testing of the stepper motors it became clear that there needed to be some kind of feedback so the orientation of the horizontal and vertical joints could be determined. Upon initial execution of the code, the system will have no idea of how it is currently orientated so it may try to move in ways that are outside its range. While it is common for devices like 3D printers to use limit switches to determine their orientation by counting the steps from a known home position, this approach could not be effectively utilised for this application. While testing and running the stepper motors, it was observed that occasionally the stepper motor/s would ‘miss’ or ‘skip’ step/s during operation. Even through thorough investigation into the code, the electrical connections and the adjustment of the current control on the DRV8825, the root cause of this issue could not be definitively identified. It was for this reason it was decided to use an angle sensor.

There were several factors in deciding the type of angle sensor that would be used for this project. One common method for determining the orientation of a motor is a hall effect sensor, however with no prior experience with these, and the uncertainty with their compatibility with stepper motors, this sensor type was ruled out. It would have been preferable to have an angle sensor which could be placed directly onto the shaft of the stepper motor to simplify the physical design. After searching for suitable angle sensors, the chosen sensor was the 3382 rotary potentiometer. Although this angle sensor cannot directly be connected to the stepper motors shaft, its price, availability and specifications where the deciding factors. According to Bourns (2015), the maximum voltage for the 3382 is 16V and the detection resolution is essentially infinite, which makes it ideal for this project. Figure 2.6 shows a CAD (Computer Aided Design) representation of the surface mount version of the sensor (3382G):

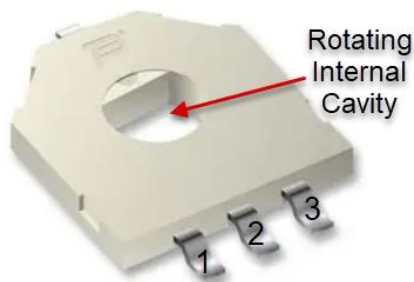


Figure 2.7: CAD Model Of 3382G Rotary Potentiometer (Bourns 2015)

As shown in Figure 2.6 above, the 3382G has 3 pin connections and an internal rotary cavity. This device is essentially a potentiometer with the internal cavity being able to rotate, which changes the resistance value between the centre pin and the outer two. As the internal cavity is rotated one direction, the resistance value between pin 1 and 2 will increase, while the resistance between pin 2 and 3 will decrease or vice-versa. The way this device is used is to connect the positive and ground connections to pin 1 and 3, and the 2nd pin will output a variable voltage level depending on how the internal cavity is orientated.

Upon receiving the 3382 angle sensors, the next necessary step was to test the operation of the device. It became apparent that the Jetson Nano does not have the capability to receive analogue inputs like many other micro controllers. For this reason it was necessary to purchase a suitable IC (Integrated Circuit) which can convert from analogue to digital and relay the data to the Jetson Nano. This analogue to digital converter (ADC) must:

- Run on 5V or 3.3V (Jetson Nano power pins)
- Detect a minimum of two different analogue voltage levels
- Communicate with the Jetson Nano through compatible protocols
- Consume very low current (Easily powered by the Jetson Nano)
- Provide relatively accurate readings
- Low cost
- High availability

Upon researching for a suitable IC which met or exceeded each of the above requirements, the ADS1115 analogue to digital converter was selected. According to Texas Instruments (2018) the ADS1115 has the following specifications:

- Supply voltage of between 2V to 5.5V
- Up to four analogue inputs
- I²C communication interface (compatible with the Jetson Nano)
- 150 μ A current draw
- 16-bit resolution accuracy (0.0055° per increment)

The ADS1115 integrated circuit is shown in Figure 2.7 below:



Figure 2.8: Adc1115 Integrated Circuit Board

As Figure 2.7 shows, there are 10 pin connections for the ADS1115, which are detailed further in Table 2.4 below:

Table 2.4: ADS1115 Pin Names And Description (Texas Instruments 2018)

Pin Number	Pin Name	Description	Connection
1	VDD	Power input positive	Jetson Nano 3.3V
2	GND	Power input ground	Jetson Nano GND
3	SCL	Serial Clock Line (I ² C communication protocol)	Jetson Nano Pin 5
4	SDA	Serial Data Line (I ² C communication protocol)	Jetson Nano Pin 3
5	ADDR	I ² C slave address select	Not used
6	ALRT	Conversion ready flag	Not used
7	A0	Analog input Channel 0	3382G
8	A1	Analog input Channel 1	3382G
9	A2	Analog input Channel 2	Not used
10	A3	Analog input Channel 3	Not used

2.6 Overall Configuration

With all electrical components of the system finalised, the electrical interconnections can be specified. Figure 2.8 below shows a block diagram of how each component is connected in the mm-wave camera tracking system:

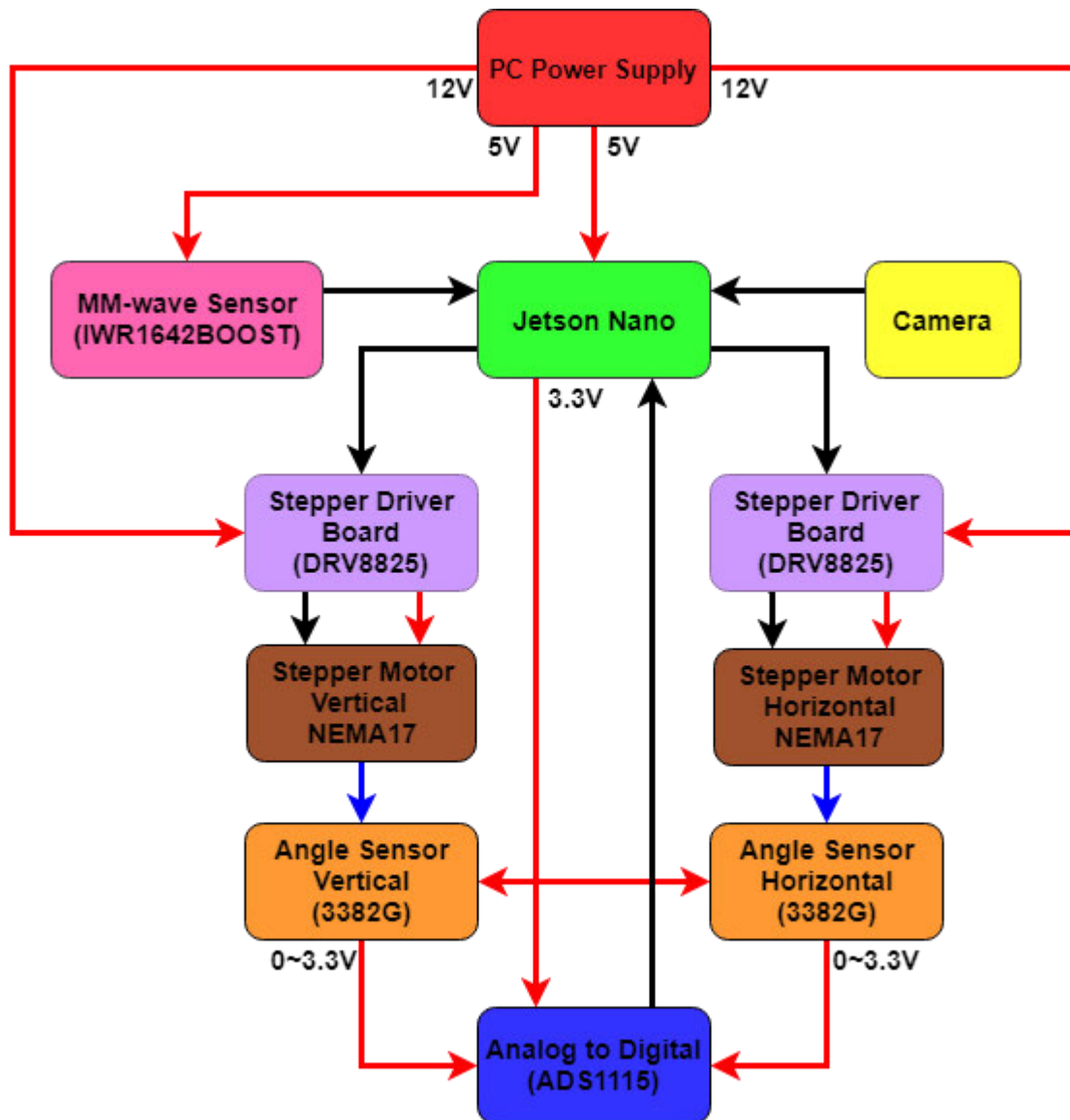
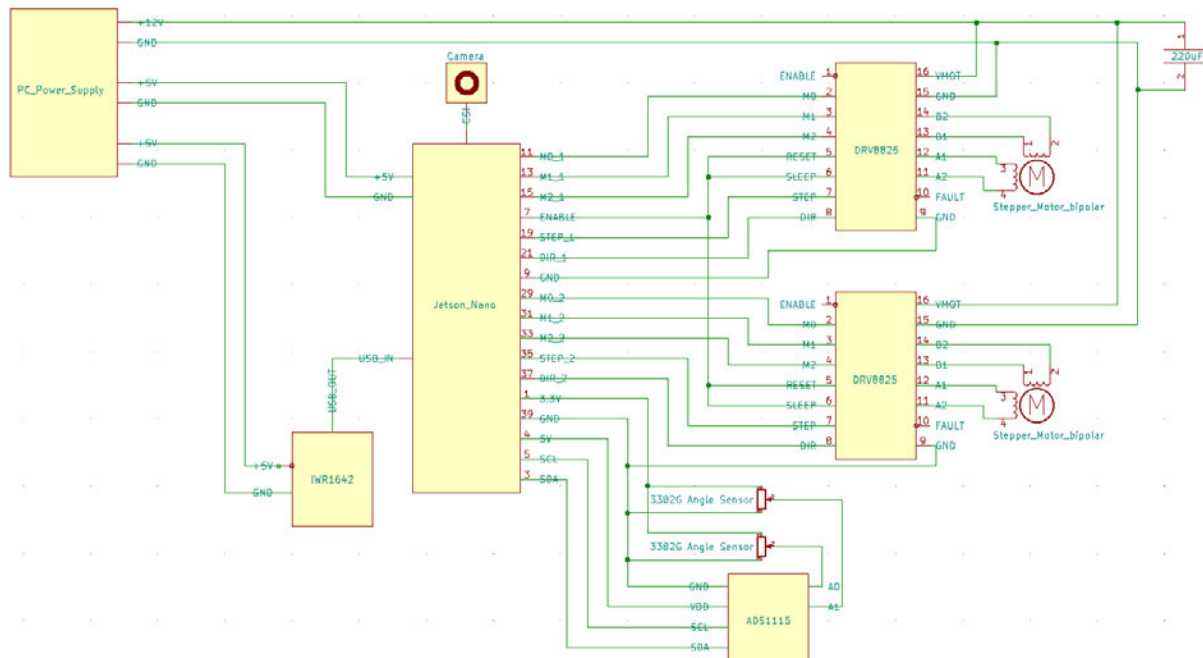


Figure 2.9: Block Diagram Of Mm-Wave Camera Tracking System

Figure 2.8 shows how each component is linked together with arrows indicating the way in which the communication or power is flowing. The black arrows represent the data flow, the blue arrows represent the mechanical orientation, and the red arrows represent how each device

is powered. Further detail on how the electronic components are interconnected is shown in the overall schematic shown in Figure 2.9 below:

Figure 2.10: Overall Schematic Of Mm-Wave Camera Tracking System



With all components and interconnections specified, the next phase of the project is the design and construction of the physical system.

2.6 Chapter 2 Summary

Chapter 2 provided a general overview of the procedures involved in the initial phase of development for the camera tracking system. It begins with the initial setup and testing of the Jetson Nano and learning how to communicate with the camera and operate the on-board IO pins. With the knowledge gained, the major electrical components and circuits were then able to be assembled. The hardware choices as well as the way in which they connected was thoroughly discussed. With the electronics completed, the focus then shifted onto controlling the stepper motors within software. Starting from simple test scripts, the complexity and functionality of the code was incrementally enhanced until the milestone of successfully controlling two stepper motors simultaneously was achieved. It became apparent that the system would need some way of sensing its current pose, so the addition of angle sensors was incorporated into the design. Having finalised the electronic design and developed simple test code successfully, the mm-wave camera tracking system could continue be further developed more easily.

Chapter 3

Mechanical Design

3.1 Chapter 3 Overview

Chapter 3 explores the process of how the mm-wave camera tracing system was built. This phase of development is highly dependent on the design choices which were outlined in [Chapter 2](#). Many of these design choices were made in the conceptualisation phase of designing the hardware. Once a reasonable concept was conceived, the 3D CAD (Computer Aided Design) modelling was started, which helped to further evolve the hardware design. This phase of development ensured that all parts would fit together in a suitable orientation without any problems. Once the parts and assembly were finalised within a CAD environment, the required components could then be manufactured and assembled.

3.2 Conceptualisation

With the appropriate electrical components selected and tested, the next step in development was to design a suitable method to mount all of these components in such a way which would allow the camera to gimbal. After pondering this problem for several days, the idea was conceived to mount a platform to a stepper motor which allowed horizontal motion. On this pivoting platform, the other stepper would be mounted in such a way that would allow vertical steering of the camera and mm-wave sensor. Since the stepper motors are quite heavy for their size, it was decided to arrange them along the same axis to prevent any torsional forces to be introduced into the system. In order to convert rotational motion along the vertical axis (z-axis rotation) to rotational motion along the horizontal axis (x-axis rotation), the idea of using a worm gear drive assembly was conceptualised. This arrangement would allow the camera to pivot up and down if necessary, while simultaneously allowing horizontal steering.

The Jetson Nano, the stepper driver boards and the PC power supply also needed to be arranged in a suitable configuration. These components require several electrical connections between them, so careful consideration was needed to decide on a suitable layout of these components. In order to minimise the overall footprint of the design, it was decided to mount the circuit boards on a base underneath the abovementioned stepper arrangement. This base can then be mounted to the PC power supply at the very bottom of the apparatus.

3.3 CAD Design

With the arrangement of all components conceptualised, the next step was to decide how this assembly may be constructed. 3D printing stood out as the most suitable manufacturing method since the design will require several (often complex). These types of parts are incredibly easy to produce with 3D printing. However before anything can be manufactured in this way, it must first be translated into a CAD assembly. The advantage to this method is that it allows to careful inspection of all components to ensure everything will fit together properly before anything is actually constructed. For this design, it was decided to model the design with the Onshape CAD package, since it is free, it allows simultaneous collaboration and is web-based.

The initial phase in the CAD design process was to re-create the existing components within the Onshape environment. This process involved carefully measuring the physical dimensions of each object and replicating their appearance and size inside of Onshape. Fortunately some components have schematic diagrams available which simplified the measuring process while ensuring the dimensions are exact. The final CAD models of the mm-wave sensor, the camera module, the stepper motor, the stepper driver circuit board and the Jetson Nano are all shown below:

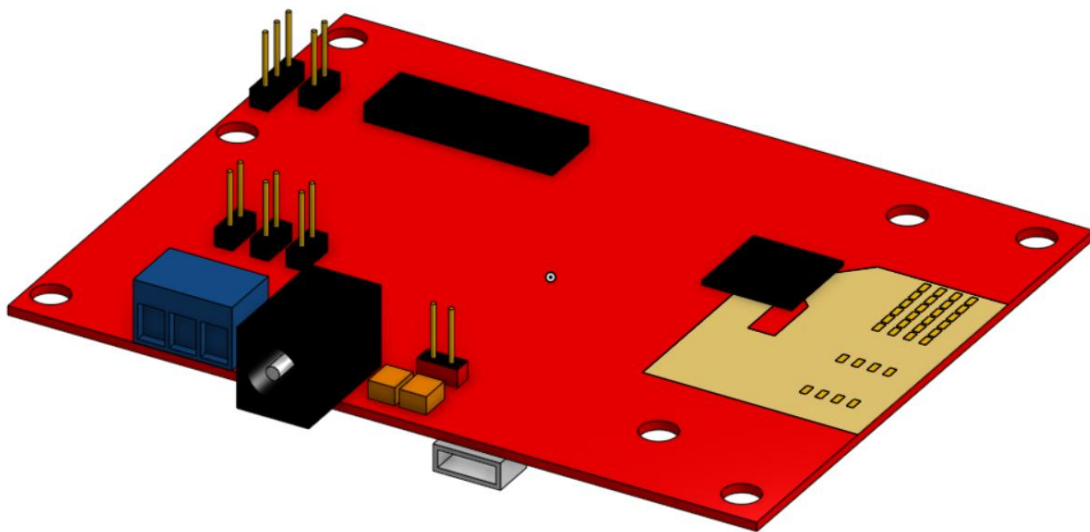


Figure 3.1: CAD Model Of Mm-Wave Sensor (IWR1642BOOST)



Figure 3.2: CAD Model Of Camera Module

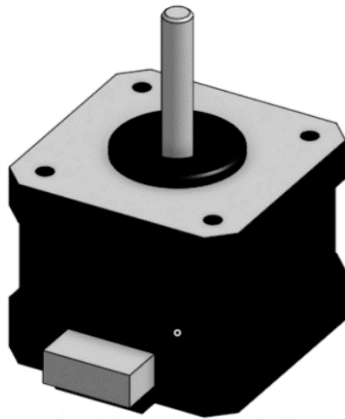


Figure 3.3: CAD Model Of Stepper Motor

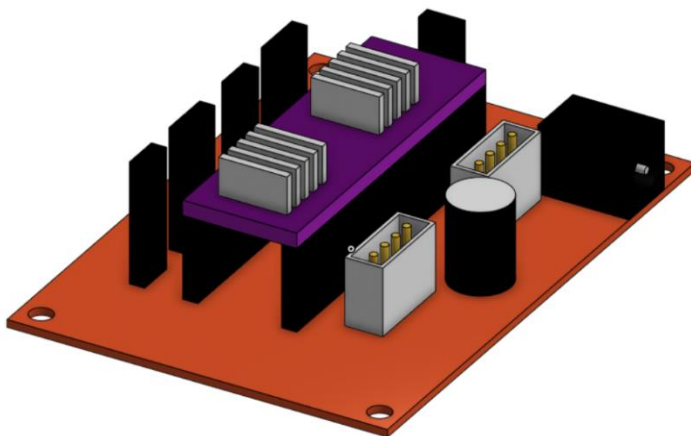


Figure 3.4: CAD Model Of Stepper Driver Circuit Board (DRV8825 \times 2)

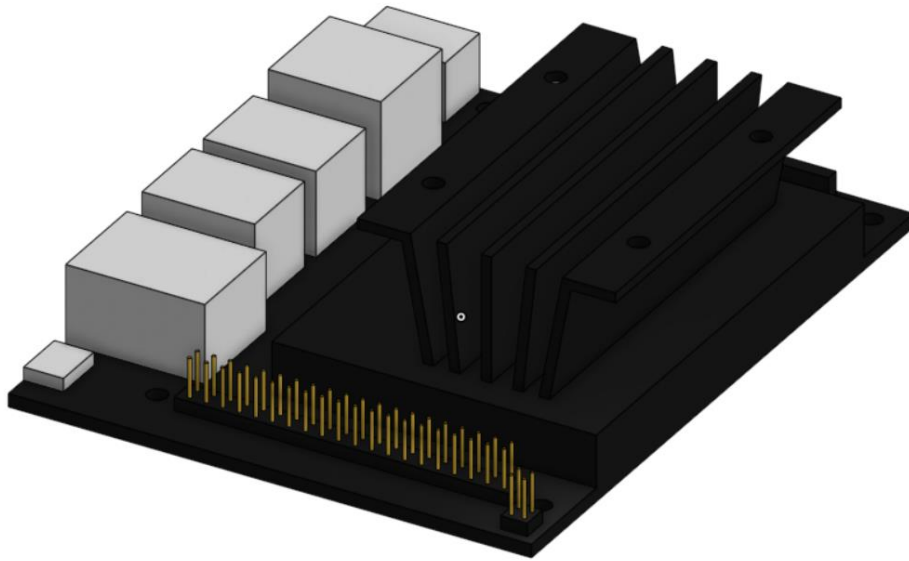


Figure 3.5: CAD Model Of Jetson Nano

The CAD models illustrated above were subsequently used to design the necessary parts for mounting the components in the proper orientation. This process started with modelling a base plate with suitably spaced mounting holes for the Jetson Nano and the driver board circuit board. Using this base plate CAD model as a starting point, the remaining parts could be created. This process often required assembling the already modelled parts within Onshape and measuring certain aspects to ensure each part will fit in its place perfectly. This method was taken a step further when designing the tilt mechanism for the camera. Special attention needed to be taken with the shape of the geared bracket which the mm-wave sensor and the camera are mounted to, since it required a specific range of motion, while ensuring there was no interference with other components. Another important consideration was to ensure that each component is easy to be manufacture by 3D printing. This means that any parts designed should keep any overhanging geometry to a minimum wherever possible. The final consideration while designing the CAD parts is how the parts will be assembled, this means careful thought on the order of assembly the placement of screws and fasteners and cable management. The fully assembled CAD design is shown in Figure 3.6 below:

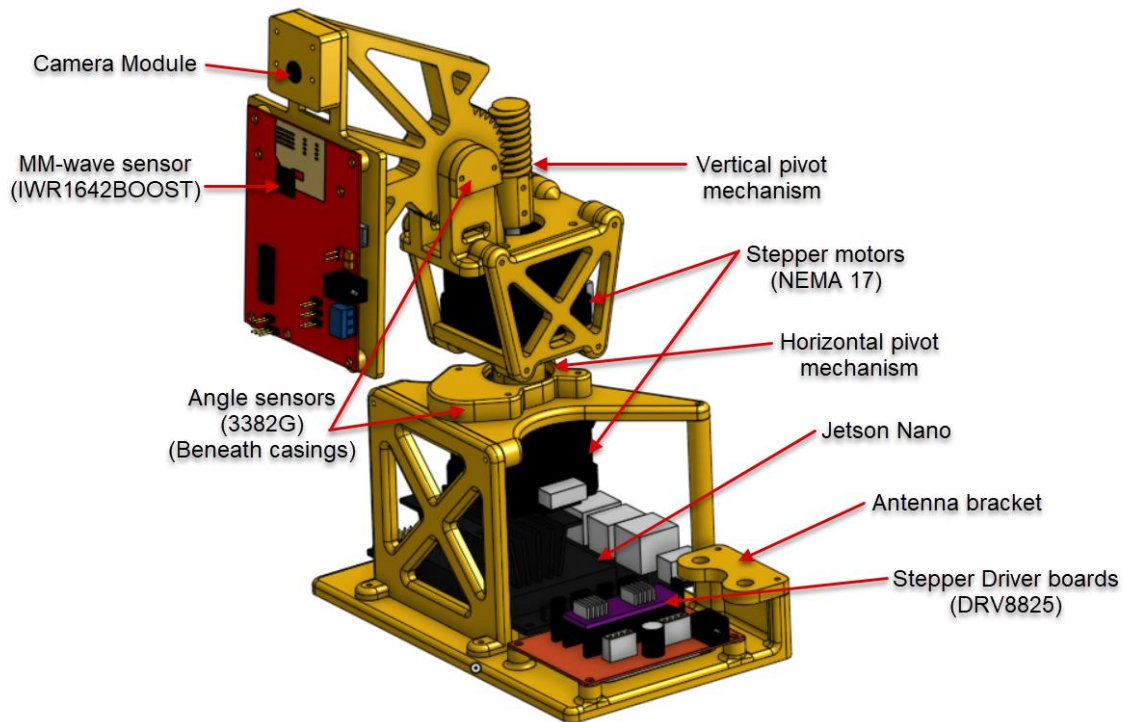


Figure 3.6: CAD Assembly Of Mm-Wave Camera Tracking System

Figure 3.6 shows the fully assembled CAD design of the mm-wave camera tracking system. The components which required 3D printing are coloured yellow. Notice the angle sensors are underneath protective casings. The vertical angle sensor is coupled directly to a specially designed shaft so the vertical angle of the camera can be determined. Direct coupling of the horizontal angle sensor was not possible due to the stepper motor shafts diameter being too large for the angle sensors cavity. To accommodate the horizontal angle sensor, it was coupled with a 1:1 gear that is connected to the horizontal pivot mechanism. Now that all components have been designed, a list of parts can be created, this is shown in Table 3.1 below:

Table 3.1: - Comprehensive List Of Parts To Build Mm-Wave Camera Tracking System

Part Name	Quantity
Electronics	
AD Converter (ADS1115)	1
Angle Sensors (3382G)	2
Camera (Raspberry Pi V2 IMX219)	1
Jetson Nano	1
Mm-Wave Sensor (IWR1642BOOST)	1
PC Power Supply	1
Stepper Drivers (DRV8825)	2
Stepper Motors (NEMA 17)	2
3D Printed	
Antenna Bracket	1
Axil Cover	1
Axil	1
Base	1
Camera Cover	1
Gear Cover	1
Panning Mount	1
Sensor Bracket	1
Sensor Cover	1
Sensor Gear	1
Side Bracket	1
Stepper Plate	1
Tilt Gear	1
Upper Side Bracket	2
Upper Stepper Bracket	1
Worm Gear	1

3.4 Manufacture & Assembly

In order to have the modelled parts 3D printed, the files for each part first needed to be converted to an .STL files. This was done with the finest detail settings available to ensure the parts were created with a high dimensional accuracy. The .STL files were then sent to the 3D printing service that UniSQ has available. After collection of these 3D printed parts, they needed to be ‘cleaned up’. This meant the removal of any additional support structures that were needed for the 3D printing process. This was achieved with the careful use of a sharp knife, long-nose pliers and side-cutters. Some components required further processing like sanding and drilling to accommodate the correct screw sizes. Finally, to attach all the components together it was necessary to purchase several M3 screws at various lengths.

The first step in assembly was to attach the Jetson Nano and the stepper driver circuit board to the base plate, then connect them together electrically. Next, the antennas were attached to the antenna bracket and this assembly was connected to the base plate. From here, the 1st platform plate needed to be assembled before it was attached. To achieve this, an angle sensor had 3 wires soldered to it, then the wires were fed through the small hole in the top of the 1st platform plate. Next, the angle sensor was pushed into place, and the 3D printed gear was inserted into the angle sensors cavity. The gear casing was then placed on top and a stepper motor was placed underneath, ensuing all holes are aligned, which were then secured together with four screws. This assembly was then attached to the base with the addition of a side bracket.

The small stepper platform plate was then attached to the lower stepper motors shaft with a screw to clamp it into position. Another sub-assembly was made by attaching the other stepper motor to the upper-most platform. From here two identical side brackets were attached to the upper-most platform. This sub-assembly was then joined with the main assembly by aligning the holes in the side brackets with the holes in the small stepper platform plate and using screws to secure it. The worm screw was then attached to the upper stepper motors shaft with two small screws. Three more wires were attached to the other angle sensor, then these wires were through the holes in the upper-most platform. Then the angle sensor was secured in position with a small panel and two screws.

The final sub-assembly was made by attaching the sensor plate to the upper geared bracket. Now the camera, camera cover and the mm-wave sensor were secured in place on the sensor plate. The upper geared bracket assembly was then attached to the upper platform of the main assembly with the 3D printed axil and secured in place with another small plate which prevents the axil from sliding out.

At this point the decision was made to secure the whole system to the PC power supply to integrate everything into one assembly. This was achieved by carefully drilling four holes on the corners of the baseplate, while it was placed on top of the PC power supply. This ensured all four holes were aligned correctly. With the holes positioned correctly the baseplate was secured to the PC power supply with M3 spacer screws and M3 screws. In this configuration, there still remained the problem of the PC power supply's fan being on the underside of the whole apparatus, which prevented adequate air-flow for cooling. To resolve this issue, four more longer spacer screws were secured to the underside of the PC power supply to elevate the assembly slightly. Lastly the remaining wires were connected to the proper locations and neatly routed and secured with small zip ties. The assembled mm-wave camera tracking system is shown below in [Figures 3.7 & 3.8](#).

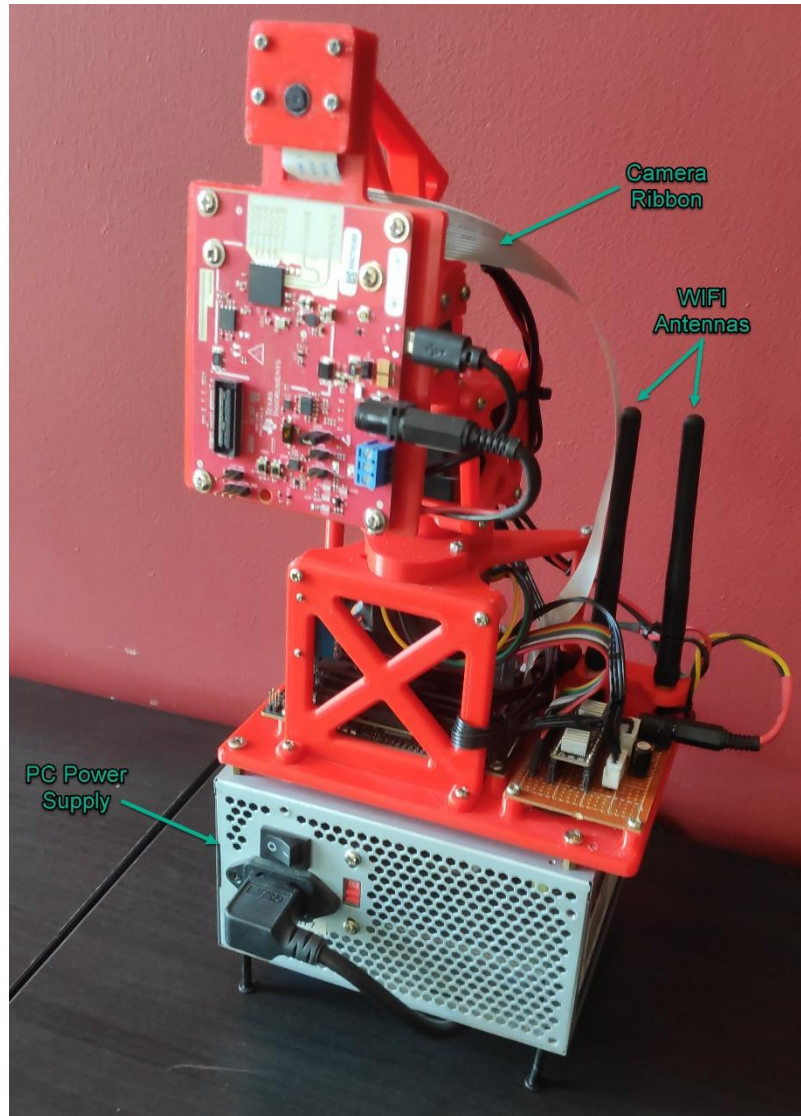


Figure 3.7: Photograph Of Mm-Wave Tracking System (1)

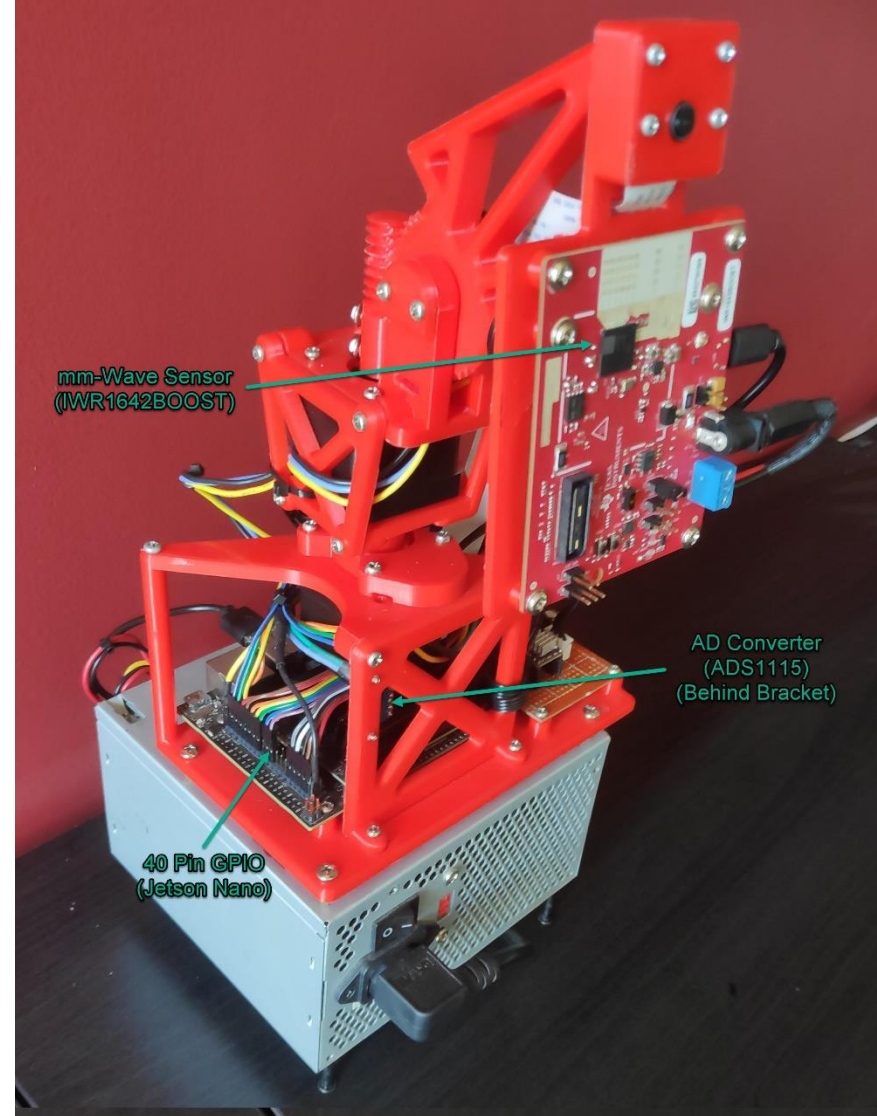


Figure 3.8: Photograph Of Mm-Wave Tracking System (2)

3.5 Chapter 3 Summary

This Chapter explored the process which was taken to design and build the mm-wave camera tracking system. With the main electrical components finalised in the previous section, the task of designing the hardware component became the main focus in this section. The direction this design took was highly dependent on the electrical component selection, initial conceptualisation and the CAD design process. Many parts which were designed relied heavily on the CAD software for specific dimensional and geometric features. Special care was taken to ensure all parts were designed in a way which would not only fit together properly, but able to be easily assembled and 3D printed. Once the parts were 3D printed, they required some minor post processing such as support structure removal, drilling and sanding. Finally these parts could be assembled using the existing electrical components, the 3D printed parts and several M3 screws. Once the system was fully assembled, the wires and cables could be neatly routed and connected with the addition of zip-ties.

Chapter 4

Software Development

4.1 Chapter 4 Overview

At this stage the mm-wave camera tracking system has been built, but the software to operate everything cohesively needs to be developed. During the [system design phase](#), a [python script](#) was created which enabled the manual control of two individual stepper motors, through the manipulation of two slider bars. To integrate this into the current system, the intention is to use the data from the mm-wave sensor to identify an object, and use this objects location as the control parameters for the stepper motors. As the camera moves toward the target, the objects relative spatial coordinates would converge on the central axis', or approach zero. Therefore the control signals for the motors will also converge to zero as this occurs. While this strategy seems plausible, there are still many unknowns and missing pieces to the puzzle to be resolved to bring this concept to fruition.

Perhaps the most pressing concern at this stage is to find a way to connect to and utilise the mm-wave sensor as this up until this point has been unsuccessful. Once the sensor is operating and producing data, the next problem becomes what to do with the data and how can this data be interpreted in a meaningful way. Will it be a straightforward task in extracting and identifying objects from the many data points produced by the mm-wave sensor? If this can be achieved, how will these objects be tracked, and how can one object be selected for the camera to follow? These are all difficult questions which currently do not have answers, however by the end of this chapter, these questions and concerns will be laid to rest.

4.2 Implementation Strategy With ROS

With the prior development of a [python script](#) which can control the stepper motors with a slider bars, the integration of the mm-wave sensor initially seemed to be a trivial task. However when the IWR1642BOOST was connected to the Jetson Nano it became apparent that the communication with this IC would prove to be a much more difficult task. Rather than developing unique code which could interface with the mm-wave sensor, the decision was made to utilise an already existing ROS (Robot Operating System) node which could communicate with the sensor. The idea here was that the mm-wave sensor data could be relayed to the existing motor control script in order to facilitate the control of the stepper motors. With this idea, the primary objective was to convert the existing motor control script into a ROS node, complete with a mechanism for transmitting test data to this node. Once this task is achieved, other components of the system like the camera feed, angle sensor readings and a ‘master’ or ‘command’ control script could be added later.

To implement this strategy several nodes would need to be created which perform the core processes needed to have the system operate as a whole. The required nodes as well as their respective functions are described in Table 4.1 below:

Table 4.1: Required ROS Nodes

Node	Description
Camera	Provide camera feed.
mm-wave	Publish mm-wave sensor data.
Angle Sensors	Publish horizontal and vertical sensor data.
Command	Interpret sensor data and coordinate control of steppers.
Manual Control	GUI to manually control stepper motors.
Horizontal Stepper	Manage the control of the horizontal stepper motor.
Vertical Stepper	Manage the control of the vertical stepper motor.

Table 4.1 give a basic description of what nodes will be required for the mm-wave camera tracking system to function. These nodes also needs to be able to communicate with each other

so all components can operate in a cohesive way. To best illustrate the way this system will intercommunicate, a comprehensive system diagram is shown in Figure 4.1

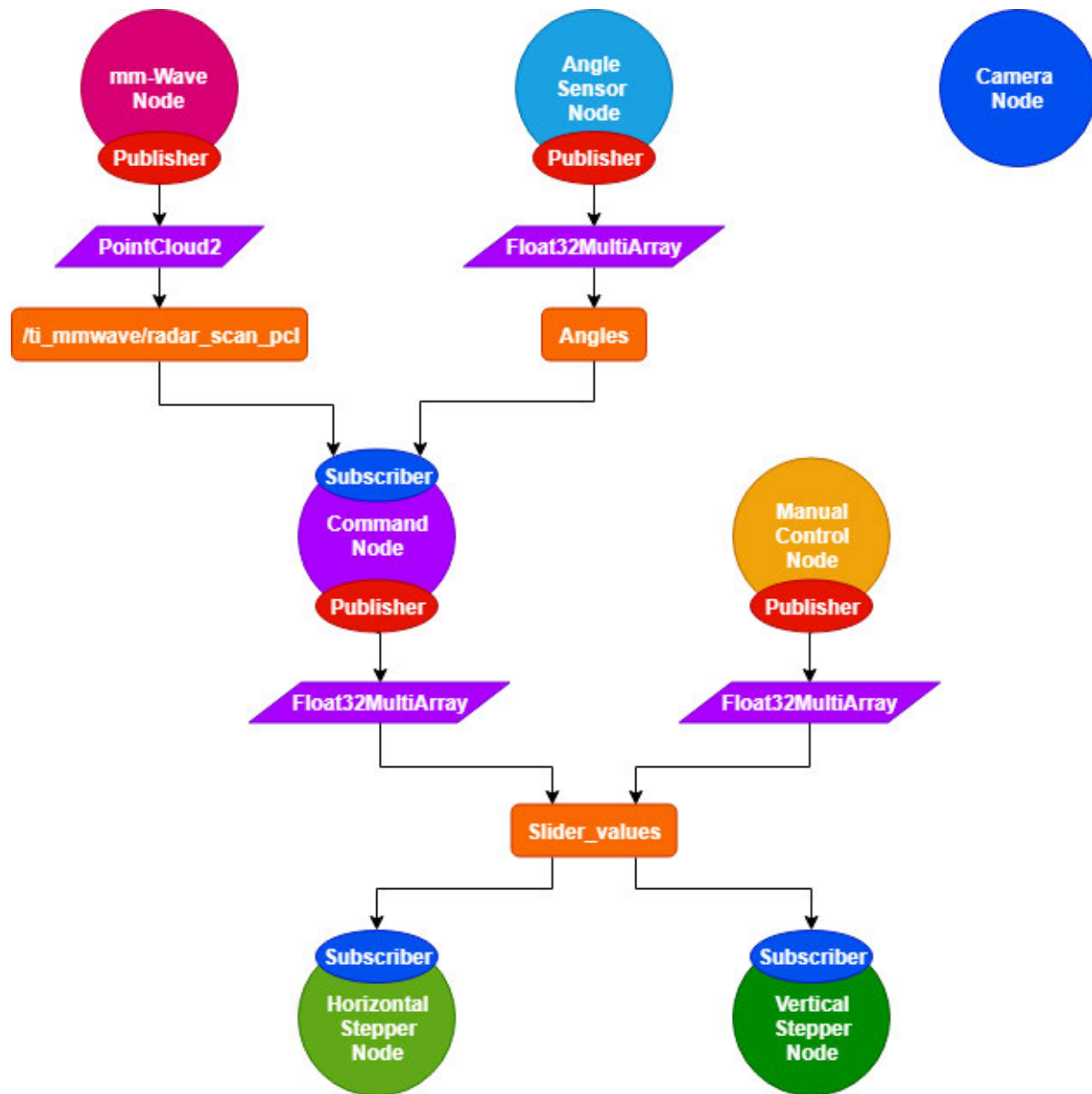


Figure 4.1: ROS System Architecture Of Mm-Wave Camera Tracking System

4.3 ROS Setup

Before any tasks could be done within a ROS system, first ROS needed to be installed, then a ROS environment needed to be set up and finally a ROS package could be created. The mm-wave node which needs to be integrated into the overall system operates on a particular version of ROS called ‘Melodic’ (Zhang 2019). To install this version of ROS on the Jetson Nano, the terminal application was first launched and the following command was executed:

```
sudo apt install ros-melodic-desktop-full
```

Once ROS has been installed, the next step is to provide the system access to the ROS files and features. This step must be done with every terminal window which is used for ROS operations:

```
source /opt/ros/melodic/setup.bash
```

Now a folder needs to be created which holds the files required for a ROS package, this is also known as the ‘ROS workspace’. For this project, the workspace folder was located at ~/Andrew/rosl_mm. The next two commands will create the folders needed for the ROS environment, and then change the working directory to this new folder. When the working directory is in the root folder for the system, the commands are:

```
mkdir -p Andrew/rosl_mm/src
```

```
cd ~/Andrew/rosl_mm/src
```

Now the package can be created, for this project the package name was called ‘mynodes’. The command to create this package is:

```
catkin_create_pkg mynodes std_msgs rospy
```

This command essentially creates all the folders and files needed to run a ROS system. The two phrases ‘std_msgs’ and ‘rospy’ are libraries which are required for the ROS package to run. The ‘std_msgs’ is a library of standard message structure types which are used by ROS and the ‘rospy’ includes the required functions and commands needed to write a functioning ROS node in python. The ‘mynodes’ package will contain all of the ROS nodes which will be developed for the system to operate.

In order to add a new node to the ‘mynodes’ package, it must first be added to this ‘mynodes’ folder, then it must also be linked to the ‘mynodes’ package. To link a new ROS node, some

lines of code need to be added to the 'CMakeLists.txt' and 'package.xml' files which are located in the 'mynodes' folder alongside any nodes that may exist. To add a node to 'CMakeLists.txt' it must be opened with a text editor and the following code needs to be included:

```
Catkin_install_python(  
    PROGRAMS  
        new_node.py  
    DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}  
)
```

If this code structure already exists, simply add the new nodes name between the lines 'PROGRAMS' and 'DESTINATION' in the form shown above. Here 'new_node.py' is the name of the new node which was added to the 'mynodes' folder. This is only for illustration purposes, this node was not actually created or needed for the operation of this system. After linking the new node in the 'CMakeLists.txt' file, it also needs to be linked in the 'package.xml' file. To do this, it needs to be opened in a text editor and the following lines of code need to be added:

```
<export>  
    <node name="new_node" pkg="mynodes" type="new_node.py" />  
</export>
```

Again, if the 'export' tags already exist, simply add a new line like the one show above, but replacing 'new_node' with whatever the name of the newly added node is. This block of code must be placed above the </package> tag which is located at the bottom of the 'package.xml' file. This 'new_node' needs to have execution permissions added so it can be run within a ROS system, which can be achieved with the following command:

```
chmod +x ~/Andrew/ros1_mm/src/mynodes/new_node.py
```

Anytime a new node is added or removed, the package needs to be built in order for it to be run. To build all packages located in the ROS working environment, the following command must be run from the ROS workspace folder (~/.Andrew/ros1_mm):

```
cd ~/.Andrew/ros1_mm
```

```
catkin_make
```

After the package has been build, a node can be run with the following command:

```
Rosrun mynodes new_node.py
```

With the process of installing ROS, creating a package, linking a node, building a package and running a node understood, the next step is to create or convert a script to a ROS node; this will be covered in [Section 4.4](#).

Now the ROS package which interfaces with the mm-wave sensor can also be installed in the same folder where the ‘mynodes’ package is located. To navigate to the correct directory where this mm-wave package needs to be installed the command is:

```
cd ~/Andrew/ros1_mm/src
```

The necessary packages needed to operate the mm-wave sensor can be acquired by running the following commands:

```
git clone https://github.com/radar-lab/ti_mmwave_rospkg.git
```

```
git clone https://github.com/wjwwood/serial.git
```

The ‘ti_mmwave_rospkg’ package interfaces with the mm-wave sensor, however this package also requires the ‘serial’ package to run. With the appropriate packages now in the workspace directory, these packages can be built:

```
cd ~/Andrew/ros1_mm
```

```
catkin_make
```

With all package in the ‘ros1_mm’ now built, the nodes contained within the packages can now be run. There are however a few more commands which need to be run before the mm-wave node can be successfully executed. In fact each time a new terminal is opened, these commands need to be executed before the mm-wave node can run:

```
source devel/setup.bash
```

```
sudo chmod 666 /dev/ttyACM0
```

```
sudo chmod 666 /dev/ttyACM1
```

Now the system is ready to run the mm-wave node, the command to do this is:

```
roslaunch ti_wave_ropkg 1642es2_short_range.launch
```

Usually, this command will not work properly on the first attempt, but simply run the command again and the mm-ros node should run successfully. This will open up a ROS visualizer called ‘RVIZ’ and the point cloud generated by the mm-wave sensor can be viewed, an example of this was shown in [Figure 1.4](#).

Some commands which need to be run each time a new terminal is opened can be added to the `~/.bashrc` file so every new terminal which is opened automatically has access to all files necessary to run ROS and the mm-wave sensor node, however this step is not essential. Another option for simplifying the launching of nodes in ROS can be achieved by creating a launch file. While this step is also not essential, it greatly increases the speed and ease in which a ROS system can be started.

4.4 ROS Node Fundamentals

To write a ROS node, several core ROS functions are used in a particular way to facilitate the communication between other nodes. Most of these functions are imported from the ‘rospy’, ‘std_msgs’ or ‘sensor_msgs’ libraries. One exception to this is the inclusion of a line of code which occupies the very first line, this is called a ‘shebang’. According to Prakash (2021), a shebang ‘is used to specify the interpreter with which the given script will be run by default’. The rest of the required components for a ROS node are mainly associated with the ‘rospy’ library, the most common functionalities include:

- `import rospy` – Makes the functionality for ‘rospy’ accessible to the script.
- `rospy.init_node('new_node')` – This creates a node called ‘new_node’.
- `new_publisher = rospy.Publisher('new_topic', message_type, queue_size=10)` – This will create a publisher called ‘new_publisher’, which can publish messages to the ‘new_topic’ topic. The ‘message_type’ is the type or structure of the messages being published, and the queue size is simply the amount of messages that will be stored in the buffer.
- `new_publisher.publish(new_message)` – This will make the publisher called ‘new_publisher’ send (or publish) the contents of ‘new_message’.
- `rospy.Subscriber('new_topic', message_type, new_function)` – This will enable a subscription to the ‘new_topic’ topic. Each time a message is received, the ‘new_function’ will be called. The message that is received will be passed to this function as the argument.
- `rospy.loginfo("Hello World!")` – This will display (and log) “Hello World!” in the terminal.

Other functionality exists with the ‘rospy’ library, but these are not needed for this project. The other library which is often used for ROS scripts is the ‘std_msgs’ library. This essentially has many standard types of message structure which one may want to use within ROS. It is possible to create custom message types, but the nodes in this project will stick with the standard message types. This library can be used in the following ways:

- `from std_msgs.msg import message_type` – This will import the message type called ‘message_type’ into the ROS script.

- `new_message = message_type()` – This will create a new variable called ‘new_message’ which will take on the structure of ‘message_type’.

Note that ‘message_type’ is not an actual message type which are part of the ROS standard messages, this was done for illustration purposes. This was also done to show how these message types are tied to use of ‘rospy.Publisher’ and ‘rospy.Subscriber’ which were described earlier. The main message type which is used in this project is called ‘Float32MultiArray’, this kind of message allows for multiple float values to be published in one message.

The final library which is used for this project is the ‘sensor_msgs’ library. This contains the messages structures needed to receive messages from the mm-wave sensor node as well as special functions which can be used to process the sensors data. Some example of how this library can be used are:

- `from sensor_msgs.msg import PointCloud2` – This will make the ‘PointCloud2’ message type available for use in the script. This is the type of message that the mm-wave sensor node publishes.
- `import sensor_msgs.point_cloud2 as PC2_function` – This will import the functions which are contained within ‘sensor_msgs.point_cloud2’ and gives these functions a new alias name of ‘PC2_function’. These functions are necessary to manipulate or create messages of the ‘PointCloud2’ type.
- `PC2_function.read_points(cloud_data, field_names=("x", "y", "z"))` – This will read the ‘cloud_data’ point cloud attributes which are found in the field_names argument, here they are x, y, and z. This kind of command is usually run in a ‘for’ loop so each point in the point cloud can be read.

Other functions and message types are includes within the ‘sensor_msgs’ library that can be used to create point cloud data in the correct structure format which can then be published, however, this functionality is not needed for this project.

4.5 Modular ROS Node Creation Strategies

The task of converting the previously written python scripts to ROS nodes initially seemed like a straightforward and easy process, however this was definitely not the case. Some difficulties of this task stem from the unfamiliar coding requirements for a ROS node. But the majority of the complexity arose from ensuring the ROS functionality could be run in unison with the scripts core functionality. Initial attempts at converting the existing code to a ROS node, was only able to run one of these processes at a time, but the simultaneous execution of these section of code was required for the system to operate properly. To overcome these problems it became clear that this system needed to be built with many more smaller, more basic nodes rather than very few more complicated nodes with advanced capabilities. This strategy also aligns more closely to the fundamental design philosophy of how ROS systems are intended to be built. In order to build a functioning ROS system, a node needs to be created for each of the hardware components. To achieve this task, three main strategies were used.

Table 4.2: Development Strategies & Node Origins

Strategy	Description	Nodes
Adoption	Utilise code and scripts which have already been developed to interface with a particular device.	<ul style="list-style-type: none"> • Camera • mm-wave sensor
Break-Down	Start with previously written complex script and break it down into several simpler nodes.	<ul style="list-style-type: none"> • Vertical Stepper • Horizontal Stepper • Motor control publisher
Build-Up	Start with basic functionality and iteratively add layers of complexity.	<ul style="list-style-type: none"> • Angle sensor • Command Node

4.5.3 Adoption Strategy

➤ Camera Code

The first strategy of integration was used to produce code which could operate the camera and the mm-wave sensor. The code which runs the camera was sourced from JetsonHacks (2022) which can be viewed at [this link](#). To have this code operate properly, the following modifications were made:

- Shebang added to start of code: `#!/usr/bin/env python`
- All instances of `flip_method=0` were changed to `flip_method=2`

As mentioned previously, the shebang is used so this script can be used by the ROS system. Changing the ‘flip_method’ to 2 inverts the camera feed image to ensure it displays correctly for this particular system.

➤ Mm-Wave Sensor Node

The code which was needed to operate the mm-wave sensor was briefly discussed in section 4.3, however the full code and more detailed documentation can be found in [this link](#) (Zhang 2019). No modifications were made to this code.

4.5.4 Break-Down Strategy

➤ Manual Motor Control Node

Using the [code](#) which was previously developed to control the stepper motor, three nodes were created. One node was created for each stepper motor (vertical and horizontal) and the other node was created to send control signals from a Graphical User Interface (GUI) to the stepper motors. The first step was to isolate all the code which was used for creating the GUI and generating the values which were sent to the motor to control their speed and direction. This was achieved by copying the existing code and stripping away all parts which were used to control the stepper motors. What remained was a relatively simple script which could generate a popup window that contained two sliding bars, which could generate specific values. From this script, the ROS functionality was added which allowed these slider bar values to be published to a new topic called 'Slider_values'.

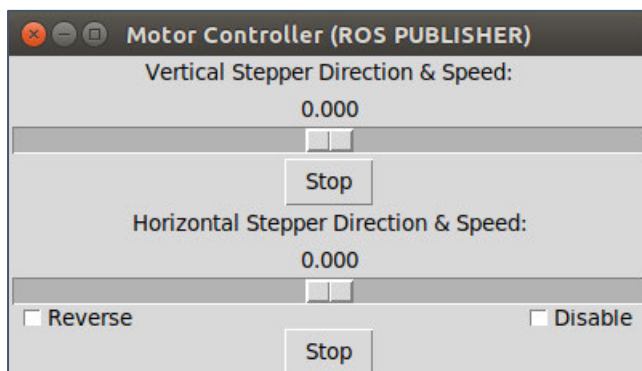


Figure 4.2: GUI For Manual Motor Controller

Some minor additions were added to the GUI to make the motor control easier, these are the two 'Stop' buttons the 'Reverse' checkbox and the 'Disable' checkbox. The stop buttons are used to quickly reset the slider bars to zero, which will instantly stop the corresponding stepper motor from spinning. The reverse checkbox is used to invert the control signals which are sent to the horizontal stepper motor. The idea behind this is the user may be looking at either the camera feed or the apparatus when they are steering the camera, this checkbox allows either option. And finally, the disable checkbox, simple disables the stepper motors all together. Besides the horizontal and vertical motor commands, one extra parameter is also sent. This third parameter is used to determine the source of where the motor control commands are

coming from, for this node, the value is 1. However this third value can also be set to -1 which will completely disable both stepper motors. When the GUI 'disable' checkbox is selected, it causes this third value to become -1. If this is sent, the motors will not respond until a value of 1 is sent; this can only be done by deselecting the 'disable' checkbox in the GUI.

➤ Horizontal and Vertical Stepper Nodes

A similar procedure was used to produce the stepper control nodes. The same script was used as the starting point and all elements related to the GUI were removed. The code was simplified even further by converting it from a 'class' structure to a series of functions. This required any variables which were shared across functions to be declared as 'global' variables. Since the GUI what contained the slider bars has now been removed, another way to receive motor control signals needs to be used. To achieve this, a new ROS subscriber was created which listened to the topic 'Slider_values'. This is where the nodes begin to differ between the vertical and horizontal stepper nodes. For each node type, the 'Slider_value' that is used to control the motor is different. Additionally, the output pins used for the direction, step and mode are also different.

Due to each stepper motor being coupled to the mechanism in different ways, the speed at which the steppers operate must also be different. The vertical stepper turns a worm gear which pivots the camera up and down, which requires several revolutions to change the vertical angle in any significant way. This is why the step mode selected for the vertical stepper is the second fastest mode (mode 2), which is operating in the half step mode. The horizontal stepper is directly linked to the left and right pivot mechanism, so this motor needs to operate in a slower, more controlled step mode. For this reason the finest mode is selected (mode 6), which runs at 1/32 step size. However, it was noticed that this can be inadequate for faster moving targets so a mode switching section was added to the horizontal stepper control node. This mechanism in the code will automatically select faster or slower modes depending on the magnitude of the speed signal sent to this node. This results in the horizontal stepper moving faster when the object is further from the centre axis.

Another addition was made to these node so it could be determined where the motor signals were being sent from. As mentioned previously, a value of 1 corresponds to messages received from the manual control node, so these messages take priority. The way this works is with the use of another variable called ‘override’. As soon as a value of -1 is received, this override value is set to 1, at the same time, the motor speed is set to 0. If a value of 1 is received, then the message is from the manual control node and the motors are enabled, so the override value is set to the speed value that the motor is commanded, this speed is also sent to the motor. Now if motor commands are sent form the ‘Command’ node, they have a value of 0 attached. The stepper motor nodes will only send these motor control signals if the override variable is set to zero. This means that the manual control node must have its speed set to zero (and the disabled unchecked) to allow the ‘Command’ node to take control of a particular stepper.

4.5.5 Build-Up Strategy

➤ [Angle Sensing Node](#)

The built up strategy simply starts from a clean slate, then adds functionality to the script iteratively. This was the strategy used to create the 'Angles' node and the 'Command' node. The 'Angles' node needs to be able to fetch the angle data from the ADS1115 and publish this information so it can be used by the 'Command' node. In order to receive the data that the ADS1115 generates, this node needs to access the data which is sent through the I2C (Inter-Integrated Circuit) communication protocol. This is achieved by setting the appropriate registers by sending a particular bit sequence to the I2C serial bus. This bit sequence controls aspects like: the address for the ADS1115, analogue channel selection, the communication mode, data rate, multiplexing options and more. When this bit sequence is sent, the ADS1115 will convert the analogue signal from the selected channel and send its value to the I2C register. This value needs to be converted to decimal, then scaled to correspond to the correct angle value. In order to accurately scale the received analogue data, the apparatus was moved to a particular angle then the number received from the ADS1115 was recorded. This process was done with both angle sensors at a few significant angles like -90° , -45° , 0° , $+45^\circ$ and $+90^\circ$, and with this data, the linear equations were determined mathematically.

One major issue which was encountered while developing this node was the returned analogue values would occasionally send a 'random' value. Eventually it was determined that these other 'random' values were actually readings from the other three analogue channels. The initial approach to attempt to remedy this issue was to use the 'flag' bit that the ADS1115 is supposed to change after a conversion is made so the program will know when to read the data. However after many attempts to use this method, this strategy was unsuccessful. To fix this issue, a very specific delay time (between setting the register and reading its result) was determined. This allowed the program to read the I2C data at the correct time, before the next value was written to the register. In order to determine the proper delay, the program was set up so a variable delay was used, which slowly increased as the program run. The delay value as well as the analogue value were printed to the console. When the correct value was printed to the console without errors, this delay value was selected. Even with a suitable delay value, about 1 in 500 reading will produce an incorrect reading, however these stray values were filtered out by

comparing an incoming value to the previous value and ignoring values which lie outside the expected range. The final step to completing the 'Angles' node was to add the relevant code which publishes this angle data to the 'Angles' topic.

➤ Command Node

The Command Node is responsible for interpreting the data sent from the mm-wave and angle sensor nodes and sending control signals to the stepper control nodes. To begin the development of this node, the first step was to create a subscriber which receives the data from the mm-wave sensor node. This data is sent through ROS as a 'PointCloud2' message type under the topic called '/ti_mmwave/radar_scan_pcl'. During testing, it was discovered that this point cloud data consists of multiple data points which each has four attributes which are structured like this: `Point_cloud_data = [[x1, y1, z1, i1], [x2, y2, z2, i2], [x3, y3, z3, i3], ...]`. The x, y, and z are the Euclidean coordinates for the data point, and the 'i' is the intensity or strength of that data point. It was observed that the 'z' coordinate is always zero, so it is evident that no vertical tracking will be possible with this sensor.

In order to develop this node, a way to consistently produce a strong radar point was needed. According to Wolff (2007) 'corner reflectors are used to generate a particularly strong radar echo'. So in order to achieve constantly strong radar point, a corner reflector was constructed, which is shown in Figure 4.2 below. This corner reflector was constructed with 0.8mm thick aluminium sheet metal, and held together with hot glue. When the corner reflector was tested, it generated a very strong radar data point as expected. With further testing it was determined that under normal conditions, most point cloud data points will not exceed an intensity level of around 35, however when the corner reflector was introduced, the reflection intensity was easily above 35. This observation was used to isolate the most intense point in the point cloud data. While this technique is very effective to isolate one particular point, it is not ideal for a person to carry around a corner reflector so the camera can follow their movements, therefore additional filtering strategies were used.

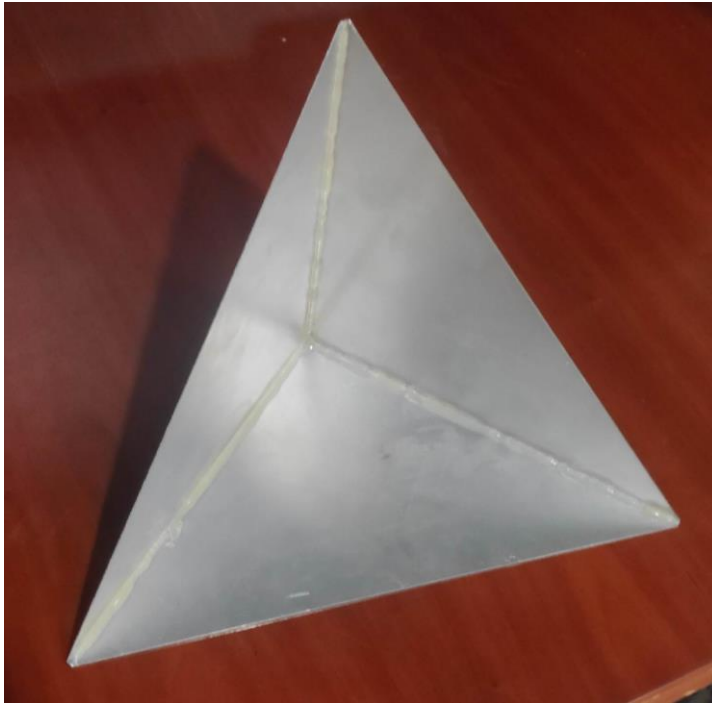


Figure 4.3: Corner Reflector

To achieve this type of tracking, a simple, yet effective strategy was used. When the point cloud data points are received, it undergoes multiple filtering strategies. First, any points which have a greater intensity than 35 are added to the 'filterered_points' array. The next part relies on the existence of a previously extracted point, which the camera tries to follow, this is called 'followed_point'. If this point contains no data, then the filtering algorithm will only keep data points which are between 15cm to 100cm away from the sensor and within 50cm to the left or right of the centre axis. If 'followed_point' does contain data, then the filtering algorithm will only keep data points that are within 20cm closer or further away and within 30cm to the left or right of where the 'followed_point' was detected. This effectively ignores all other data points (except the corner reflector) and tries to find the same object from the last program cycle. Once all of these points have been collected into the 'filterered_points' array, the most intense point from this is selected to be the target.

The greatest advantage to using this technique is the ability to use the corner reflector to get the system to lock on to a target, then even when the reflector is hidden, the system follows the person quite reliably. This is essentially a method for an object to be selected for tracking. Other more complex object tracking algorithms were explored, but due to the numerous

complexities and time constrains this simpler, yet still effective solution was developed. The following section discusses some of the techniques which were explored as well as the source of the added complexity.

Once the data point is selected, its 'x' coordinate relates to how far from the sensor the object is, and the 'y' value corresponds to how far to the left or right the object is from the central axis. This 'y' value is of particular interest to the way this systems control mechanism works. The magnitude and sign of the 'y' value is proportional to the desired rotational speed and direction that the horizontal stepper motor needs to operate successfully to track the object.

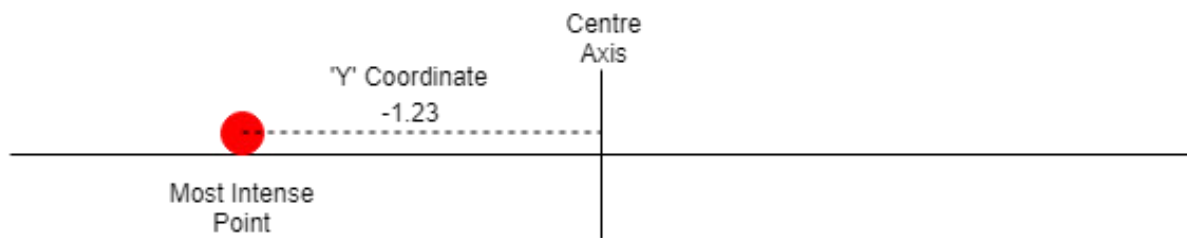


Figure 4.4: Importance Of Object 'Y' Coordinate For Motor Control

The motion caused by the horizontal stepper motor will bring the centre axis closer to the detected object, which in turn will reduce the magnitude of the 'y' value. When the object is aligned with the centre axis, the 'y' value will be zero, which will stop the stepper motor from spinning. It is rather obvious that the sign of 'y' will correspond to the spin direction of the stepper motor.

While it is possible to simply use the 'y' value directly for the horizontal motors direction and speed, a few issue would arise from this, hence the use of a few signal processing techniques. To increase the tracking speed, the 'y' value was multiplied by 20. The next concern was the stepper moving too fast so it was limited to a maximum of 5, this was handled with a simple 'if' statement. Another problem which was encountered caused another object to become the most intense data point, which would cause the motor to violently turn toward the new object. To handle this problem, the speed from the previous program cycle was used, and if large

difference in speed was detected, then only a small change was made to the speed in the direction of the speed change. This acted as a way to slow down the acceleration of the stepper speed.

The last issue was the camera moving too far to the left or right, causing the wires to become too tight. For this reason the sensed angle for the horizontal mechanism was used to limit the speed. To acquire the angle values, a ROS subscriber was used to receive the angle data which is sent through the 'Angles' topic. If the angle of the horizontal stepper exceeded 60° and the speed is set in the direction which would increase the angle, the speed was set to zero. This way if the targeted object moves outside the mechanisms range of motion, the camera will remain as far to the side as possible. Usually this would occur if another object became the focus and the camera would move to one side. For this reason, the 'followed_point' is also cleared if the camera moves outside the $\pm 60^\circ$ range, this would cause the algorithm to look for either the corner reflector or the most intense point right near the front of the sensor. When the detected object is within $\pm 60^\circ$ from the central axis, the camera will again move toward the target. The final step in this node is to publish the calculated speed value, along with the identifier of 0 on the message topic 'Slider_values' so the motor control nodes can receive this data.

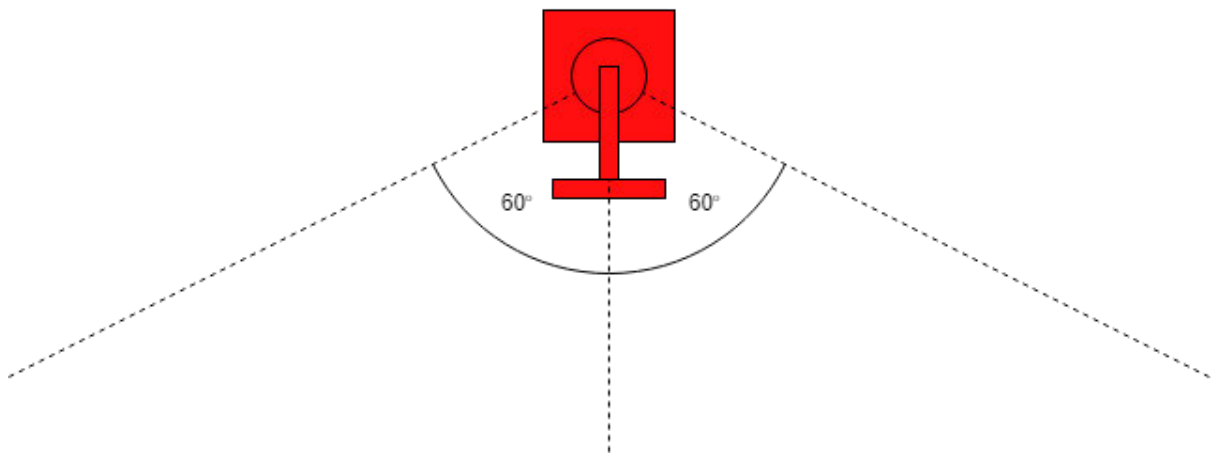


Figure 4.5: Mm-Wave Camera Tracing System Horizontal Range Of Motion

4.6 Alternate Development Strategies

During the process of creating this project many issues were encountered which significantly slowed the progress of developing this project. Besides the problems with software integration, interfacing with the mm-wave sensor and angle sensing discrepancies, other issues were present which could not be overcome within the time remaining for the project. Many more features were intended to be added to the 'Command' node which would increase the overall functionality of the mm-wave tracking system. The system in its current form will still track a human object well, but the method of object selection is not the most elegant solution. Also, the tracked point will occasionally get lost. The most likely reason for this is that the most intense radar point is followed, and often another object will generate a more intense radar reflection than the specified threshold, hence distracting the system. To address this issue, an alternate solution was explored and while some of these features were tested successfully, the complete set of features needed for this other approach to work properly could not be implemented effectively without much more development time.

The alternate strategy was to cluster several stronger points together to isolate a handful of objects. While this part was easily achievable, this method opens up quite a number of other challenges which proved to increase the difficulty substantially. The first obvious challenge was creating a mechanism to selecting a particular object to track. This requires some kind of user interface which could display a list of isolated clusters which some mechanism of selecting a particular point. Attempts were made to use tkinter with a series of checkboxes which corresponded to each point. While this was achieved with limited success, another more important issue became more evident.

These clustered points needed a way to be tracked and identified. In each cycle of the command nodes program, these point clusters could be calculated, but then a method of linking the currently detected objects to the previously detected objects needed to be devised. The most obvious solution was to compare the locations of each point in the current cycle with the points detected in the previous cycle. If the location of a point was close to a previously detected point, then it was assumed to be the same point. While an algorithm of this type was a

challenge, some success was achieved here. This also required a way of storing a list of detected objects along with an identifying number so the points can be tracked.

The final issue which ultimately thwarted this approach was the objects disappearing and reappearing in scan cycles. When this happens, duplicate entries were made in the stored points array, and the identifying numbers would become messed up. Again, some success was achieved with this, but in a separate more simplified python script, but this could not be effectively incorporated into the main 'Command' node script. Since this method required so many other complex processes and the ever-increasing complexity made the script more and more unmanageable, the decision was made to simplify the solution significantly. Using the basic strategy of following the most intense radar point, other effective tracking methods were identified and ultimately implemented into the final camera tracking system.

4.7 Chapter 4 Summary

Chapter 4 focussed on the development and integration of the software needed to control the camera tracking system. From the beginning of this section, a major focus was determining a successful method for interfacing with the IWR1642BOOST sensor. It was discovered that software which can achieve this task has already been developed, but it uses the ROS protocol for intercommunication. This revelation caused a shift in focus with the software integration strategy to convert all existing code to ROS nodes, and add other ROS nodes if required. During this process, it was necessary to understand the way a ROS system works as well as how to set up and use them. With this understanding, several methods of node creation were used to create a functioning ROS system; these were the Adoption, Break-down and Build-up strategies. The creation process for each node was also discussed. Once most nodes were functional, the focus shifted to the development of the ‘Command’ node, which is responsible for the overall control on the system. Numerous significant challenges became evident during the development of the ‘Command’ node, which adjusted the object tracking strategy to a simpler, yet very effective solution. The technique that the Command node used to facilitate the camera following a target object were explained. Finally, the alternate tracking strategy which faced problems was explained, and the sources of the added complications were identified.

Chapter 5

Testing & Conclusions

5.1 Chapter 5 Overview

Now that the system is operating successfully, all that is left to do is to test out its capabilities. This section discusses the processes used to test certain attributes of the system like its tracking performance as well as its resistance to disturbances. These tests will finally answer the question to how well the mm-wave sensor can cope with poor lighting and external distractions. The test results will then be analysed and discussed so potential areas of improvement can be identified as well as where the system excels. Finally, several conclusions are drawn from evaluating the project from a holistic perspective.

5.2 System Testing

In order to ascertain the effectiveness of this system, several attributes relating to how the system operates need to be tested. The original plan was to compare the performance of this system to an existing camera tracking system, however this could not be achieved due to the budget for the project being exceeded in other areas. For this reason, the testing strategy has been modified to focus on collecting system specifications and performance characteristics. These camera tracking characteristics can be divided into two broad categories: Tracking performance and resistance to disturbances. The main attributes which can be tested are shown in Table 5.1:

Table 5.1: Camera Tracking Attributes For Testing

Tracking Performance	Explanation
Maximum tracking scope	How far to the left or right can the object be tracked? How much vertical motion can be accounted for in the system?
Maximum tracking range	How far from the sensor is an object able to be reliably tracked?
Maximum tracking speed	How fast can an object move before tracking is lost?
Maximum angular velocity	How fast does the camera tilt in the horizontal and vertical axis?
Tracking smoothness	How smooth are the motions of the camera?
Resistance to disturbances	
Lighting conditions	Does poor lighting conditions affect the performance of the tracking?
Fast movements	Will the camera lose track of an object if it moves too quickly?
Distractions	How well does the camera stay focussed on one particular target when other targets are moving nearby?

5.3 Tracking Performance

To gain an understanding of the tracking performance of the system, 5 key areas have been identified for investigation: scope, range, speed, angular velocity, and smoothness. While some attributes are easily measured, others are more subjective, like smoothness.

Maximum Tracking Scope

The maximum tracking scope is solely related to how far the camera can look in any direction. For the horizontal range of motion, this has been limited in software to $\pm 60^\circ$, since the wires and cables become too tightly bound around the pivot point. Even with this software limit in place, the system has been observed moving as far as 70° , when the tracked object moves quickly out of range. The vertical range of motion is $\pm 45^\circ$, which was intentionally limited by the hardware design. This was done for the lower vertical angles to prevent the system interfering with itself, and the upper range was set to $+45^\circ$ because higher ranges of motion are not expected to be needed for a camera tracking system of this type.

Horizontal Scope: 120°

Vertical Scope: 90°

Maximum Tracking Range

Before the maximum range test is undertaken, it must be noted that the mm-wave sensor node has a short range mode and a long range mode, however the long range mode has errors when attempted to be executed. Therefore for all tests (including this maximum range test) the short range mode was used. In order to determine how far away a target can be tracked by the mm-wave camera tracking system, the apparatus was set up outdoors. The apparatus was set up on a small table near a large clearing of land. One end of a 30m tape measure was placed in line with the mm-wave sensor but under the table; this was secured in place with a stone block. The other end of the tape measure was extended out to the back fence, which was about 20m away. The set-up of this apparatus is shown in Figure 5.1 below:



Figure 5.1: Maximum Tracking Range Test

With the system now set up in a suitable location, it was activated and quickly found the target. The method used was to slowly walk backward in a zig-zag motion and take notice of the distance that the system stops following the target. This yielded a result of about 9m from the sensor, however the system struggles to retain a lock on the target at this range. When the target moves to a range of 8m, the system has no problems tracking and retaining a stable target lock. It was also noticed, that the corner reflector did not seem to improve the performance of the system during this test.

Total Maximum Range: 9m

Maximum Stable Range: 8m

Maximum Tracking Speed

With the apparatus still set-up outdoors, the maximum tracking speed could also be determined. For this test the tape measure was laid out along the ground at a distance of 8m from the sensor. The start of the tape measure was once again held in place with the stone block near the edge

of the system field of view, and the other end was placed near the other field of view limit of the system. The measurement of where the tape measure was placed was exactly 15m. Another marker (the corner reflector) was placed at exactly 1m on the tape measure, which identifies the starting point. Using this 14m distance as well as a stopwatch, the speed of the tracked target can be determined. The intention is to determine the maximum speed that the system can track a human target.

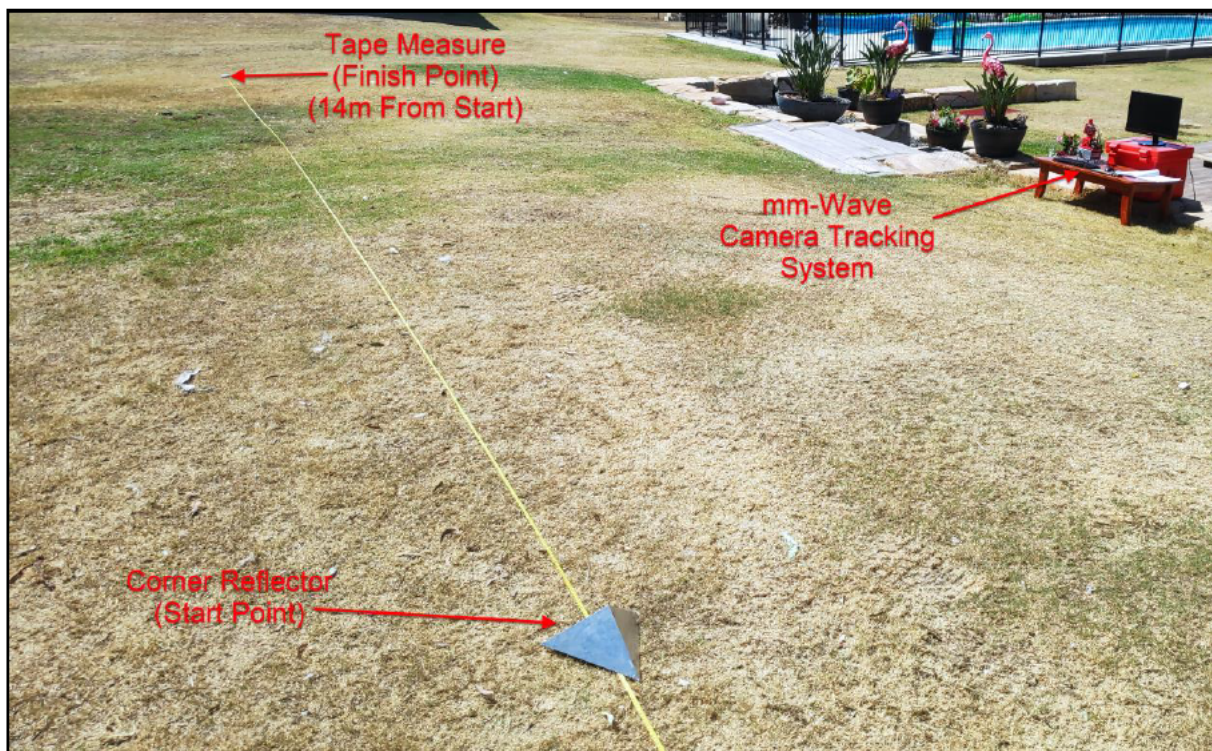


Figure 5.2: Maximum Tracking Speed Test

With the system operating and a stable lock was established, the target moved to the start of the tape measure. With the stopwatch in hand, the target ran the length of the tape measure, taking care to start and stop the stopwatch at the appropriate marker locations. While undertaking this test, the camera tracking system was monitored to ascertain whether the system could keep up and maintain a lock on the target. The first test was completed with the target running as fast as possible, and the system was able to keep up to the target and did not lose the target lock. This was completed three times, with each run having its time recorded. While it may be plausible the system could track a person moving faster, it is unlikely that tracking speeds higher than this are necessary. Therefore the results found by these tests will be

taken to be the maximum possible tracking speed. The results for all three runs of the speed test is shown in Table 5.2 Below:

Table 5.2: Maximum Tracking Speed Test Results

	Speed Test (14m)	
Unit	seconds	m/sec
1st Test	2.74	5.11
2nd Test	2.85	4.91
3rd Test	2.52	5.56

Maximum Tracking Speed: 5.56 m/s

Maximum Angular Velocity

To test the Maximum angular velocity, nine tests will be conducted, which will be three tests done three times each. The first test will determine the maximum angular velocity for the vertical motion of the camera, and the other two tests will examine the horizontal motion. The difference between the horizontal motion tests are one test will be using the manual control window, and the other test will track an object from left to right, as quickly as possible. The vertical motion test is the most straight forward, the camera will be aimed as far down as possible (-45°), then a stopwatch will be started and simultaneously the vertical speed will be set to maximum. Once the camera reaches the maximum vertical angle ($+45^\circ$), the stop watch will be stopped. This will be done three times to attain an average value. The horizontal angular speed test will be similar, except the angle will have to be monitored in the console, since the manual control mode does not have any angular limitations. Finally the last horizontal test will use the corner reflector so tracking performs at its best. The reflector will be moved to the left of the camera, so it goes out of range, then the corner reflector will be moved quickly to the other side, but slow enough to ensure tracking is still happening. The stopwatch will be started when the camera begins to move and stopped when the motion has ceased. The results for each of these tests is shown in Table 5.3 below:

Table 5.3: Maximum Angular Velocity Test Results:

	Vertical (Manual)		Horizontal (Manual)		Horizontal (Automatic)	
Range	90°		120°		120°	
Unit	seconds	deg/sec	seconds	deg/sec	seconds	deg/sec
1st Test	19.34	4.65	2.66	45.11	6.67	17.99
2nd Test	18.58	4.84	2.18	55.05	6.4	18.75
3rd Test	20.15	4.47	2.81	42.7	6.6	18.18
Average	19.35	4.65	2.55	47.06	6.56	18.3

The results from Table 5.3 show that the angular velocity of the vertical motion is by far the slowest at 4.65 deg/s. This result is expected since the coupling mechanism used is a worm gear which dramatically reduces the spin ratio and speed. The fastest speed by far is the Horizontal tracking in manual mode at 47.06 deg/s, this is because the speeds that can be generated by the motor controller GUI are limited to a maximum speed of 10. Conversely, the automatic horizontal tracking is limited to a speed of 5, to allow for smoother tracking, and avoid losing the object being tracked.

Maximum Angular Velocity Vertical (Manual): 4.65 deg/s

Maximum Angular Velocity Horizontal (Manual): 47.06 deg/s

Maximum Angular Velocity Horizontal (Automatic): 18.3 deg/s

Tracking Smoothness

While this parameter is highly subjective, it is possible to make some observations to describe how well the camera moves. When the manual control is used to move the camera up and down, the motion is noticeably jagged. This may be caused by a number of different factors such as the small amount of play in the meshing of the worm gear and the geared bracket. As the motor spins, it may be causing the geared bracket to move in small increments instead of smoother motions. Another factor may be due to the vertical stepper running at a faster speed in which a larger step size is used. It is also suspected that as the stepper motor speed increases, for steps are missed, which is causing the jagged motions. The vertical motion smoothness can be described as noticeably jagged.

When considering the motion of the horizontal motion, several other observations are made. If the tracking speed is very slow, the motions are very smooth. However when faster motions are observed, the motion becomes more jagged. This issue is most likely due to the change in step sizes needed for the faster speeds. While this jagged motion is noticeable when evaluating the motion smoothness, it is not as obvious as in the vertical motions. It has also been observed that when an object is being automatically tracked, there seems to be some minor lag between an objects movement and the cameras motion. This is less pronounced when the radar reflector is used, but this become more pronounced that faster the object moves around. While the lag in the camera tracker is evident, it is still quite responsive and by no means would make this system unusable. To summarise the smoothness of the horizontal motions, it can be described as reasonably smooth, with a slight lag.

Vertical Smoothness: Noticeably Jagged

Horizontal Smoothness: Reasonably Smooth, With Slight Lag

5.4 Resistance To Disturbances

One of the most touted attributes of mm-wave sensors is their ability to operate under very poor lighting conditions, since it does not rely on visual methods. One particular pitfall of some visual based camera tracking system is that they can be easily distracted by other objects moving in the field of view. Due to the techniques used in tracking with the mm-wave sensor, this issue should be improved. However while this system is expected to have improved resistance to disturbances in some areas, the expected performance for tracking fast moving objects is not expected to offer any significant improvement.

Lighting Conditions

To test the resilience of the system to poor lighting conditions, the system was tested under three different conditions. These conditions are normal good lighting, very low lighting, and extreme high lighting (glare). The system was first tested under normal lighting conditions, with and without the corner reflector, and the tracking characteristics were identified. It was noted that fast motions may cause the tracking to become lost, but this can easily be regained by showing the corner reflector so the system can ‘lock on’ to the person once again. When the corner reflector is used, the tracking is very consistent and quite responsive.

When all lights are switched off at night (including the computer monitors), the room is almost completely dark. The only sources of light are the small LED’s on the Jetson Nano and the IWR1642BOOST. These LED’s on the mm-wave sensor were useful for determining how well the system was able to track the target. When the system is tested under these conditions, there seemed to be no difference in the tracking performance. Even though this result was expected, it is still very interesting to see that it is still able to track objects under such poor lighting conditions.

The final test to determine the systems resistance to difficult lighting conditions was the high glare test. In order to introduce strong glare, a 12000lm flashlight was directed straight at the sensor while it was in operation. When this was done, the camera feed on the screen showed a completely white screen what at close range, and a strong bright circle with a black background when at an increased distance. With the camera showing this, it is assumed that a standard camera tracking system would be completely useless under these lighting conditions. Despite

what the camera was showing, the system was still able to track the target objects with exactly the same performance as the normal lighting conditions.

Normal Lighting Condition: Adequate tracking without reflector, Great tracking with reflector.

Darkness Condition: No change to performance characteristics.

High Glare Condition: No change to performance characteristics.

Fast Movements

With the mm-wave sensor showing improved performance under poor lighting conditions, one area where no significant improvement is expected is with tracking fast moving objects. To test the systems capability with tracking fast objects, two modes of testing were undertaken. Both tests will have the system attempt to track a person moving quickly from one side of the room to the other side, however one test the target will have the corner reflector, and the other test is un-aided. When the test was performed without the corner reflector, the system could not keep up, and subsequently lost its 'lock' with the object, the object could be found once again by using the corner reflector. When the test was performed while holding the corner reflector, the system did seem to move more quickly to track the object, however it could not keep up with the object, and the target left the camera frame. However despite this, the camera did catch up with the target and the 'lock' was not lost.

These observations in the tracking performance can be attributed to a number of key factors. The most significant factor is likely the way the command node handles the sensor data. Several trade-offs were made with selecting parameters such as motor speed, search area, intensity thresholds etc. The problem is that while these parameters are designed to achieve certain performance benefits, they can also introduce other undesirable effects. One such example was the decision to make the search area smaller, so other targets were not selected. This has the undesirable effect of reducing the maximum speed at which the tracked object can move at. However, even with a command node that is written perfectly, the limiting factor becomes the refresh rate of the mm-wave sensor, which is 30Hz. Modern cameras can easily exceed this refresh rate by a significantly large margin, which means that advanced cameras are capable of tracking objects that are moving at much faster speeds, the limiting factor here becomes the processing power requirements.

Fast Target Without Corner Reflector: Tracking lost

Fast Target With Corner Reflector: System couldn't keep up, but tracking was retained

Distractions

One significant problem with any camera tracking system is their susceptibility to being distracted by other people or objects in the field of view. This generally stems from the methods used to control the tracking of the target. Cameras can only detect objects in two dimensions, so depth information cannot be used to track objects. However to counteract this, facial recognition can be used so only one person is followed, but not all camera tracking systems use this feature. There are circumstances where it is desirable for the camera to have the ability to track any person, but this introduces the problem of the system being easily distracted and cameras are not well suited to counteract this. With the mm-wave sensor, the objects which are detected not only horizontal spatial information, but depth is also captured. This means that objects which lie at varying distances from the sensor can be accurately detected and measured. By taking advantage of this property of mm-wave sensors, improved distraction resistance can be achieved.

To determine the distraction resistance of this system, a simple test will be performed. Once the camera tracking system has locked on to a target person, other people are introduced into the room and walk in front of and behind the person who is being tracked. When the system was started, it detected and tracked the target person. This target person moved backward to about 3m distance from the sensor. This person stood still and two others walking into the room and began moving around randomly. The camera tracking system did not lose its focus on the target person. Even when the other two people tried to become the tracked target, this was almost impossible for them to achieve. These tests were undertaken without the use of the corner reflector.

While the system performs extremely well with resisting distractions from other people, it does not accomplish this level of tracking proficiency with all kinds of objects. During testing of this system, it has been frequently observed that the system will become distracted by other metallic objects in the room. These other metallic objects are creating strong radar reflections, and the suspicion is that occasionally one of these objects returns a radar reflection stronger than the intensity threshold of 35. This would cause the system to mistake the other object as

the corner reflector, and hence will set this object to become the new target to be tracked. While it would be possible to improve this issue with alternate filtering strategies, these methods were not pursued due to time limitations. Despite this fixable issue, it is clear that the mm-wave sensors bring a clear advantage to distraction resistance in a camera tracking systems.

Distractions from other people: Excellent performance.

Distractions from high radar reflectiveness: Adequate performance, improvements possible.

5.5 Conclusions

5.5.1 Test Conclusions

During the process of testing the system, several other conclusions were reached. In the first test of determining the maximum scope of motion of the system, it was concluded that the system has a horizontal range of motion of 120° , and the vertical range of motion is 90° .

While this range of motion is regarded to be adequate for this system, this could be further improved if the mounting brackets and cable management were redesigned.

Outdoor testing determined that maximum stable range for tracking a person is at about 8m from the sensor. For indoor tracking of people, this range would generally be adequate, however increased distances would be a welcome improvement in the system. As mentioned earlier, the mm-wave sensor node does have a long range mode, but this was not able to be run. Perhaps, the developer of this node could be contacted to troubleshoot this issue to have this system operate at further distances. According to Texas Instruments (2020), the mm-wave sensor has a maximum operating range of up to 30m, which would be more than adequate for most person tracking applications. It was also determined that the maximum tracking speed of the system is above 5.56 m/s, which is easily fast enough to track a person under normal circumstances.

This test was completed at close to the maximum range, so the angular rotational velocity of the camera was not the limiting factor in this test, however at closer ranges this is likely to become a problem. This is because the maximum angular velocity of automatic tracking in the horizontal plane is 18.3 deg/s, which may struggle to keep up to close, fast moving objects. This performance could be improved by changing the maximum speed of the horizontal stepper, but this would decrease the smoothness of the tracking. It was also determined that the maximum vertical angular velocity is 4.65 deg/s, which is very slow. Since the camera tracking system cannot tracking vertical motion, this is not a major factor. Besides being convenient in setting the correct camera height, this vertical motion does not serve much other purpose in this system. However if vertical tracking was required (another sensor would be needed), a worm gear and stepper motor would not be recommended, since it results in very Jagged, slow movements. The horizontal motion had only slightly jagged movements which further adds to the reason why servo motors would have been a more suitable choice in this system. Some

minor lag was also noticed in the camera tracking, which could be improved by adjusting some of the tracking parameters in the 'Command' node.

Perhaps the most significant finding of this project was the tracking resilience attained by using a mm-wave sensor. Even with the imperfect algorithms used, the tracking resilience was excellent for two of the three tests conducted. The test which had the poorest result was the fast moving object tracking, which caused the system to lose its lock on the target. This is because the target was relatively close to the sensor, and the angular velocity of the system could not keep up. Another minor issue was other high radar reflective object would sometimes take the focus of the camera tracking system. However with further work and modifications to the algorithms, these issues could be eliminated. Where the mm-wave sensor made the greatest improvement, was the resistance to poor lighting conditions. The tracking performance was not impacted in any way when it operated in complete darkness or with extreme glare. While these conditions would not produce usable video images from a standard camera, this type of system would still be useful for steering other devices such as infrared cameras, lasers, weapons or fire hoses etc.

The final conclusion which the testing uncovered is the mm-wave sensors depth perception feature dramatically improved the resistance to the system being distracted from other people. The tests conducted determined that it was extremely difficult for another person to steal the focus of the tracking system, once a lock was achieved on a particular human target. This kind of distraction resistance is ideal for situations where the camera focus needs to remain on one person, such as a speaker in front of an audience, which is the main purpose of this system. While there are still some minor issues with tracking, further development could iron out these problems. Even with the system in its current form, it can be used effectively as a camera tracking system under most circumstances. If the system gets distracted from another strong radar reflection, the corner reflector can be used to re-focus the system when needed.

5.5.2 General Conclusions

During the development and testing of the system, several conclusions can be drawn. When considering the project from start to finish, the reality of how certain aspects progress is much different to the initial expectations. This is evident by the amount of difficulties encountered and project re-directs that occurred. Perhaps the main conclusion from all this is that projects do not always play out the way they are expected, and changes in methodologies are sometimes necessary for the project to be completed successfully. Problems and setbacks are usually seen as negative experiences, however these can also be the greatest learning experiences, and this is definitely true for this project.

Several setbacks which occurred during this project was related to some of the hardware choices which were made early in the project. While it was convenient selecting steppers and the Jetson Nano as these components were already in possession, these components caused much of the setbacks later in the project. For example, the stepper motors required additional driver boards to operate, and the software to operate the stepper motors did not exist on the Jetson Nano. This resulted in a massive increasing in development time in writing control code for the stepper motors. In addition to this, it was discovered that the Jetson Nano does not have the capability to receive analogue inputs natively, which required the purchase of another IC, adding even more development time to the project. For developing a system of this type, it would have been a much smoother process if servo motors were used, since their control is much simpler and their accuracy is still sufficient. The other conclusion is that using a more popular micro-controller such as the raspberry pi would have also made the development of this project much simpler. This is because the raspberry pi has many software libraries already developed (like stepper controllers and I2C communication) which could have been adopted into the system.

Other difficulties arose from selecting the particular mm-wave sensor, the IWR1642BOOST. When attempting to connect this device, it became clear that communicating with this device was not a trivial process. For this reason an already existing ROS node was selected to interface with the sensor. This requires nearly all of the coding which already was written to be re-worked to be usable in a ROS system adding much more work to the project. These kinds of experiences illustrate why is it so important to undertake some investigative research into potential component choices before they are committed to. Even though implementing this project as a ROS system was not the original plan and required much more work, it can

be concluded that ROS is a very useful and versatile framework for complex systems. This system also gains some added advantages by using the ROS framework such as reusable code blocks, easy system expansion and easy control over networks.

Overall, it was found that this system does have some minor issues, but further development could easily address most of these concerns. It was also found that the idea of pairing a mm-wave sensor with a camera tracking system does have merit, since it has inherent sensing properties which make it very resistance to disturbances.

5.7 Chapter 5 Summary

Chapter 5 focussed on testing the camera tracking system, discussing the results and drawing conclusions. Tracking performance and the resistance to disturbances were the two main categories of system attributes which were of interest for testing. These tests yielded quantifiable results such as maximum tracking: scope, speed, distance and angular velocity. Other tests resulted in more observational findings such as tracking stability and resistance to: poor lighting, fast movements and distractions. With the results, several strengths and weaknesses were identified, which opened discussions for system usefulness and improvement strategies. A final reflection of the entire project was also considered and several learned lessons were identified. The main conclusion was that the idea of pairing a mm-wave sensor with a camera tracking system does indeed bring some useful advantages that can be used to improve tracking resilience.

References

- Alvarez, A 2022, '*I2C Input on Jetson Nano*', Millspaw Electronics, Mason, Ohio, USA, viewed 9 May 2023, <<https://www.millspawelectronics.com/2022/07/25/i2c-input-on-jetson-nano>>.
- Bourns 2015, '*3382 12 mm Rotary Position Sensor*', Bourns Electronics, Riverside, California, USA, viewed 6 September 2023, <https://www.bourns.com/docs/product-datasheets/3382.pdf?sfvrsn=8e3b8ff1_9>.
- Chen, A, Wang, X, Shi, K, Zhu, S, Chen, Y, Fang, B, Chen, J, Huo, Y & Ye, Q 2022, '*ImmFusion: Robust mmWave-RGB Fusion for 3D Human Body Reconstruction in All Weather Conditions*', arXiv preprint arXiv:2210.01346.
- Collins, A 2021, '*How to Set Up a Camera for NVIDIA Jetson Nano*', Automatic Addison, Atlanta, Georgia, USA, viewed 6 May 2023, <<https://automaticaddison.com/how-to-set-up-a-camera-for-nvidia-jetson-nano/>>.
- Gupta, K, Srinivas, M, Soumya, J, Pandey, OJ & Cenkeramaddi, LR 2022, '*Automatic Contact-less Monitoring of Breathing Rate and Heart Rate utilizing the Fusion of mmWave Radar and Camera Steering System*', IEEE Sensors Journal.
- Huang, X, Cheena, H, Thomas, A & Tsoi, JKP 2021, '*Indoor Detection and Tracking of People Using mmWave Sensor*', Journal of Sensors, vol. 2021, p. 6657709.
- Jaiswal, S & Pandey, MK 2021, '*A Review on Image Segmentation*', Rising Threats in Expert Applications and Solutions, Springer Singapore, Singapore, pp. 233-40.
- JetsonHacks 2020, '*CSI-Camera/simple_camera.py*', JetsonHacks, California, USA, viewed 6 May 2023, <https://github.com/JetsonHacksNano/CSI-Camera/blob/master/simple_camera.py>.
- Kumar, A 2020, '*How to Use GPIO Pins on Jetson Nano Developer Kit*', Maker Pro, Boise, Idaho, USA, viewed 6 May 2023, <<https://maker.pro/nvidia-jetson/tutorial/how-to-use-gpio-pins-on-jetson-nano-developer-kit>>.
- Last Minute Engineers 2018, '*Control Stepper Motor with DRV8825 Driver Module & Arduino*', Last Minute Engineers, viewed 10 May 2023, <<https://lastminuteengineers.com/drv8825-stepper-motor-driver-arduino-tutorial>>.
- Lovescu, C & Rao, S 2020, '*The fundamentals of millimeter wave radar sensors*', Texas Instruments, Dallas, Texas, USA, viewed 27 October 2023, <<https://www.ti.com/lit/wp/spyy005a/spyy005a.pdf>>.
- McDaniel, B 2004, '*How To Turn On An ATX Power Supply Without A Motherboard*', Overclockers Club, Vancouver, Canada, viewed 12 May 2023, <https://www.overclockersclub.com/guides/atx_psu_startup>.
- McGrath, D 2021, '*Overcome 5G mmWave measurement issues*', 5G Technology World, viewed 12 September 2023, <<https://www.5gtechnologyworld.com/overcome-5g-mmwave-measurement-issues>>.


- NVIDIA 2023, '*Jetson Nano Developer Kit and Module*', NVIDIA, Santa Clara, California, USA, viewed 29 September 2023, <<https://www.nvidia.com/en-au/autonomous-machines/embedded-systems/jetson-nano/product-development>>.
- Prakash, A 2021, 'What is Shebang in Linux Shell Scripting?', Linux Handbook, Reykjavík, Iceland, viewed 30 September 2023, <<https://linuxhandbook.com/shebang>>.
- Redish, WL 1976, *Airborne ballistic camera tracking systems*, NASA, Washington D.C.
- Texas Instruments 2014, '*DRV8825 Stepper Motor Controller IC*', Texas Instruments, Dallas, Texas, USA, viewed 10 May 2023, <<https://www.ti.com/lit/ds/symlink/drv8825.pdf>>.
- Texas Instruments 2018, '*ADS111x Ultra-Small, Low-Power, I²C-Compatible, 860-SPS, 16-Bit ADCs With Internal Reference, Oscillator, and Programmable Comparator*', Texas Instruments, Dallas, Texas, USA, viewed 6 September 2023, <<https://www.ti.com/lit/ds/symlink/ads1115.pdf>>.
- Texas Instruments 2020, '*mmWave radar sensors*', Texas Instruments, Dallas, Texas, USA, viewed 10 October 2022, <<https://www.ti.com/sensors/mmwave-radar/overview.html>>.
- Wei, Z, Zhang, F, Chang, S, Liu, Y, Wu, H & Feng, Z 2022, '*MmWave Radar and Vision Fusion for Object Detection in Autonomous Driving: A Review*', Sensors (Switzerland), vol. 22, no. 7, p. 2542.
- Wolff, C (2007) '*Corner Reflectors*', Radar Tutorial, Rostock, Germany, viewed 13 October 2023, <<https://www.radartutorial.eu/17.bauteile/bt47.en.html>>.
- Zhang, L 2019, '*TI mmWave ROS Package (Customized)*', University of Arizona, Tuscan, Arizona, USA, viewed 2 September 2023, <https://github.com/radar-lab/ti_mmwave_ropkg>.
- Zhao, P, Lu, CX, Wang, J, Chen, C, Wang, W, Trigoni, N & Markham, A 2019, '*mID: Tracking and Identifying People with Millimeter Wave Radar*', IEEE xplore, 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS), pp. 33-40.

Appendix A

Risk Assessment

There will be some level of risk associated with the development of this project during the different phases. While proceeding with this project, many factors will cause some level of risk. The negative consequences of this risk could be: loss or damage to property, personal injury, illness or death. It is necessary to take steps to mitigate this risk and for this reason a Risk Management Plan (RMP) was completed:

Table D1: Risk Management Plan (Background Information)



University of Southern Queensland

Offline Version

USQ Safety Risk Management System

Note: This is the offline version of the Safety Risk Management System (SRMS) Risk Management Plan (RMP) and is only to be used for planning and drafting sessions, and when working in remote areas or on field activities. It must be transferred to the online SRMS at the first opportunity.

Safety Risk Management Plan – Offline Version			
Assessment Title:	MM-Wave Assisted Automatic Camera Tracking System	Assessment Date:	24/05/2023
Workplace (Division/Faculty/Section):	Faculty of Engineering, UniSQ	Review Date:(5 Years Max)	24/06/2023
Context			
Description:			
What is the task/event/purchase/project/procedure?	Final Year Project		
Why is it being conducted?	Development of camera tracking system for final year project.		
Where is it being conducted?	17 Stieler Drive, Plainland, Queensland & USQ Toowoomba		
Course code (if applicable)	ENG4111	Chemical name (if applicable)	N/A
What other nominal conditions?			
Personnel involved	Andrew Jawney, Craig Lobsey		
Equipment	MM-wave sensor (IWR1642) development kit, Jetson Nano, Jetson OS (Linux), HD Camera, Camera tracking system, Gimbal hardware, Stepper Motor, Stepper driver boards, USB Cable, HDMI Cable, Power supply, Soldering Equipment, Power connector (Male), Power connector (Female), Perf Board, 220 µF Capacitor, Jumper pins (Female), Jumper pins (Male), Jumper cables, Microsoft word, Desk, Office Chair, Monitor, Keyboard, Mouse, and Modem.		
Environment	Indoors, air conditioned, well lit, well ventilated area.		
Briefly explain the procedure/process	Software development, hardware testing, CAD Design, 3D Printing, Soldering, word processing		
Assessment Team - who is conducting the assessment?			
Assessor(s)	Andrew Jawney		
Others consulted:	Craig Lobsey		

Table D2: Risk Tolerance Matrix

Eg 1. Enter Consequence

	Consequence				
Probability	Insignificant No Injury 0-\$5K	Minor First Aid \$5K-\$50K	Moderate Med Treatment \$50K-\$100K	Major Serious Injuries \$100K-\$250K	Catastrophic Death More than \$250K
Almost Certain 1 in 2	M	H	E	E	E
Likely 1 in 100	M	H	H	E	E
Possible 1 in 1000	L	M	H	H	H
Unlikely 1 in 10 000	L	L	M	M	M
Rare 1 in 1 000 000	L	L	L	L	L

Eg 2. Enter Probability

Eg 3. Find Action

Recommended Action Guide

E=Extreme Risk – Task **MUST NOT** proceed

H=High Risk – Special Procedures Required (See USQSafe)

M=Moderate Risk – Risk Management Plan/Work Method Statement Required

L=Low Risk – Use Routine Procedures

Table D3: Risk Management Plan (Steps 1-5)

Step 1 (cont)	Step 2	Step 2a	Step 2b	Step 3			Step 4				
Hazards: From step 1 or more if identified	The Risk: What can happen if exposed to the hazard without existing controls in place?	Consequence: What is the harm that can be caused by the hazard without existing controls in place?	Existing Controls: What are the existing controls that are already in place?	Risk Assessment: Consequence x Probability = Risk Level			Additional controls: Enter additional controls if required to reduce the risk level	Risk assessment with additional controls:			
				Probability	Risk Level	ALARP? Yes/no		Consequence	Probability	Risk Level	
Example											
COVID-19	Contracting Covid, illness, nausea, fatigue	Minor	Wash hands, disinfect work surfaces.	Unlikely	Low	No		Select a consequence	Select a probability	Select a Risk Level	Yes or No
Car Accident	Damage or destruction of property. Serious injury or death.	Catastrophic	Seatbelts, Airbags, Stability control, ABS, Adhere to speed limits, Driving only when alert and un-impaired, full comprehensive insurance, good tyres, mechanically sound vehicle	Rare	Low	No		Select a consequence	Select a probability	Select a Risk Level	Yes or No
Use of AC power mains for equipment	Power trip, circuit overload, electric shock, loss of data.	Moderate	Wire insulation, Australian standards, Work with power off, Circuit breakers.	Rare	Low	No		Select a consequence	Select a probability	Select a Risk Level	Yes or No
Motors and moving parts	Pinched finger, hardware damage	Minor	Low torque motors, keep objects and fingers away from moving parts, software protection, isolation switch.	Unlikely	Low	No		Select a consequence	Select a probability	Select a Risk Level	Yes or No
Robbery, Terrorism, Fire or medical emergency	Damage or destruction of property. Serious injury or death.	Catastrophic	Authorities notified of event, emergency response plan developed, fire extinguishers, clear access to multiple exits, emergency assembly points.	Rare	Low	No		Select a consequence	Select a probability	Select a Risk Level	Yes or No

Step 5 - Action Plan (for controls not already in place)			
Additional controls:	Resources:	Persons responsible:	Proposed implementation date:
			Click here to enter a date.

Appendix B

Ethical & Environmental Considerations

While undertaking the tasks involved in project development, it is essential to acknowledge and address the ethical and environmental considerations that may arise. One ethical consideration that may arise is the unintentional filming of individuals without their permission, which not only has the potential to cause distress but also exposes the project to potential legal consequences.

In relation to environmental considerations for the project, the impacts will be very minimal. Some electricity will be used to operate the equipment, which may lead to minor release of CO₂ gases. Additionally, all electronics used will be properly recycled at the end of their lifecycles. There may be some production of waste materials during the 3D printing and prototyping phase of development.

While there may be some minor ethical and environmental considerations, they are not expected to significantly impact the overall progress of the project.

Appendix C

Resource Planning

There are a number of different resources which will be required for this project to be completed successfully. These resources, as well as their expected associated costs are described in Table D.1 below:

Table C.1: Resource Requirements & Acquisition Details

Resource	Quantity	Supplier	Cost
MM-wave sensor (IWR1642) development kit	1	UniSQ	N/A
Jetson Nano	1	UniSQ	N/A
Jetson OS (Linux)	1	UniSQ	N/A
HD Camera	1	UniSQ	N/A
Camera tracking system	1	Purchase	\$99.00
Gimbaling hardware	1	3D Printed	N/A
Stepper Motor	2	Student	N/A
Stepper driver boards	2	Purchase	\$7.18
USB Cable	2	Student	N/A
HDMI Cable	1	Student	N/A
Power supply	1	Student	N/A
Soldering Equipment	1	Student	N/A
Power connector (Male)	3	Purchase	\$7.65
Power connector (Female)	1	Purchase	\$2.45
Perf Board	1	Student	N/A
220 μ F Capacitor	1	Student	N/A
Jumper pins (Female)	2	Purchase	\$2.90
Jumper pins (Male)	2	Purchase	\$5.90
Jumper cables	1	Purchase	\$6.95
Microsoft word	1	Student	N/A
Desk	1	Student	N/A
Office Chair	1	Student	N/A
Monitor	3	Student	N/A
Keyboard	2	Student	N/A
Mouse	2	Student	N/A
Modem	1	Student	N/A
TOTAL			\$132.03

Appendix D

Project Phases

D.1 Gathering

The gathering phase of the project is primarily related to sourcing all of the materials that will be needed for the project. While several items are already in possession, many other components may be found online or at Jaycar electronics. Some components will be sourced from USQ like the IWR1642 development kit, camera, microcontroller and 3D printing services. Some components may require visiting multiple locations, while others can conveniently be delivered to the home address.

Some items needed for the development of the camera tracking system are specific software platforms. It is likely that CAD software, Linux, Python, Microsoft word and Windows 10 will also be needed. Some of these software packages require a licence, most of which are accessible through student licenses held by the University. It is possible that certain important items could be overlooked initially or that replacement hardware may be needed during advanced project phases. As a result, the gathering phase may continue periodically throughout all project stages. For this reason, the final aspect of the gathering phase, which involves applying for financial reimbursement, will be intentionally delayed.

D.2 Setup

The setup phase involves organising and preparing the gathered resources for testing and development. This includes tasks like software installation, hardware assembly, and component connectivity. The main focus of this phase is to ensure the functionality and proper interfacing of all essential components, including the Jetson Nano, camera, motors, and power supply. It is important to verify camera is operational, and its data stream is able to be captured and potentially relayed to other devices. Another important aspect is the successful control of the motors in order to effectively steer the camera. It is also important to enable connectivity of the mm-wave sensor, in order to receive the stream of data it will generate. Both of these aspects this will require iterative testing and development of control code. Lastly, the setup phase entails the design and fabrication of a suitable camera gimbaling system consisting of brackets and mounts. It is important to organise all components effectively, including deciding whether the mm-wave sensor should be mounted stationary or move along with the camera.

Minor complications are anticipated during the setup phase, such as communication issues between components and initial configuration problems. These challenges will be addressed by employing strategies like internet searches for common solutions, trial and error troubleshooting, and reaching out to the manufacturer for assistance if needed. In the unlikely event that the issue persists, the project supervisor will be consulted for further guidance.

D.3 Development

The development phase will mainly be concerned with using the IWR1642 sensor and attempting to interpret its data stream. This data stream will be collected and processed by the Jetson Nano through the use of several algorithms and procedures which were identified in the [literature review](#). However, it is crucial to test and verify the methods employed in previous research before applying them to this project.

A significant task in this phase will involve determining how to utilise the interpreted sensor data to intelligently steer the camera to track a person. It is expected that the sensor data interpretation, object tracking through software, and operation of camera motors will have been achieved at this stage. However, the crucial missing link lies in establishing a seamless connection between the software tracking and the motor commands. Sending simple ‘static’ commands to the camera will likely lead to jagged motions. To overcome this, a PID control system will need to be developed. A PID control system will allow the camera to move smoothly, quickly and efficiently to a target location; but these systems need to be tuned to achieve optimum performance.

D.4 Testing

Testing will commence once the system is adequately operational which is undertaken to ensure that the camera tracking system meets all of the project objectives. This will involve conducting performance comparisons between the developed system and another commercially available system. The intended areas of improvement for the camera tracking system include enhancing its performance in challenging lighting conditions and increasing its immunity to external distractions. As a result, several tests will be conducted under different lighting conditions in order to compare performance of both camera tracking systems. One test will be conducted under good lighting conditions, the next test will be performed at night with the lights off, the next test will involve using a smoke machine to obscure the camera's view, and another test will examine the camera's response to a bright light shining directly toward it. The final test will introduce other people into the field of view to assess the system's resistance to distractions. The results of these tests will be tabulated and further discussed in the final dissertation.

D.5 Documentation

Ideally this phase will be completed in parallel to all other phases, in order to more easily write the final thesis paper at the end of development. However, there is a possibility that it may not receive equal attention due to the main focus being on other phases of the project. To mitigate this, maintaining consistent records throughout all phases is vital, this will reduce the reliance on memory when writing the final dissertation. This consistent documentation may be in the form of physically written notes or electronic records. Regardless of the recording method, the final writing of the thesis will be done using word processing software, which involves tasks such as typing, editing, formatting, and performing spelling and grammar checks. The thesis mainly consists of describing in detail the procedures which were used to develop the camera tracking system. It also includes an introduction and justification of the idea, linking other relevant research, justification of design choices and discussing the key outcomes of the whole process. Once all aspects of the dissertation have been written, it will be proof-read to further improve its quality before final submission.

Appendix E

Project Timelines

With the project phases being clearly defined, the expected timeline can be estimated. While it is possible to give a rough indication on what aspects will be achieved at certain times, some factors could dramatically change the timeline. Certain factors such as unforeseen development problems, unexpected family emergencies, supply issues and greater than expected workloads. To achieve all aspects of the project there has been eight months allocated for all stages of development. This time period is undoubtedly adequate to complete all aspects of the specified progression phases. Assuming there are none of these setbacks, the project timeline should look something like [Figure E.1](#) below:

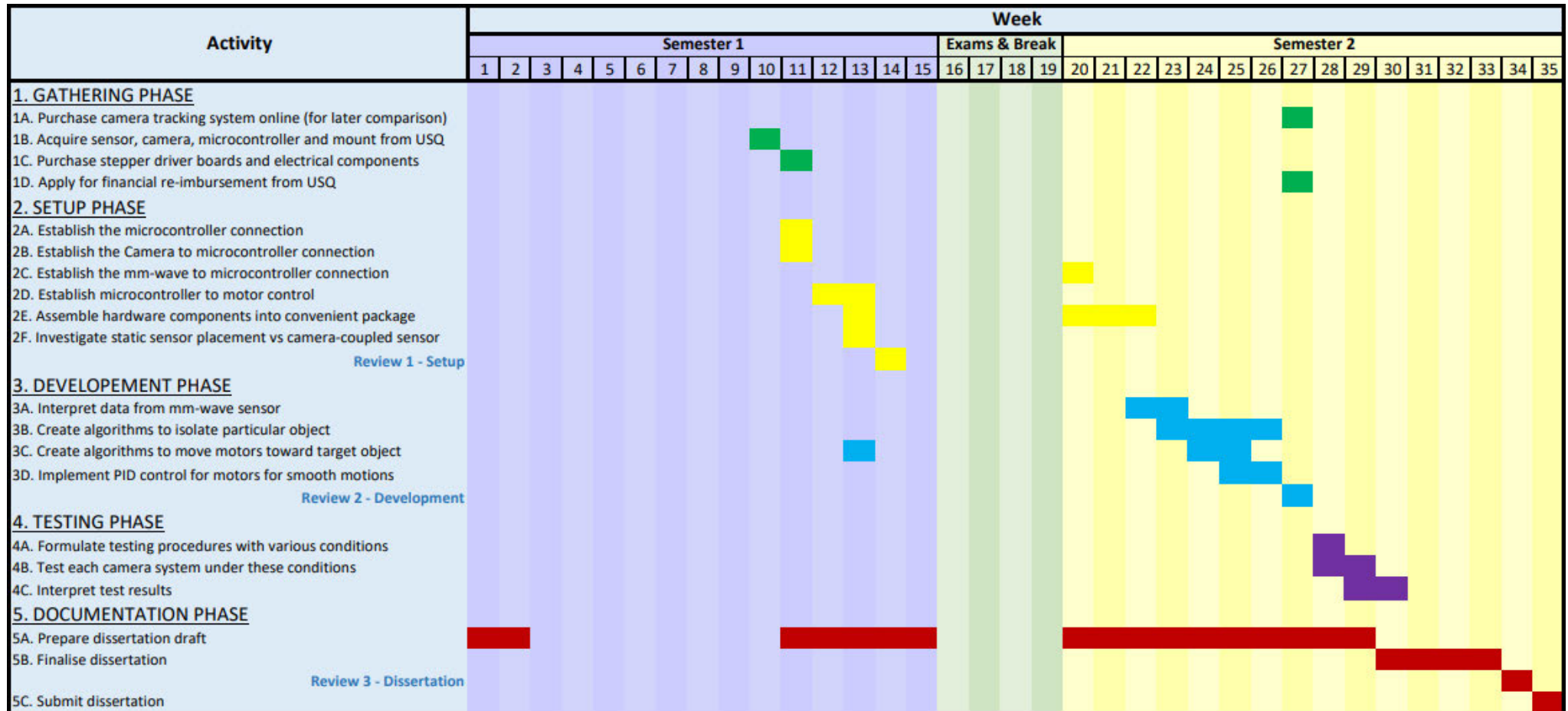


Figure E.1: Gantt Chart Of Timeline For Final Year Project

Appendix F

Original Stepper Control Code

```
#!/usr/bin/env python
# Creator: Andrew Jawney
# Date: 12/5/2023
# This code controls two stepper motors rotation speeds and direction simultaneously.
# When Run, a popup window will appear with two sliders, which are used to control each
# stepper motor. Every time a slider bar is moved, the stepper will change its speed or
# direction accordingly, and the terminal will display the new speed or direction value
# of the corresponding stepper motor. This code has been designed to run the stepper
# motors in the background, while other tasks can be completed simultaneously.
# The intention is to build upon this code to connect a mm-wave sensor and use its
# data to control the stepper motors in real time to steer a camera toward a person.

# Import Libraries
import Jetson.GPIO as GPIO
import time
import threading
import tkinter

# Define the GPIO pins for the stepper motor drivers
ENABLE_PIN = 7
MODE1_0 = 11
MODE1_1 = 13
MODE1_2 = 15
MODE2_0 = 29
MODE2_1 = 31
MODE2_2 = 33
STEP_PIN_1 = 19
DIR_PIN_1 = 21
STEP_PIN_2 = 35
DIR_PIN_2 = 37

# Set up the GPIO channel
GPIO.setmode(GPIO.BOARD)
GPIO.setup(ENABLE_PIN, GPIO.OUT, initial=GPIO.HIGH)
GPIO.setup(MODE1_0, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(MODE1_1, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(MODE1_2, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(MODE2_0, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(MODE2_1, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(MODE2_2, GPIO.OUT, initial=GPIO.LOW)
```



```
GPIO.setup(STEP_PIN_1, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(DIR_PIN_1, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(STEP_PIN_2, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(DIR_PIN_2, GPIO.OUT, initial=GPIO.LOW)
```

Create class to control stepper motor/s:

```
class Stepper():
```

Initialise variables:

```
    def __init__(self, Controller, Motor, Mode):
        self.Controller = Controller
        self.SliderValue = 0
        self.Motor = Motor
        self.next_time = time.time()
        self.i=0
        self.done=False
        self.toggle = False
        self.increment = 0.002
        self.mode = Mode
        self.Direction = 0
        if self.Motor == 1:
            self.M0 = MODE1_0
            self.M1 = MODE1_1
            self.M2 = MODE1_2
            self.STEP = STEP_PIN_1
            self.DIR = DIR_PIN_1

        if self.Motor == 2:
            self.M0 = MODE2_0
            self.M1 = MODE2_1
            self.M2 = MODE2_2
            self.STEP = STEP_PIN_2
            self.DIR = DIR_PIN_2
```

Create Slider Controller:

```
    self.SliderBar = tkinter.Scale(self.Controller, from_=-10, to=10, resolution=0.01,
    orient="horizontal", length=250, label=("Stepper",self.Motor, "Direction & Speed:"),
    command=self.UpdateSlider)
    self.SliderBar.pack()
```

```
print("| Stepper:", self.Motor, " | Mode:", self.mode, " |")
GPIO.output(ENABLE_PIN, GPIO.HIGH)
self._run()
```

This part runs every time slider is moved:

```
def UpdateSlider(self, value):
```

```
# Set Direction for stepper
```

```
    self.SliderValue = (float(value))
    if self.SliderValue > 0:
        GPIO.output(self.DIR, GPIO.HIGH)
        self.toggle = True
    if self.SliderValue < 0:
        GPIO.output(self.DIR, GPIO.LOW)
        self.toggle = True
    if self.SliderValue == 0:
        GPIO.output(self.DIR, GPIO.HIGH)
        self.toggle = False
        GPIO.output(ENABLE_PIN, GPIO.LOW)
        self.increment = 0.0002
    else:
        GPIO.output(ENABLE_PIN, GPIO.HIGH)
        self.increment = 0.02/(abs(self.SliderValue))
```

```
# Set Boundaries for stepper speeds
```

```
    if self.increment < 0.002:
        self.increment = 0.002
    if self.increment > 0.02:
        self.increment = 0.02
```

```
# Set Step size for stepper speeds
```

```
    if self.mode == 1: # Full Step Mode
        GPIO.output(self.M0, GPIO.LOW)
        GPIO.output(self.M1, GPIO.LOW)
        GPIO.output(self.M2, GPIO.LOW)

    if self.mode == 2: # 1/2 Step Mode
        GPIO.output(self.M0, GPIO.HIGH)
```

```
GPIO.output(self.M1, GPIO.LOW)
GPIO.output(self.M2, GPIO.LOW)

if self.mode == 3: # 1/4 Step Mode
    GPIO.output(self.M0, GPIO.LOW)
    GPIO.output(self.M1, GPIO.HIGH)
    GPIO.output(self.M2, GPIO.LOW)

if self.mode == 4: # 1/8 Step Mode
    GPIO.output(self.M0, GPIO.HIGH)
    GPIO.output(self.M1, GPIO.HIGH)
    GPIO.output(self.M2, GPIO.LOW)

if self.mode == 5: # 1/16 Step Mode
    GPIO.output(self.M0, GPIO.LOW)
    GPIO.output(self.M1, GPIO.LOW)
    GPIO.output(self.M2, GPIO.HIGH)

if self.mode == 6: # 1/32 Step Mode
    GPIO.output(self.M0, GPIO.LOW)
    GPIO.output(self.M1, GPIO.HIGH)
    GPIO.output(self.M2, GPIO.HIGH)

# Print Motor status:
print("Motor:", self.Motor, "Increment:", self.increment, " SliderValue:",
self.SliderValue)

# This Part sends the step signals for the steppers:
def _run(self):
    if self.i % 2 == 0:
        self.next_time += self.increment
        GPIO.output(self.STEP, GPIO.LOW)
    else:
        self.next_time += 0.00002
        if self.toggle == True:
            GPIO.output(self.STEP, GPIO.HIGH)

    self.i += 1
    if not self.done:
```

```
threading.Timer( self.next_time - time.time(), self._run).start()
```

```
# This part will stop the steppers
```

```
def stop(self):  
    self.done=True  
    GPIO.output(ENABLE_PIN, GPIO.LOW)  
    print("Stepper OFF")
```

```
# End of 'Stepper' Class
```

```
#-----
```

```
# Start of main program loop:
```

```
Controller = tkinter.Tk()  
Controller.title("Motor Controller")  
Stepper_1 = Stepper(Controller,1,1) # (Controller, Motor, Mode)  
Stepper_2 = Stepper(Controller,2,1) # (Controller, Motor, Mode)  
Controller.mainloop()  
Stepper_1.stop()  
Stepper_2.stop()  
GPIO.cleanup()
```

Appendix G

Command Node Script

```
#!/usr/bin/env python

# Creator: Andrew Jawney
# This code receives data from the mm-wave sensor node and the angle sensor node.
# With the received point cloud data, the most intense point is isolated.
# If the radar reflector is not detected, the points which are near its last known whereabouts are isolated.
# The most intense point from these isolated points is selected.
# Using this point's 'y' value, and the angle of the horizontal stepper, the speed signal is generated.
# The speed signals are published for the stepper control nodes

# Import Libraries
import rospy
from sensor_msgs.msg import PointCloud2
import sensor_msgs.point_cloud2 as PC2_functions
from std_msgs.msg import Float32MultiArray

class Command:
    def __init__(self): # Initialise the variables needed for this node
        self.motor_msg = Float32MultiArray() # Create message structure for motor commands
        self.angles = [0,0] # Variable for storing Vertical and Horizontal angles
        self.speed = 0 # Current motor speed for Horizontal stepper
        self.old_speed = 0 # Previous motor speed for Horizontal Stepper
        self.follow_point = [0, 0, 0] # Point selected to follow
        self.detect_range = [0.15, 1.0, 0.5] # when no point is selected search in this area [x_min, x_max,
y_max]
        self.search_area = [0.2, 0.3] # distance from last detected point to search [x, y]

    def filter_intensity(self, data):
        self.filtered_points = [] # clear array of filtered points

        # Scan through pointcloud data:
        for point in PC2_functions.read_points(data,
            field_names=("x", "y", "z", "intensity"),
            skip_nans=True):

            # When the radar reflector is detected, add to filtered points
            if point[3] > 33:
                self.filtered_points.append(point)
```

```

# When followed point has been lost, look in this area for intense data points
if self.follow_point == [0, 0, 0]:
    # This is set to between 15cm to 1m away from sensor and within 50cm to the left or right:
    if ( (abs(point[0]) > self.detect_range[0]) and
        (abs(point[0]) < self.detect_range[1]) and
        (abs(point[1]) < self.detect_range[2])):
        self.filtered_points.append(point)
    else:
        # If a point has been followed, look for this point again within this search range
        # This is set to 20cm closer or further, and 30cm to the left or right:
        if ((abs( abs(point[0]) - abs(self.follow_point[0]) ) < self.search_area[0] ) and
            (abs( abs(point[1]) - abs(self.follow_point[1]) ) < self.search_area[1] )):
            self.filtered_points.append(point)

# If points are found that meet the above conditions, select the most intense point:
if len(self.filtered_points) >= 1:
    self.follow_point = max(self.filtered_points, key=lambda point: point[3])

self.set_speed() # Determine speed for horizontal stepper
self.display() # Display current state

# This part determines an acceptable spin speed and direction for the horizontal stepper
def set_speed(self):
    self.speed = float(20*(self.follow_point[1])) # set speed to tracked point (y coord)

# Prevent large changes in speed (this happens if detected point jumps suddenly to another location)
if (abs(abs(self.speed) - abs(self.old_speed)) > 0.5):
    if self.speed > self.old_speed:
        self.speed = self.old_speed + 0.2
    if self.speed < self.old_speed:
        self.speed = self.old_speed - 0.2

# This part slows down (or stops) the speed the closer the camera is centered on target
if ((abs(self.follow_point[1])) < 0.04):
    self.speed = 0
elif ((abs(self.follow_point[1])) < 0.08):
    self.speed = self.speed*0.0625
elif ((abs(self.follow_point[1])) < 0.15):

```

```

    self.speed = self.speed*0.125

# If the object is tracked out of range, then stop the motor spinning, and the followed point is reset.
if (self.angles[1] > 60) and (self.speed >= 0):
    self.speed = 0
    self.follow_point == [0, 0, 0]
if (self.angles[1] < -60) and (self.speed <= 0):
    self.speed = 0
    self.follow_point == [0, 0, 0]

# Limit the maximum speed to 5 or below
if abs(self.speed) > 5:
    self.speed = 5*(self.speed/abs(self.speed))

self.motor_msg.data = [0, self.speed, 0] # Assemble message ready for publishing
self.old_speed = self.speed # Save the current speed for next program cycle.
Pan_camera.publish(self.motor_msg) # Send speed command to horizontal motor


# Display tracking infomation to console (Debugging purposes)
def display(self):
    print("Point: [{:.2f}, {:.2f}] | Speed: {:.1f} | Vert Angle: {:.1f} | Hor Angle: {:.1f}
".format(self.follow_point[0], self.follow_point[1], self.speed, self.angles[0], self.angles[1]))


# Fetch Angle data:
def Get_Angles(self, Angle_msg):
    self.angles = Angle_msg.data


# Set up ROS subscribers for Angle data and mm-wave data
def listener(self):
    rospy.init_node('intensity_filter', anonymous=True)
    rospy.Subscriber('Angles', Float32MultiArray, Commander.Get_Angles)
    rospy.Subscriber('/ti_mmwave/radar_scan_pcl', PointCloud2, Commander.filter_intensity)
    rospy.spin()


# Main program loop:
if __name__ == '__main__':

```



```
try:
    # Setup ROS publisher for sending motor commands:
    Pan_camera = rospy.Publisher('Slider_values', Float32MultiArray, queue_size=10)
    Commander = Command() # Create an instance of 'Command' class
    Commander.listener() # Call the ROS listener function
except rospy.ROSInterruptException:
    pass
```

Appendix H

Angles Node Script

```
#!/usr/bin/env python

# Creator: Andrew Jawney
# This code receives the information from the angle sensors and publishes this information.
# The angle values are read as analogue values by the ADS1115, which sends this information via the I2C
# protocol.
# This node directs the ADS1115 which channel to read and specifies other configuration settings for this IC.
# Once the ADS1115 receives this message, it converts the applicable analogue value to digital and writes it to
# the I2C bus.
# This script waits for a very specific time before reading this result.
# Once this data has been read, it is converted to decimal and then scaled appropriately to convert this to the
# proper angle.
# This process is done for both the horizontal and vertical angle sensors.
# The previous angle value is retained so any stray values received can be ignored.
# The horizontal and vertical angle values are then published to the topic called 'Angles' in a simple array.

# Import Libraries
import time
import rospy
import smbus
from std_msgs.msg import Float32MultiArray

class Angle_Publisher:

    def __init__(self):
        rospy.init_node('Calc_Angles') # Initialise ROS node
        self.ang_publisher = rospy.Publisher('Angles', Float32MultiArray, queue_size=10) # Set up ROS
        publisher to send Angle data
        self.I2C_BUS = 1 # Define the I2C bus number
        self.i2c = smbus.SMBus(self.I2C_BUS) # Create I2C bus object
        self.ang_msg = Float32MultiArray() # Create message structure for Angle data

        # Configuration values for the ADS1115:
        self.ADS_ADDRESS = 0x48 # ADS1115 I2C address
        self.ADS_CONFIG_REG = 0x01 # Set up configuration register for ADS1115
        self.ADS_SINGLE_SHOT = 0x8000 # Set ADS1115 to read and send one value at a time
        self.ADS_AIN0 = 0x4000 # Channel address for Vertical angle
        self.ADS_AIN1 = 0x5000 # Channel address for Horizontal angle
        self.Angles = [0, 0] # Store angle data
```

```

self.Prev = [0, 0] # Store previous angle data
self.delay = 0.11891 # Delay between sending I2C message and reading result

self.fetch_throw_angle() # Read angle data ad publish it

# Read, filter and publish angle data:
def fetch_throw_angle(self):
    while not rospy.is_shutdown():

        self.Angles[0] = self.read_adc(0) # Read Vertical Angle
        self.Angles[1] = self.read_adc(1) # Read Horizontal Angle

        # Remove stray values:
        while (self.Angles[0] < -50) or (self.Angles[0] > 50):
            time.sleep(0.2) # Introduce delay to re-synce timing for reading values
            self.Angles[0] = self.Prev[0] # Set to previous valid value

        while (self.Angles[1] < -86) or (self.Angles[1] > 100):
            time.sleep(0.2) # Introduce delay to re-synce timing for reading values
            self.Angles[1] = self.Prev[1] # Set to previous valid value

        self.Prev = [ self.Angles[0], self.Angles[1] ] # Save angle data
        self.Ang_msg.data = self.Angles # Place angle data into message structure
        self.Ang_publisher.publish(self.Ang_msg) # Publish angle data

        # Display and log Angle values to console:
        rospy.loginfo(" ANGLES | Vertical: {:.3f} Deg | Horizontal: {:.3f} Deg ".format(self.Angles[0],
self.Angles[1]))

# Read data from the ADS1115
def read_adc(self, adc_channel):

    # Assemble configuration message to send to ADS1115
    config = self.ADS_SINGLE_SHOT | self.ADS_CONFIG_REG
    if adc_channel == 0:
        config |= self.ADS_AIN0
    else:
        config |= self.ADS_AIN1

```

```
# Send config message to I2C port:
self.i2c.write_i2c_block_data(self.ADS_ADDRESS, self.ADS_CONFIG_REG, [(config >> 8) & 0xFF,
config & 0xFF])

time.sleep(self.delay) # Allow time for conversion

data = self.i2c.read_i2c_block_data(self.ADS_ADDRESS, 0x00, 2) # Read data from I2C port
value = (data[0] << 8) | data[1] # Convert byte sequence to decimal value

# Scale values accordingly:
if adc_channel == 0:
    value = ((-15 * value) / 803) + 73.505
if adc_channel == 1:
    value = ((-27 * value) / 1205) + 308.353
return value

# Main program loop
if __name__ == '__main__':
    Calc_Angles = Angle_Publisher() # Create and initialise and run an Instance of Angle_Publisher class
```

Appendix I

Sliders Node Script

```
#!/usr/bin/env python

# Creator: Andrew Jawney

# This code creates a popup window with two slider bars which are used to manually control the motion of the
# gimballing system.
# The window has two reset buttons, which set the corresponding motor speed to zero.
# There is also a reverse checkbox which inverts the signals generated for the horizontal stepper motor.
# This feature is useful for different viewing perspective when operating.
# The last feature is a disable checkbox, and when selected, will both steppers, and prevent any further control
# of the motors, until it is deselected.

# Import Libraries:
import rospy
from std_msgs.msg import Float32MultiArray
import Tkinter as tk
import signal

class SliderPublisherNode:
    def __init__(self):
        rospy.init_node('slider_publisher_node') # Initialise ROS node
        self.publisher = rospy.Publisher('Slider_values', Float32MultiArray, queue_size=10) # Set up ROS
        publisher to send slider bar values

        self.msg = Float32MultiArray() # Create message structure for motor commands
        self.Vertical_Value = 0.0 # Initialise Vertical slider value
        self.Horizontal_Value = 0.0 # Initialise Horizontal slider value
        self.Toggle_H = 1.0 # Create value to allow reversing of horizontal value
        self.Enable = 1 # Create value to enable or disable stepper motors

        self.create_slider_window() # Call window creation routine
        self.popup.mainloop() # Start tkinters main program loop

# Function which creates the user interface:
def create_slider_window(self):
    self.popup = tk.Tk() # Create tkinter object
    self.popup.title("Motor Controller (ROS PUBLISHER)") # Create title for popup window
    self.popup.geometry("400x200+10+260") # Specify size and location of tkinter window
```

```
# Create frames for vertical and horizontal stepper controls:
```

```
vertical_frame = tk.Frame(self.popup) # Create frame for vertical stepper controls
```

```
vertical_frame.pack() # Implement vertical frame
```

```
horizontal_frame = tk.Frame(self.popup) # Create frame for horizontal stepper controls
```

```
horizontal_frame.pack() # Implement horizontal frame
```

```
# Create labels for each frame:
```

```
Vertical_Label = tk.Label(vertical_frame, text="Vertical Stepper Direction & Speed:")
```

```
Vertical_Label.pack() # Implement vertical frame label
```

```
Horizontal_Label = tk.Label(horizontal_frame, text="Horizontal Stepper Direction & Speed:")
```

```
Horizontal_Label.pack() # Implement horizontal frame label
```

```
# Add Vertical Slider:
```

```
self.Vertical_Slider = tk.Scale(
```

```
    vertical_frame,
```

```
    from_=-10,
```

```
    to=10,
```

```
    resolution=0.001,
```

```
    orient="horizontal",
```

```
    length=400,
```

```
    command=lambda x: self.Fetch_and_Publish()
```

```
)
```

```
self.Vertical_Slider.pack() # Implement Vertical Slider
```

```
# Add Vertical stop button:
```

```
Vertical_Stop_Button = tk.Button(
```

```
    vertical_frame,
```

```
    text="Stop",
```

```
    command=lambda: self.Vertical_Slider.set(0)
```

```
)
```

```
Vertical_Stop_Button.pack() # Implement Vertical stop button
```

```
# Add Horizontal Slider:
```

```
self.Horizontal_Slider = tk.Scale(
```

```
    horizontal_frame,
```

```
    from_=-10,
```

```
    to=10,
```

```
    resolution=0.001,
```

```
    orient="horizontal",
```

```
    length=400,
```



```
        command=lambda x: self.Fetch_and_Publish()
    )
    self.Horizontal_Slider.pack() # Implement Horizontal Slider

    # Add Horizontal stop button:
    Horizontal_Stop_Button = tk.Button(
        horizontal_frame,
        text="Stop",
        command=lambda: self.Horizontal_Slider.set(0)
    )
    Horizontal_Stop_Button.pack(side=tk.BOTTOM) # Implement Horizontal stop button

    # Add a toggle checkbox to control horizontal direction
    Reverse_Label = tk.Label(horizontal_frame)
    Reverse_Label.pack(side=tk.RIGHT)
    Reverse_Checkbox = tk.Checkbutton(
        horizontal_frame,
        text="Reverse",
        command=self.toggle_horizontal
    )
    Reverse_Checkbox.pack(side=tk.LEFT) # Implement reverse checkbox

    # Add a toggle checkbox to disable all motors
    Disable_Label = tk.Label(horizontal_frame)
    Disable_Label.pack(side=tk.RIGHT)
    Disable_Checkbox = tk.Checkbutton(
        horizontal_frame,
        text="Disable",
        command=self.toggle_disable
    )
    Disable_Checkbox.pack(side=tk.RIGHT) # Implement disable checkbox

    self.popup.protocol("WM_DELETE_WINDOW", self.stop) # Set up a call back function to handle
window closing

# Function which fetches slider values and publishes this data to the motors
def Fetch_and_Publish(self, *args):
    self.Vertical_Value = self.Vertical_Slider.get() # Get vertical slider value
```

```

        self.Horizontal_Value = self.Horizontal_Slider.get() * self.Toggle_H # Get horizontal slider value and
invert (if applied)
        # Assemble motor control signal message:
        self.msg.data = [self.Vertical_Value, self.Horizontal_Value, self.Enable]
        self.publisher.publish(self.msg) # Publish motor signals

        # Display and log current motor control signals to console:
        rospy.loginfo("SLIDERS | Vertical: {:.3f} | Horizontal: {:.3f}".format(self.Vertical_Value,
self.Horizontal_Value))

# Toggle the horizontal slider value multiplier between 1 and -1:
def toggle_horizontal(self):
    self.Toggle_H = -self.Toggle_H
    self.Fetch_and_Publish()

# Toggle the Enable value between 1 and -1:
def toggle_disable(self):
    self.Enable = -self.Enable
    self.Fetch_and_Publish()

# Function to handle clean shutdown of node:
def stop(self):
    rospy.signal_shutdown("Slider Publisher Node: Window Closed")
    self.popup.destroy()

# Main Program loop:
if __name__ == '__main__':
    Slider_Window = SliderPublisherNode() # Create instance of SliderPublisherNode
    rospy.spin(Slider_Window) # Run ROS and tkinter window
    Slider_Window.stop() # Stop node

```

Appendix J

Horizontal Stepper Node Script

```
#!/usr/bin/env python

# This code controls the horizontal stepper motor. The commands are recieved from the 'Slider_values' topic and
# interpreted.
# Depending on where the message came from, different behaviours are achieved.
# A value is sent along with the motor signals which indicate the proiority of the message sent.
# If this value is -1 then the motor is stopped and no further commands will have an affect on the motor speed.
# The motor can only be enable by a value of 1 being recieved in its place.
# Control signals with a 1 attached take priority over signals with a 0 attached
# This horizontal stepper node has finer control of mode selection depending on specified speed.

# Import Libraries
import Jetson.GPIO as GPIO
import time
import threading
import cv2
import sys
import rospy
from std_msgs.msg import Float32MultiArray

class Stepper:
    def __init__(self):
        rospy.init_node('stepper_node', anonymous=True) # Initialise the ROS node
        rospy.Subscriber('Slider_values', Float32MultiArray, self.Priority_Decoder) # Subscribe to
Slider_values topic

        # Define the GPIO pins for the stepper motor drivers
        self.ENABLE = 7
        self.M0 = 29
        self.M1 = 31
        self.M2 = 33
        self.STEP = 35
        self.DIR = 37

        self.Speed = 0 # Target speed for motor
        self.i = 0 # Step counter
        self.done = False # Flag to indicate if motor control is done
        self.toggle = False # Toggle to enable or disable motor stepping
        self.increment = 0 # Initial increment for motor steps
```

```
self.override = 0 # Create value so manual control always takes preference
self.next_time = time.time() # Next time for motor step

# Set up the GPIO channel and initialise GPIO pins:
GPIO.setmode(GPIO.BOARD)
GPIO.setup(self.ENABLE, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(self.M0, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(self.M1, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(self.M2, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(self.STEP, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(self.DIR, GPIO.OUT, initial=GPIO.LOW)

self.set_mode(6) # Set initial motor mode to 1/32 step mode
self.UpdateSpeed(0) # Ensure initial motor speed is set to 0
motor_thread = threading.Thread(target=self.run) # Create a thread to run the motor control loop
motor_thread.start() # Start motor control thread
```

This function selects the step mode for the motor:

```
def set_mode(self, mode):
    # Set mode pins:
    if mode == 1: # Full Step Mode
        GPIO.output(self.M0, GPIO.LOW)
        GPIO.output(self.M1, GPIO.LOW)
        GPIO.output(self.M2, GPIO.LOW)
    if mode == 2: # 1/2 Step Mode
        GPIO.output(self.M0, GPIO.HIGH)
        GPIO.output(self.M1, GPIO.LOW)
        GPIO.output(self.M2, GPIO.LOW)
    if mode == 3: # 1/4 Step Mode
        GPIO.output(self.M0, GPIO.LOW)
        GPIO.output(self.M1, GPIO.HIGH)
        GPIO.output(self.M2, GPIO.LOW)
    if mode == 4: # 1/8 Step Mode
        GPIO.output(self.M0, GPIO.HIGH)
        GPIO.output(self.M1, GPIO.HIGH)
        GPIO.output(self.M2, GPIO.LOW)
    if mode == 5: # 1/16 Step Mode
        GPIO.output(self.M0, GPIO.LOW)
        GPIO.output(self.M1, GPIO.LOW)
```

```
GPIO.output(self.M2, GPIO.HIGH)
if mode == 6: # 1/32 Step Mode
    GPIO.output(self.M0, GPIO.LOW)
    GPIO.output(self.M1, GPIO.HIGH)
    GPIO.output(self.M2, GPIO.HIGH)

# Update the motor control parameters based on the Speed value
def UpdateSpeed(self, value):
    self.Speed = float(value)

    # Change step mode for different speed values received:
    if abs(self.Speed) > 7:
        self.set_mode(4)
    elif abs(self.Speed) > 3:
        self.set_mode(5)
    else:
        self.set_mode(6)

    # No spin (stopped)
    if self.Speed == 0:
        self.toggle = False
        self.increment = 0
    else: # Is spinning
        self.toggle = True
        self.increment = (-0.00495*(abs(self.Speed))) + 0.02495

    # Set direction pin:
    if self.Speed > 0: # Look Left
        GPIO.output(self.DIR, GPIO.LOW)
    if self.Speed < 0: # Look Right
        GPIO.output(self.DIR, GPIO.HIGH)

# Set upper and lower limits for time increment
if self.increment != 0:
    if self.increment < 0.0002:
        self.increment = 0.0002
    if self.increment > 0.02:
        self.increment = 0.02
```

```

# Display current motor speed and time increment for stepper:
if self.Speed != 0:
    print("| Increment: {:.3f} | Speed: {:.3f} |".format(self.increment, self.Speed))

# This function handles the priority of the messages received
def Priority_Decoder(self, msg):
    if len(msg.data) > 0: # if a valid message is received
        if msg.data[2] != 0:
            if msg.data[2] == -1: # Disable motor if -1 is received
                self.override = -1
                GPIO.output(self.ENABLE, GPIO.LOW)
                self.UpdateSpeed(0)
            if msg.data[2] == 1: # Enable and set motor speed
                self.override = msg.data[1] # If the motor speed has been set to 0, the lower priority can
control motor
                GPIO.output(self.ENABLE, GPIO.HIGH)
                self.UpdateSpeed(msg.data[1])
            elif self.override == 0: # This message will only work if override has been cleared
                GPIO.output(self.ENABLE, GPIO.HIGH)
                self.UpdateSpeed(msg.data[1])

# Run the motor control loop
def run(self):
    if self.i % 2 == 0: # This occurs every second cycle
        self.next_time += self.increment
        GPIO.output(self.STEP, GPIO.HIGH) # Send high signal to step pin
    else: # This occurs every other cycle
        self.next_time += 0.00002 # set next time to complete square-wave pulse to stepper
        if self.toggle == True: # If stepper speed is not zero, this part will run
            GPIO.output(self.STEP, GPIO.LOW) # Send low signal to step pin
        else:
            self.i = 0 # Reset step counter if motor is not spinning
    self.i += 1 # Increment step counter

# Run motor timing thread:
if not self.done:
    threading.Timer(self.next_time - time.time(), self.run).start()

```

```
# Stop the motor and cleanup GPIO pins
def stop(self):
    self.done = True
    GPIO.output(self.ENABLE, GPIO.LOW)
    GPIO.cleanup()

# Check if this script is being run as the main program
if __name__ == '__main__':
    Horizontal_Stepper = Stepper() # Create an instance of Stepper class
    rospy.spin() # Run ROS processes
    Horizontal_Stepper.stop() # Stop stepper gracefully
```


Appendix K

Vertical Stepper Node Script

```
#!/usr/bin/env python

# This code controls the vertical stepper motor. The commands are received from the 'Slider_values' topic and
# interpreted.
# Depending on where the message came from, different behaviours are achieved.
# A value is sent along with the motor signals which indicate the priority of the message sent.
# If this value is -1 then the motor is stopped and no further commands will have an affect on the motor speed.
# The motor can only be enable by a value of 1 being received in its place.
# Control signals with a 1 attached take priority over signals with a 0 attached

# Import Libraries
import Jetson.GPIO as GPIO
import time
import threading
import cv2
import sys
import rospy
from std_msgs.msg import Float32MultiArray

class Stepper:
    def __init__(self):
        rospy.init_node('stepper_node', anonymous=True) # Initialise the ROS node
        rospy.Subscriber('Slider_values', Float32MultiArray, self.Priority_Decoder) # Subscribe to
        Slider_values topic

        # Define the GPIO pins for the stepper motor drivers
        self.ENABLE = 7
        self.M0 = 11
        self.M1 = 13
        self.M2 = 15
        self.STEP = 19
        self.DIR = 21

        self.Speed = 0 # Target speed for motor
        self.i = 0 # Step counter
        self.done = False # Flag to indicate if motor control is done
        self.toggle = False # Toggle to enable or disable motor stepping
        self.increment = 0 # Initial increment for motor steps
        self.override = 0 # Create value so manual control always takes preference
```

```
self.next_time = time.time() # Next time for motor step

# Set up the GPIO channel and initialise GPIO pins:
GPIO.setmode(GPIO.BOARD)
GPIO.setup(self.ENABLE, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(self.M0, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(self.M1, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(self.M2, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(self.STEP, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(self.DIR, GPIO.OUT, initial=GPIO.LOW)

self.set_mode(2) # Set initial motor mode to 1/32 step mode
self.UpdateSpeed(0) # Ensure initial motor speed is set to 0
motor_thread = threading.Thread(target=self.run) # Create a thread to run the motor control loop
motor_thread.start() # Start motor control thread

# This function selects the step mode for the motor:
def set_mode(self, mode):
    # Set mode pins:
    if mode == 1: # Full Step Mode
        GPIO.output(self.M0, GPIO.LOW)
        GPIO.output(self.M1, GPIO.LOW)
        GPIO.output(self.M2, GPIO.LOW)
    if mode == 2: # 1/2 Step Mode
        GPIO.output(self.M0, GPIO.HIGH)
        GPIO.output(self.M1, GPIO.LOW)
        GPIO.output(self.M2, GPIO.LOW)
    if mode == 3: # 1/4 Step Mode
        GPIO.output(self.M0, GPIO.LOW)
        GPIO.output(self.M1, GPIO.HIGH)
        GPIO.output(self.M2, GPIO.LOW)
    if mode == 4: # 1/8 Step Mode
        GPIO.output(self.M0, GPIO.HIGH)
        GPIO.output(self.M1, GPIO.HIGH)
        GPIO.output(self.M2, GPIO.LOW)
    if mode == 5: # 1/16 Step Mode
        GPIO.output(self.M0, GPIO.LOW)
        GPIO.output(self.M1, GPIO.LOW)
        GPIO.output(self.M2, GPIO.HIGH)
```

```
if mode == 6: # 1/32 Step Mode
    GPIO.output(self.M0, GPIO.LOW)
    GPIO.output(self.M1, GPIO.HIGH)
    GPIO.output(self.M2, GPIO.HIGH)

# Update the motor control parameters based on the Speed value
def UpdateSpeed(self, value):
    self.Speed = float(value)

# No spin (stopped)
if self.Speed == 0:
    self.toggle = False
    self.increment = 0
else: # Is spinning
    self.toggle = True
    self.increment = (-0.00495*(abs(self.Speed))) + 0.02495

# Set direction pin:
if self.Speed > 0: # Look Left
    GPIO.output(self.DIR, GPIO.LOW)
if self.Speed < 0: # Look Right
    GPIO.output(self.DIR, GPIO.HIGH)

# Set upper and lower limits for time increment
if self.increment != 0:
    if self.increment < 0.0002:
        self.increment = 0.0002
    if self.increment > 0.02:
        self.increment = 0.02

# Display current motor speed and time increment for stepper:
if self.Speed != 0:
    print("| Increment: {:.3f} | Speed: {:.3f} |".format(self.increment, self.Speed))

# This function handles the priority of the messages received
def Priority_Decoder(self, msg):
    if len(msg.data) > 0: # if a valid message is received
        if msg.data[2] != 0:
```

```

    if msg.data[2] == -1: # Disable motor if -1 is received
        self.override = -1
        GPIO.output(self.ENABLE, GPIO.LOW)
        self.UpdateSpeed(0)
    if msg.data[2] == 1: # Enable and set motor speed
        self.override = msg.data[0] # If the motor speed has been set to 0, the lower priority can
control motor
        GPIO.output(self.ENABLE, GPIO.HIGH)
        self.UpdateSpeed(msg.data[0])
    elif self.override == 0: # This message will only work if override has been cleared
        GPIO.output(self.ENABLE, GPIO.HIGH)
        self.UpdateSpeed(msg.data[0])

# Run the motor control loop
def run(self):
    if self.i % 2 == 0: # This occurs every second cycle
        self.next_time += self.increment
        GPIO.output(self.STEP, GPIO.HIGH) # Send high signal to step pin
    else: # This occurs every other cycle
        self.next_time += 0.00002 # set next time to complete square-wave pulse to stepper
        if self.toggle == True: # If stepper speed is not zero, this part will run
            GPIO.output(self.STEP, GPIO.LOW) # Send low signal to step pin
        else:
            self.i = 0 # Reset step counter if motor is not spinning
    self.i += 1 # Increment step counter

# Run motor timing thread:
if not self.done:
    threading.Timer(self.next_time - time.time(), self.run).start()

# Stop the motor and clean up GPIO pins
def stop(self):
    self.done = True
    GPIO.output(self.ENABLE, GPIO.LOW)
    GPIO.cleanup()

# Check if this script is being run as the main program

```

```
if __name__ == '__main__':  
    Vertical_Stepper = Stepper() # Create an instance of Stepper class  
    rospy.spin() # Run ROS processes  
    Vertical_Stepper.stop() # Stop stepper gracefully
```