ABSOLUTE CALIBRATION OF A SOLAR TRACKING PROBE

By

Sean Halpin-Jordan

Supervised by

Dr. Marcin Gradziel

A thesis submitted in partial fulfilment for the
degree of Bachelor Of Science

Maynooth University
National University of Ireland Maynooth

Faculty of Science and Engineering
Experimental Physics Department

April 2024

Abstract

An excellent renewable energy source that provides a long-term replacement for fossil fuels is solar power. The efficiency and cost-effectiveness of solar power are rising along with technology, making it a feasible option for supplying the world's energy demands.

The goal of this paper is to present the absolute calibration of a solar tracking device – the “Solar Hedgehog”. The solar hedgehog has been designed and manufactured by a previous group, it contains a hemispherical array of intensity-controlled LED’s that read in light intensity that is directed to it, which is used to track the sun’s path. An ADALM2000 data acquisition module is used to analyse the information. The focus on this project lies on its absolute calibration, laboratory field testing, and development of field alignment and data stream acquisition and processing routines.

Acknowledgements

I would like to thank my supervisor, Dr. Marcin Gradziel, for his amazing and tireless support and guidance through the course of this project, his profound enthusiasm and drive throughout the semester was inspiring and aided my own motivation towards the project. I could not have undertaken this journey without his provided knowledge and expertise.

Next, I would like to express gratitude to the head of the department, Dr. Creidhe O' Sullivan for her invaluable feedback and approachability in times of need.

I would also like to thank my lab partner, Craig, for his continued support and teamwork throughout the project.

Lastly, I am grateful for my family. Their belief in me was a huge driving factor throughout my course and fuelled my motivation profoundly.

Table of Contents

List of Figures

List of Tables

Introduction

1.1 Aims for this project

The aim of this project is to complete an absolute calibration of the solar tracking device – the "Solar hedgehog". To complete this the following are to be conducted.

- Testing components of tracking device.
- Design and construction of tracking device holder.
- Installing machines to move and rotate the holder for calibration.
- Implementing computational methods to control and automate the calibration.
- Perform a laboratory field test of the tracking device.

1.2 Motivation for this project

Solar power is a renewable energy source that is gaining worldwide popularity at extreme rates. This growth in industry will always require technological advancements to accommodate its needs. The development of solar energy technologies has become a priority for many countries across the globe, to reduce the reliance on non-renewable sources. The upgrading of solar panel installation will only benefit this.

This report will explore and derive analysis of light intensity consumption and how the movement of light detectors can benefit solar efficiency.

1.3 Solar tracking device – the "Solar Hedgehog" description

The "Solar Hedgehog" was designed by a previous group. The device consists of 43 LED's (Light - emitting diodes) in a hemispherical setup giving it a hedgehog like appearance. The middle support of the device houses the 7 interconnected multiplex boards which aid the device in scanning through each LED. An ADALM2000 data acquisition module is used to manage the data from each individual detector as well as computational

programming in python to sort through the data and display it graphically for easier and more in-depth visualisation.

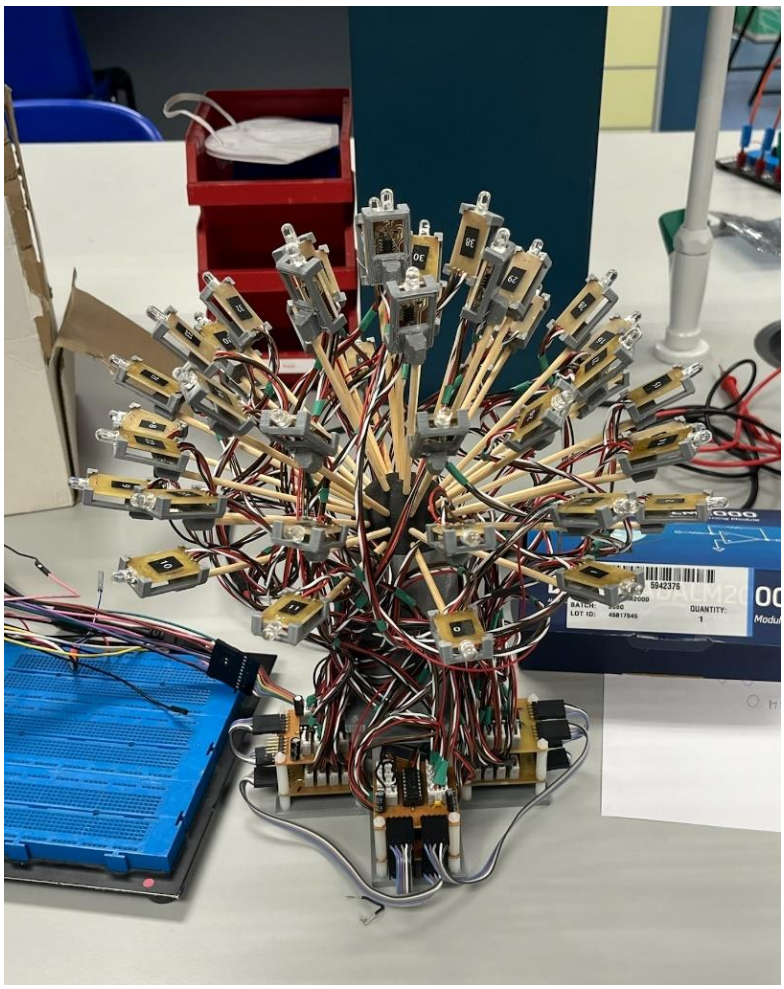

Figure 1.1: Solar tracking device - The "Solar Hedgehog”

The setup of the LED’s in the above tracking device can be seen in the figure below. The LED’s are numbered form 0 to 43 and are all connected to the central multiplexing hub. LED 0 to 11 are found in the bottom circular ring and the numbering carries upwards up to LED 42 which is located at the top alone in the centre.

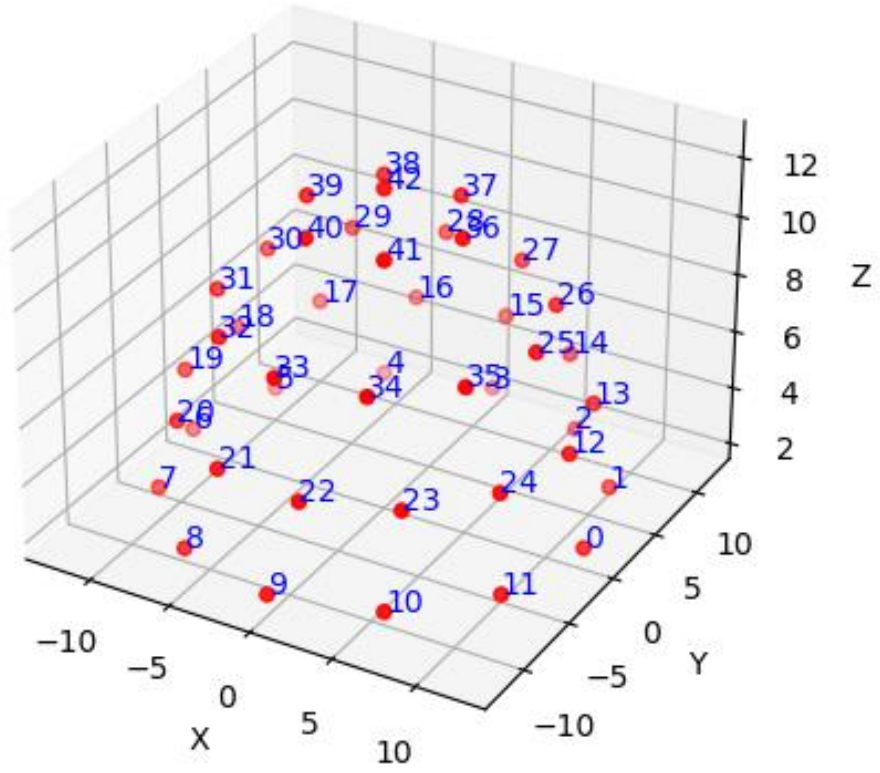

Figure 1.2: 3D plot of LED points with respective numbering

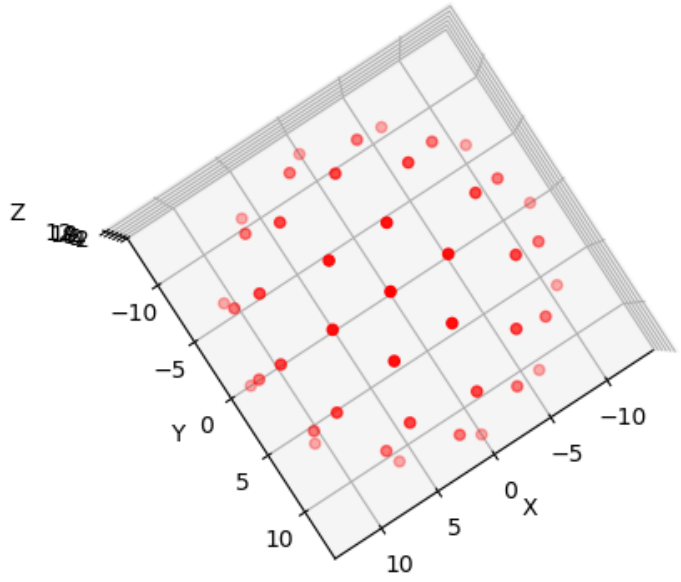

Figure 1.3: Top-down view of LED points in 3D plot

Chapter 2.
Theory

2.1 Solar energy collection

Solar energy collection refers to the harnessing of sunlight and converting it into usable energy. Photovoltaic (PV) solar panels use the sun's power to create a flow of electricity. This is the most widely adopted method of harvesting solar energy today. These panels, which range in size from a few square centimetres to a few square meters, are constructed from many PV cells arranged in an intricate matrix. Intuitively, the larger the surface area available for sunlight to penetrate the PV cells, the more solar energy that gets harvested [1] .

A solar collector is a device that collects and/or concentrates solar radiation from the Sun. These collectors play a crucial role in various applications, including:

- Solar Ovens: Before photovoltaic (PV) cells were used to directly convert sunlight into electricity, solar collectors were employed to cook food. As early as 1767, the naturalist and physicist Horace de Saussure created a solar oven that could reach temperatures up to 230°F (110°C). Solar ovens are still used globally as a practical way to cook food without electricity or combustion. By using the Sun’s heat, they reduce carbon emissions and indoor air pollution. Additionally, replacing wood with solar ovens helps prevent deforestation, as a single solar cooker can save a ton of wood per year [2].

- Residential Electricity Generation: Solar panels (which use photovoltaic cells) are commonly used in residential homes and commercial buildings to generate electricity from sunlight [3].

2.2 Solar Geometry

Solar geometry is the measurement of the angle of the sun to the earth and the corresponding amount of solar energy hitting a given object or surface [4] . This can be described by two angles, the azimuth angle and the elevation angle.

The azimuth angle is the angle between North, measured clockwise around the observer's horizon, and the sun. it determines the direction of the celestial body. For example, a celestial body due North has an azimuth of 0º, one due East 90º, one due South 180º and one due West 270º.

The elevation is the vertical angular distance between a celestial body (sun, moon) and the observer's local horizon or, also called, the observer's local plane.

For us, the elevation of the sun is the angle between the direction of the geometric centre of sun's apparent disk and the observer's local horizon [5] .

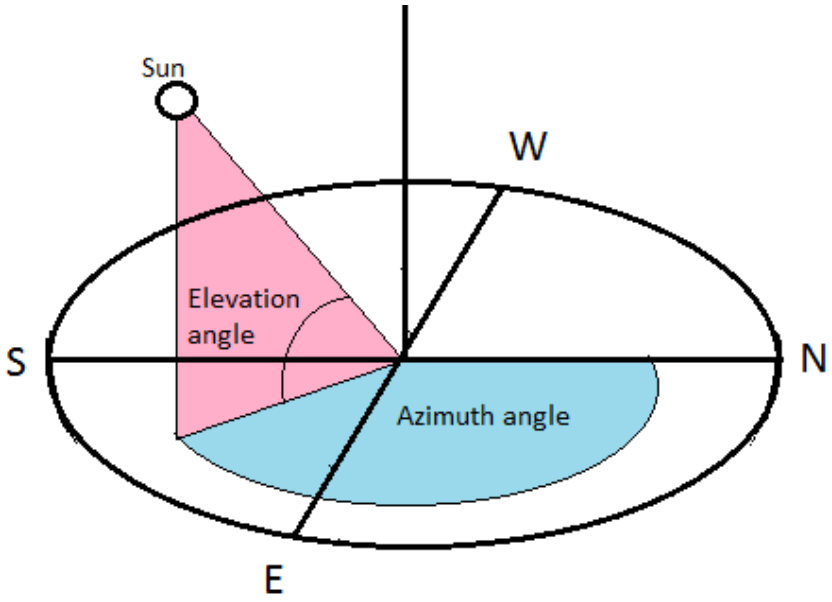

Figure 2.1: Solar geometry diagram [6]

2.3 Solar tracking

Solar tracking is a feature that enables solar panels or collectors to adjust their position to follow the sun's path during the day. The goal is to optimize sunlight exposure for improved efficiency and energy production.

There are two types of solar trackers: single-axis trackers and dual-axis trackers,

- A single-axis solar tracker allows the movement of the photovoltaic panels in one direction, from east to west, following the sun's path from sunrise to sunset. This effective function allows a significant increase in the collection of solar energy throughout the day.

- A dual-axis solar tracker moves in two directions, both from east to west and from north to south, which allows a very accurate tracking of the sun throughout the year. This capacity to adapt to the seasons of the year optimizes solar energy production, ensuring a constant and reliable performance in any climate [7] . BOOST

2.4 ADALM2000 acquisition module

Figure 2.2: The ADALM200 acquisition module

The ADALM2000 is a data acquisition module, it provides a range of features for data acquisition and signal processing, including:

Analog Inputs and Outputs: It offers two analog input channels and two analog output channels, allowing users to interface with analog signals from sensors, transducers, or other devices.

Digital Inputs and Outputs: It includes 16 digital input/output channels, which can be used for interfacing with digital devices or for digital signal processing tasks.

Portable and USB-powered: The module is compact and USB-powered, making it suitable for use in various environments, including laboratories, classrooms, and fieldwork [8] .

2.5 Zaber RST

A Zaber RST is product made by the Zaber company and is part of the RST family, it can be described as a motorized rotary stage.

A motorized rotary stage, also referred to as a rotation stage, is defined as a device that restricts motion to a single axis of rotation and precisely controls angular position about that axis of rotation [9] .

These stages are commonly used in automation, research, and industrial applications where precise linear motion is required. Zaber RST stages typically offer high resolution, repeatability, and a compact design, making them suitable for a wide range of positioning tasks. They are often controlled using computer software or programmable controllers to achieve precise and automated motion control.

Figure 2.3: Zaber RST [10]

2.6 CAD (Computer-Aided Design) & Onshape

CAD (computer-aided design) is the use of computer-based software to aid in design processes. CAD software is frequently used by different types of engineers and designers. CAD software can be used to create two-dimensional (2-D) drawings or three-dimensional (3-D) models [11] .

There are many software applications that are used to design and create objects with computer-aided design. In this project a software called Onshape was used.

Onshape is an easy-to-use CAD system that is useful for beginners. Onshape's built-in collaboration tools allow teams to effectively communicate and work on projects together effortlessly.

2.7 CENTENT CN0162 (Microstep driver)

The CENTENT CN0162 is a high-resolution step motor drive featuring four user selectable microstep resolutions. Available step rates range from full step to 256 microsteps. The CN0162 driver can deliver up to 1.5 million microsteps per second to the step motor. The drive is compact, measuring 4" x 4.75" x .85". Twenty-one different microstep resolutions are available in the CN0162 [12].

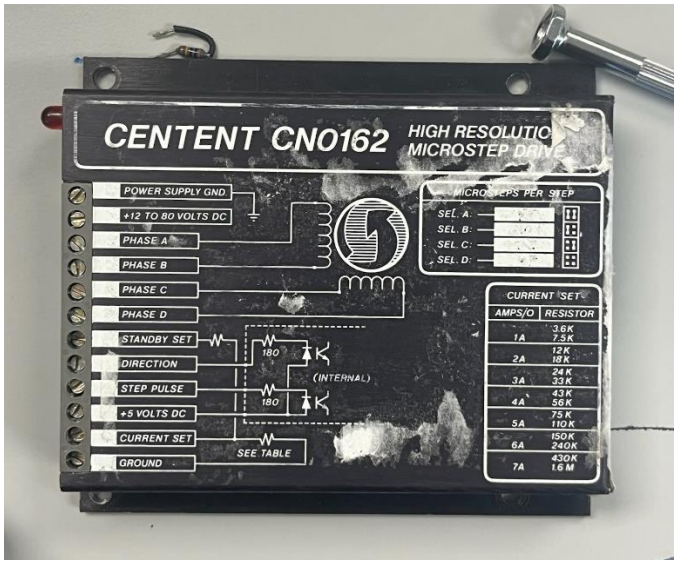

Figure 2.4: CENTENT CN0162 microstep driver

2.8 BSD Series Driver

The BSD series driver is a microstep stepping motor driver with an ultra-compact and optimized design to reduce space and cost, combined with Adaptive Microstepping technology ensuring noise and vibration suppression [13] .

Some of its technical features include:

- Range of current: 0.7-2.2 Amp. Setting up to four possible values by means of dip-switches.
- Microstepping: 400, 800, 1.600 and 3.200 steps/revolution. Setting by means of dip-switches.
- Management of the current profile setting by means of a dip-switch [9] .

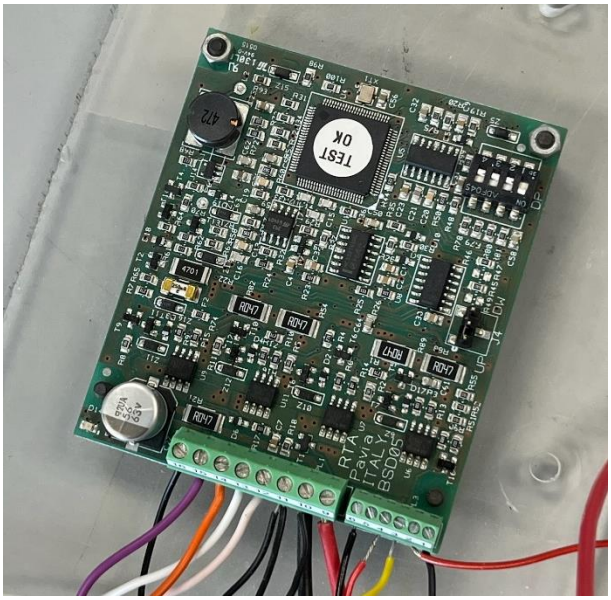

Figure 2.5: BSD series driver.

Chapter 3.

Experimental process

The first task to be filled was to test and calibrate components of the solar tracking device. As the device had been idle and stored away over the summer, it was decided that a test must be done on key components of the device to ensure all parts were still functional.

3.1 LED Testing

An experimental approach was undertaken to produce measurements for the field of view of a single LED on the solar tracking device. The apparatus was set up as shown below.

Figure 3.1: Experimental set up to test single LED.

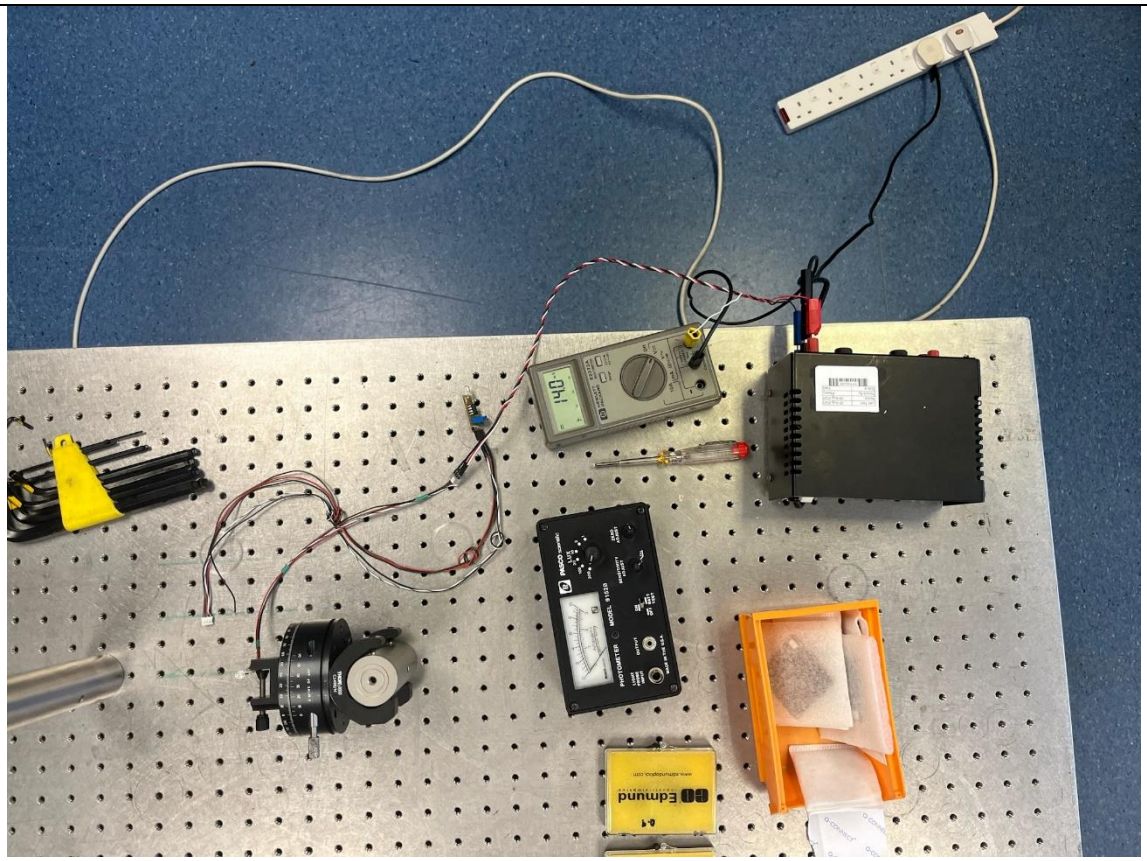

Figure 3.2: Close-up of experimental apparatus (before LED was placed in rotator).

The light source used was measured to be 8000 LUX, this was done using a photometer. A multimeter was connected in parallel to a 25V power supply and the rotating device that was used to rotate the LED on an axis.

Measurements began by placing the LED upright facing directly upwards toward the light source as shown below.

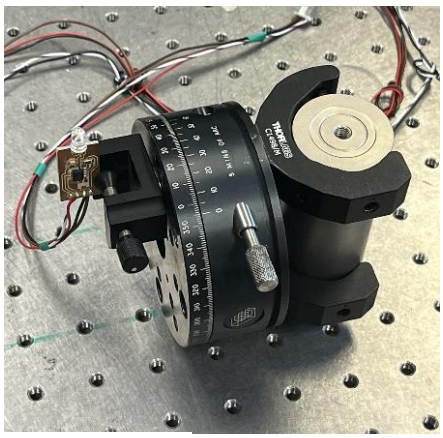

Figure 3.3 LED setup in rotating device.

The LED was moved in an anti-clockwise direction at increments of 5° and a recording of the voltage seen on the multimeter was taken. The respective data can be found in the results section.

This experimental procedure was then repeated with the LED starting in the same position as the last measurements, but this time it was moved in a clockwise direction. Recordings of the voltage were again taken after the LED had been rotated at increments of 5°.

Another version of this experiment was done that was aimed at yielding a more informative set of data that showed the absolute field of view of the LED better.

It was agreed that instead of setting the LED at an initial position directly facing the light source, it would be more beneficial to begin the measurements at an angle of -90° respective to the previous starting position and rotating the LED all the way to +90.

While the apparatus was set up, another set of measurements were taken with much smaller increments in angle for the LED rotation. This was done to give a more detailed and accurate depiction of the field of view a long a smaller range. The range was decided to be from -30° to +30°, as this was where the highest voltages were seen in previous recordings.

It was revised that the above experiments tested the field of view of the LED in only one line of symmetry, it was concluded that another experiment was to be completed where the setup of the LED in the rotating device was to be changed so that the PCB board was to be rotated -45° and the LED would be facing to the pole of the light source, this can be shown better in the figure below.

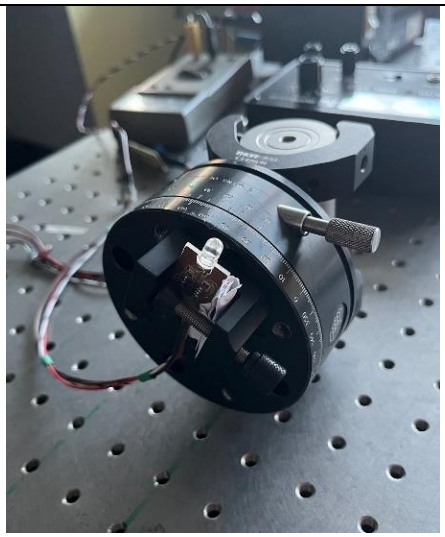

Figure 3.4: LED setup inside the rotating device.

The previous recordings where the LED was setup in a different configuration were then re-measured in the same way but for the new setup of the LED in the rotating device.

3.2 Computer-aided design process

The next step in the experimental process was to design and print an adequate holder for the device that will allow for field testing and calibration. The ideal holder would be able to bolt the solar tracking device down and move it in a swing-like motion to replicate the elevation angle. As well as this, it was devised that the holder be mounted to an external holder that would rotate 360 degrees along the ground to replicate the azimuth angle.

Onshape was chosen to support these ideas and create the 3D models. Before the project, my partner and I had no prior knowledge of computer-aided design, thus there was a lot of trial and error in creating the designs.

After an initial brainstorm, we decided to attempt to create a design without exact measurements to become familiar with the software. A mock model was created to show a general overview of the design and how the rotations could happen.

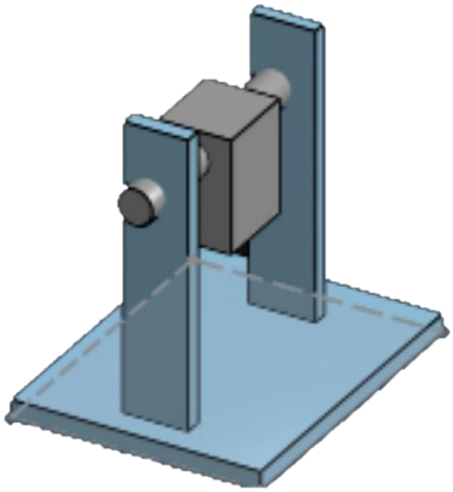

Figure 3.5: Initial mock model of solar tracker holder.

The idea was that the block in the middle would be changed to be a swing-like format where the solar tracking device would sit in the middle. The swing walls needed to be short enough so that none of the LED's on the tracking device would be blocked. We decided to create this model on onshape with accurate measurements, but we printed the model at a scale of 33% initially to serve as a mock structure and investigate any errors in the printing that may cause bigger problems if it were to be printed at full scale.

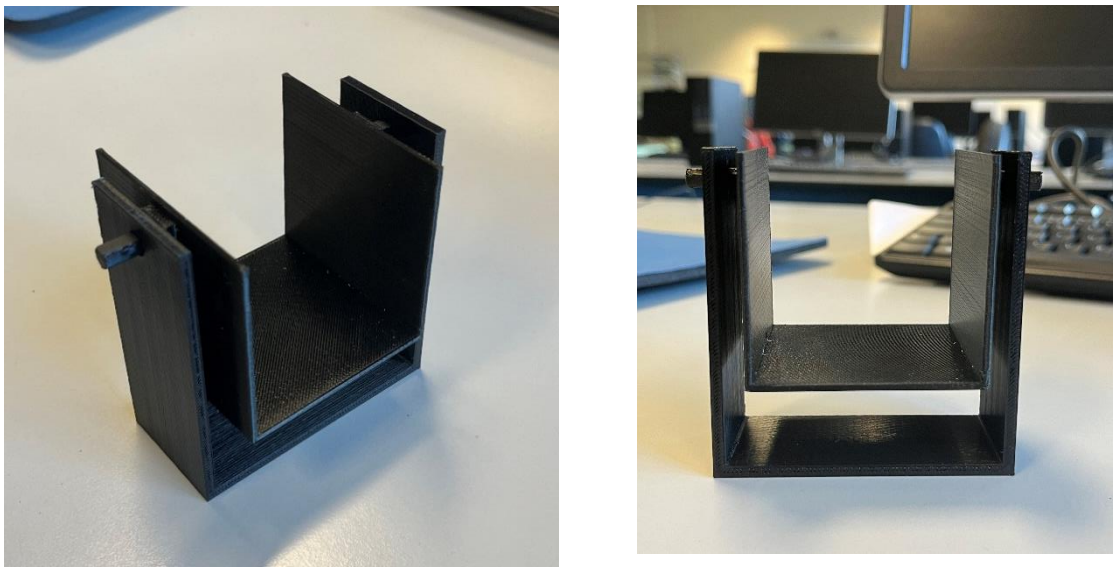

Figure 3.6: Printed mock design of solar tracking holder.

It was clear to see that this mock did not produce any errors when printing, although at full scale we agreed that there was a lot of unnecessary plastic being used, thus some time was spent on re-modifying the design to minimise material usage but still holding the core structure and weight that would hold the tracking device with a rigid base.

A mechanism was also created to provide a smooth rotation of the swing. A rod and rod key was designed and used to allow this. The motor that would be used to drive the swing would need to be connected to the rod key, to account for this a hollow cylinder was attached to the rod key, this hollow cylinder would house the motor cylinder. Measurements were taken to design this hollow cylinder so there was little to no gap between this and the motor cylinder. Super glue was injected in between these to mitigate any possibility of the parts disconnecting or slipping away. The rod would be connected to the swing and would slot into the rod key that is connected to the motor. This provided a seamless axis for rotation and little amount of design in the 3D printed model, thus lowering possibilities of error. Images of the rod and rod key can be seen in the figures below.

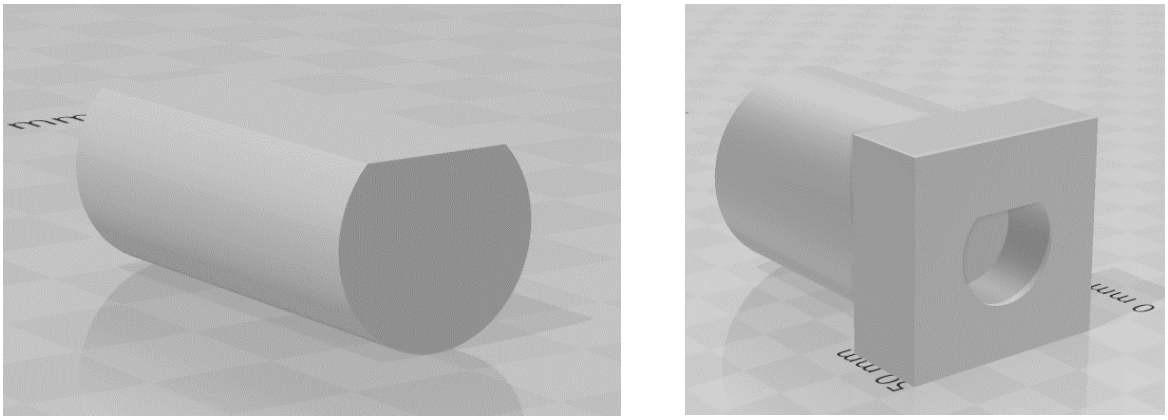

Figure 3.7: Rod and Rod key design for rotation of device.

The final design for the swing-like holder for the solar tracking device was completed. Material usage was a big factor when designing this, and anywhere plastic as not needed, it was removed. The key holder was mounted on both sides of this holder to provide as a connection between the swing holder and the outer holder that was used for the azimuth rotation. Holes were added at the base of the design to screw bolts to hold the tracking device in place on the holder.

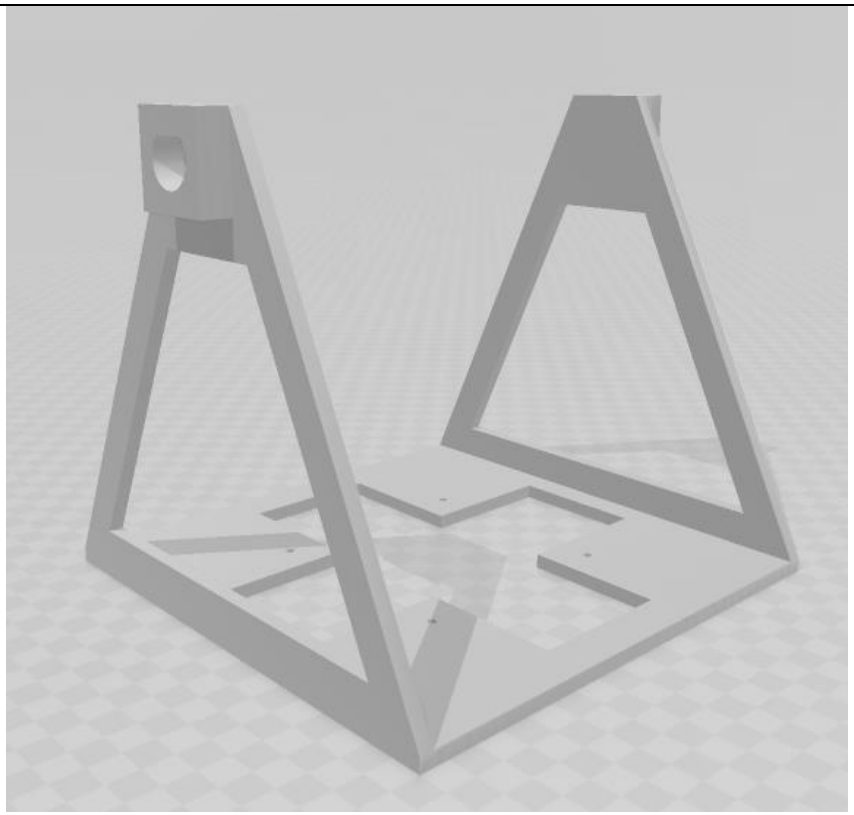

Figure 3.8: Swing-like holder design used to hold the solar tracking device.

From the figure it can be seen where material usage was prioritised. The swing walls were cut significantly by making the sides diagonal, also a large section in the centre of the walls was cut to minimise material usage.

The outer holder for the device, which would be connected to the swing holder loosely with the rods was also modified drastically from the initial mock model. The same approach taken with the swing holder of making the walls diagonal to mitigate material usage was applied to the outer holder. These walls were also made narrower but did not include a cut-out section in the centre as it was deemed to be too weak. Holes were added to the bottom in a square grid to allow the holder to be bolted to a rotating device that would rotate the entire solar tracking device as well as the holder to represent the azimuth rotation.

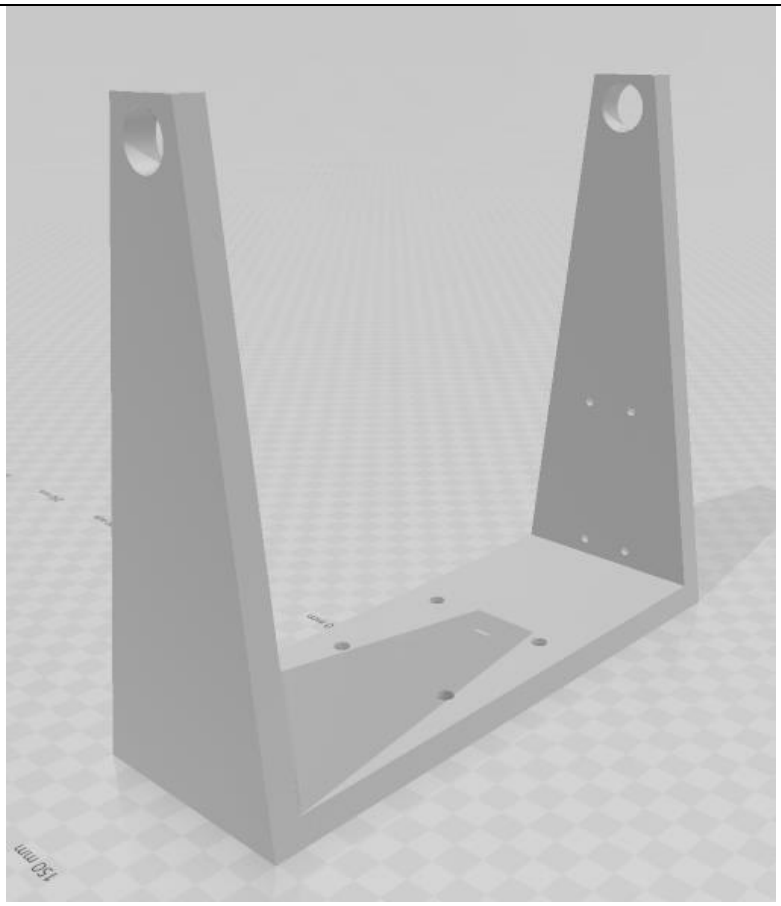

Figure 3.9: Design for outer holder of the solar tracking device.

The final item to be 3D printed was a support bracket that would connect the exterior wall of the outer holder. The purpose of this support bracket was to hold the motor in place, as it was found that the motor was displaced at a distance from the outer holder where it didn't receive enough support to be held up completely, thus, to reduce the sag of the motor a support bracket was designed. The sane principles as before where implemented, where a minimal amount of material should be used but enough to support the motor completely. It was devised that a flat plane would be connected entirely to the exterior wall of the outer holder and a table-like structure would extrude out of this that would hold the motor. Precise measurements were taken to ensure drill holes lined up the motor and the exterior holder wall.

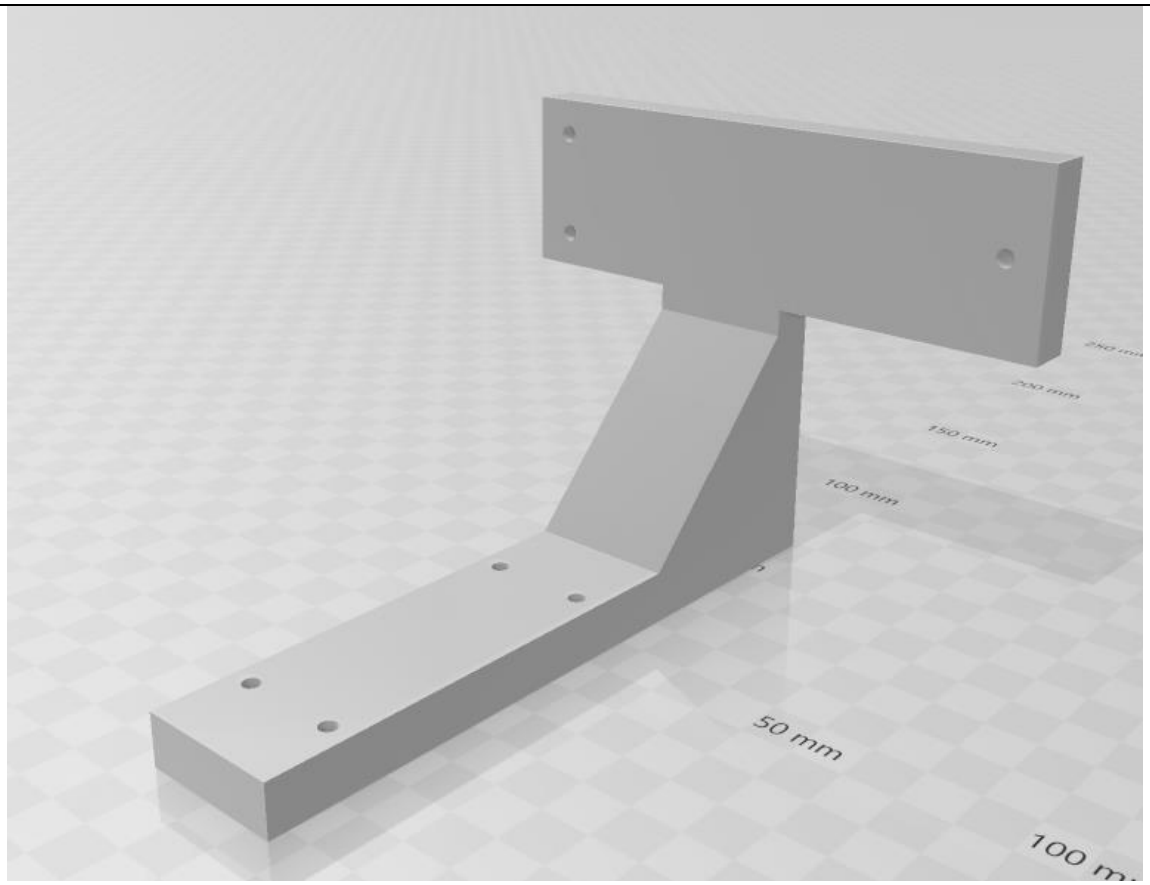

Figure 3.10: Design for the motor support bracket.

3.3 Motor driving and Microstepping.

A microstep driver called the CENTENT CN0162 shown in figure 2.4 was initially used to test the microstep in the motor. An apparatus was set up as shown below.

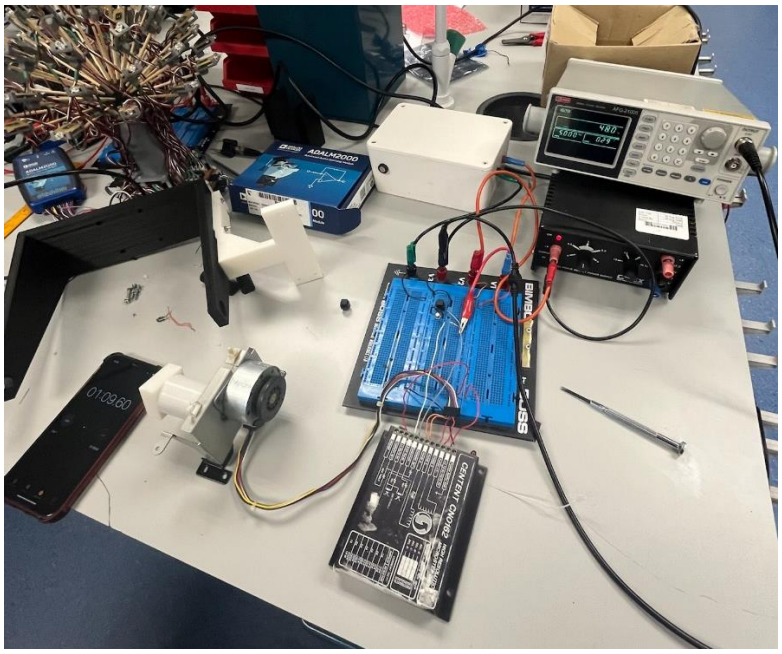

Figure 3.6: Apparatus to test the microstep of motor.

The function generator was set to 48 Hz to send pulses to the apparatus and drive the motor. This value was used as the drive angle of our motor was 7.5 degrees, thus 360 ÷ 7.5 = 48. The microstep resolutions were made by changing the jumpers on the microstep drive. Full rotation time measurements were taken for different configurations of jumper positions.

	Jumper Configuration	Time for full rotation (seconds)
A	Both	32
B	Left only	63
C	None	63
D	Right only	32

Table 1: Microstepping times for different jumper configurations.

It was decided that both jumpers would be used for the rest of the project as this yielded the most rigid rotation.

A full rotation time measurement was recorded without the transmission on the motor, this was measured to be 128 seconds. Thus the transmission ratio could be calculated by finding the ratio between the time for full rotation with the microstepping and time for full rotation without the transmission on the motor. The transmission ratio therefore was found to be 4:1.

After more testing and calibration it was found that the current microstep driver had some faults. The motor was creating too much heat when being used for a period of time, when the current form the driver was reduced, the heat would still generate. This became a problem as this build-up of heat would create problems for the final field test where the motor will be used for a long period of time to gather measurements.

A decision was made to scrap the previous motor drive as this problem was unable to be fixed. A BSD series stepping motor shown in figure 2.5, was brought in to replace the CENTENT CN0162. This new microstep driver also had jumpers to regulate the microstepping and a dip switch to regulate nominal current. This microstep driver produced a steady drive to the motor which mitigated shakiness and displacement of the swing holder when in motion with the solar tracking device.

3.4 Assembly process

All the needed parts to conduct a laboratory field test had been created and tested. The entire system was assembled to further calibrate the solar tracking device. All the 3D printed items were connected by the rod and rod key system. The motor support bracket was bolted to the exterior wall at its base and fixed to the motor from the bottom of the motor. The motor was connected to the system by the support bracket from below but also with the rod key in figure 3.8. The microstep driver was connected with its respective connectors to the motor and taped to the base of the motor. The ADALM200 module was taped to the other side of the exterior outer holder and the respective cables were connected to the solar tracking device. The holder of the system was bolted to the zaber RST device and thus the full system as assembled for field testing. The zaber RST and the ADALM2000 module were connected via USB to a PC for computational calibration, 25V power supplies were connected to these both also.

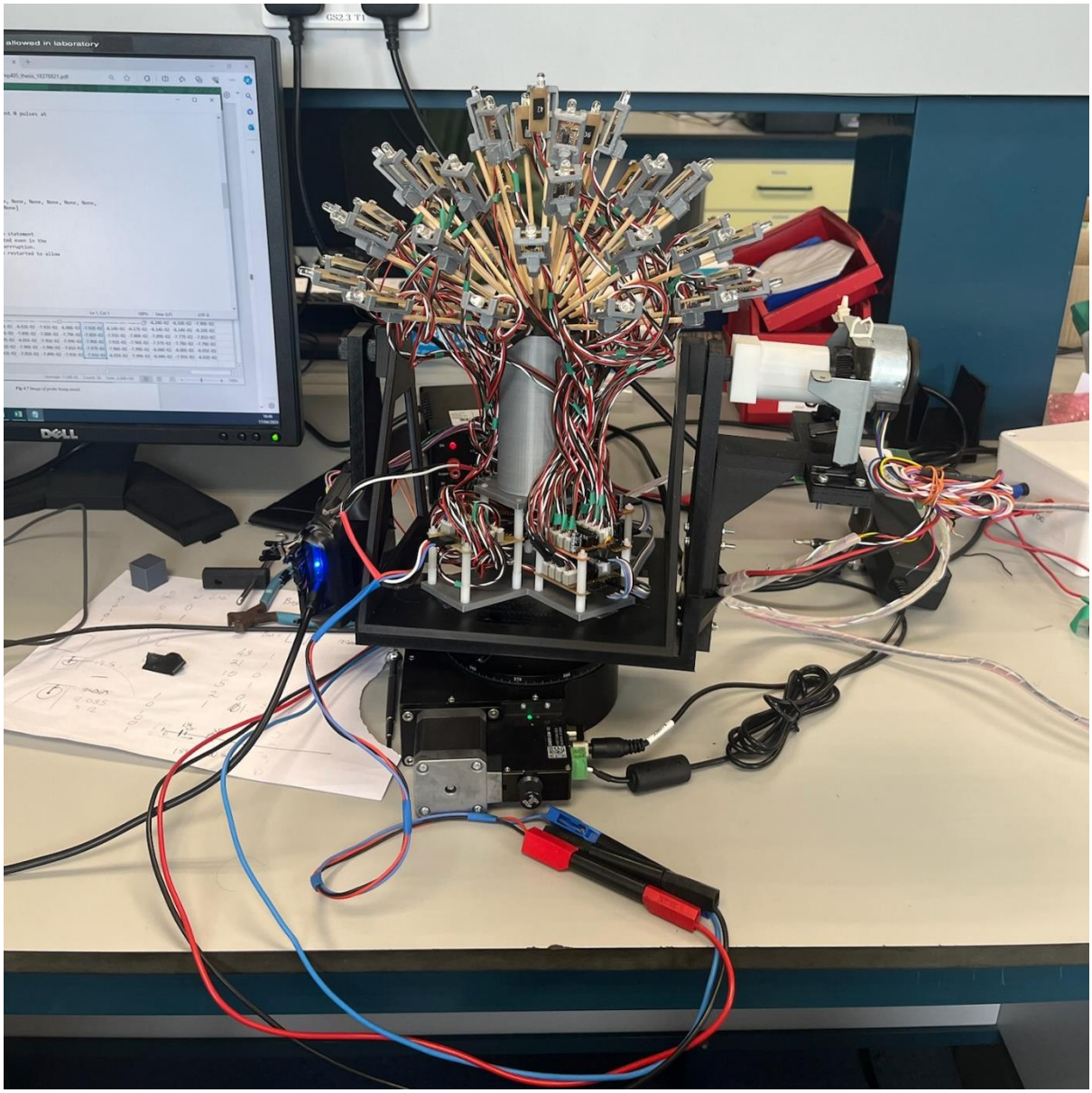

Figure 3.7: Assembled solar tracking device with holder and motor.

Chapter 4.

Results & Field testing

4.1 LED Testing

The LED was moved in an anti-clockwise direction at increments of 5° and a recording of the voltage seen on the multimeter was taken. The following table represents the recordings taken.

Anti-clockwise	
Angle (°)	Voltage (mV)
0	5820
5	2900
10	920
15	363
20	113.6
25	69
30	79.3
35	83.5
40	67.5
45	41.3
50	30.6
55	26.3
60	26.5
65	26.5
70	28.5
75	25.4
80	23.5
85	18.8
90	18.4

Table 2: Data for Anti-Clockwise LED field of view.

The results found looked reasonable as there was obvious decline in the voltage as the LED was directed away from the light source as expected. The table was analysed graphically also to gain a better understanding of the measurements.

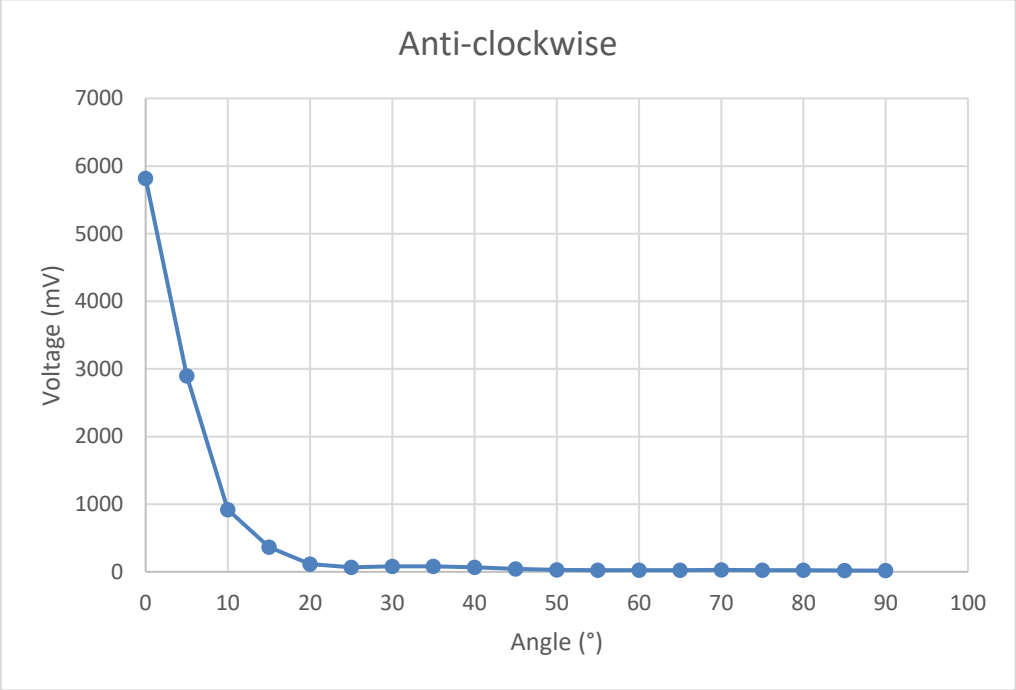

Figure 4.1: Graphic representation of anti-clockwise FOV data.

This experimental procedure was then repeated with the LED starting in the same position as the last measurements, but this time it was moved in a clockwise direction. Recordings of the voltage were again taken after the LED had been rotated at increments of 5°.

Clockwise			
Angle (°)	Voltage (mV)	Angle (°)	Voltage (mV)
0	5780	55	28.8
5	6180	60	28.9
10	4440	65	35.6
15	1748	70	40.9
20	726	75	43.8
25	371	80	43.6
30	113.6	85	36.7
35	45	90	29.6
40	37.7		
45	33.7		
50	30.7		

Table 3: Data of Clockwise LED field of view.

These measurements also were as expected, as an obvious decline in the voltage was again seen as the LED was angled further from the light source, the measured data was also analysed graphically.

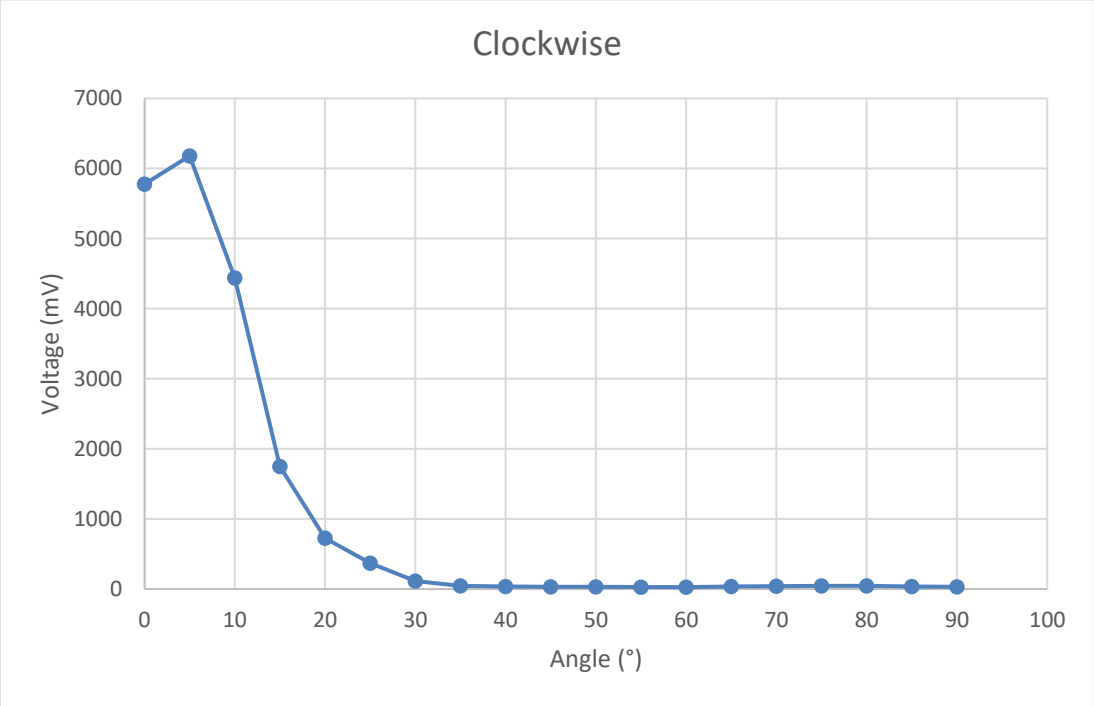

Figure 4.2: Graphic representation of clockwise field of view data.

It was concluded that the field of view of the LED was tested and analysed enough to show that the LED had not become faulty or generated any errors or offset while being idle. It was revised that the graphic plots above in figure 5.1 and figure 5.2 did not display a great representation of this, thus another version of this experiment was done that was aimed at yielding a more informative set of data that showed the absolute field of view of the LED better.

It was agreed that instead of setting the LED at an initial position directly facing the light source, it would be more beneficial to begin the measurements at an angle of -90° respective to the previous starting position and rotating the LED all the way to +90°. In this revised change, At 0° the LED is directly upright facing the light source, this was the initial position from the previous experiments. The apparatus was set up exactly the same as in figure 3.1 and figure 3.2. The LED was set up in the rotating device with the PCB board facing the light source pole, the same as in figure 3.3. The following data was recorded for the voltage seen on the multimeter and the respective angle.

Angle (°)	Voltage (mV)	Angle (°)	Voltage (mV)
-90	30.3	5	1906
-85	38	10	725
-80	44.9	15	345
-75	45.1	20	114.9
-70	42.3	25	72.6
-65	37.8	30	82.1
-60	31.3	35	85.4
-55	30.9	40	66.6
-50	32.9	45	42
-45	36.1	50	31.1
-40	41.4	55	27.8
-35	49.1	60	28.2
-30	99	65	30.8
-25	320	70	30.6
-20	660	75	27.7
-15	1450	80	23.8
-10	3430	85	20.9
-5	4980	90	20.8
0	4050		

Table 4: Revised field of view measurement data.

This revised data gave more insights into the field of view of the LED as when the data was analysed graphically it gave a clear understanding of the field of view of the LED as it could be seen at what angles was light read in from the source and where the most/least light was taken in form the LED. The data is represented graphically below.

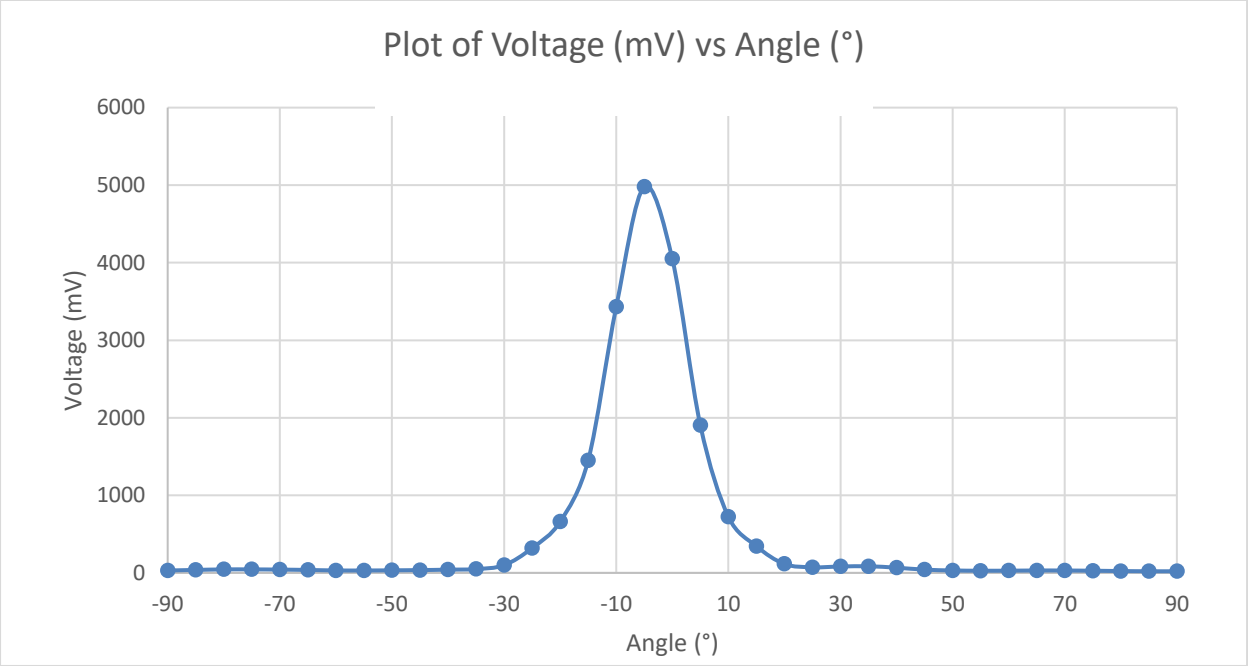

Figure 4.3: Graphic representation of revised field of view data.

While the apparatus was set up, another set of measurements were taken with much smaller increments in angle for the LED rotation. This was done to give a more detailed and accurate depiction of the field of view a long a smaller range. The range was decided to be from -30° to +30°, as this was where the highest voltages were seen in previous recordings. The resulting data is represented below.

Angle (°)	Voltage (mV)	Angle (°)	Voltage (mV)
-30	117.9	2	4140
-28	191.8	4	2980
-26	316.8	6	2030
-24	422	8	1250
-22	541	10	838
-20	705	12	574
-18	999	14	419
-16	1503	16	314
-14	2210	18	183
-12	3123	20	114
-10	4200	22	78.3
-8	5170	24	70.5
-6	5660	26	70.8
-4	5830	28	74.1
-2	5620	30	77.8
0	5030		

Table 5: Focused field of view data.

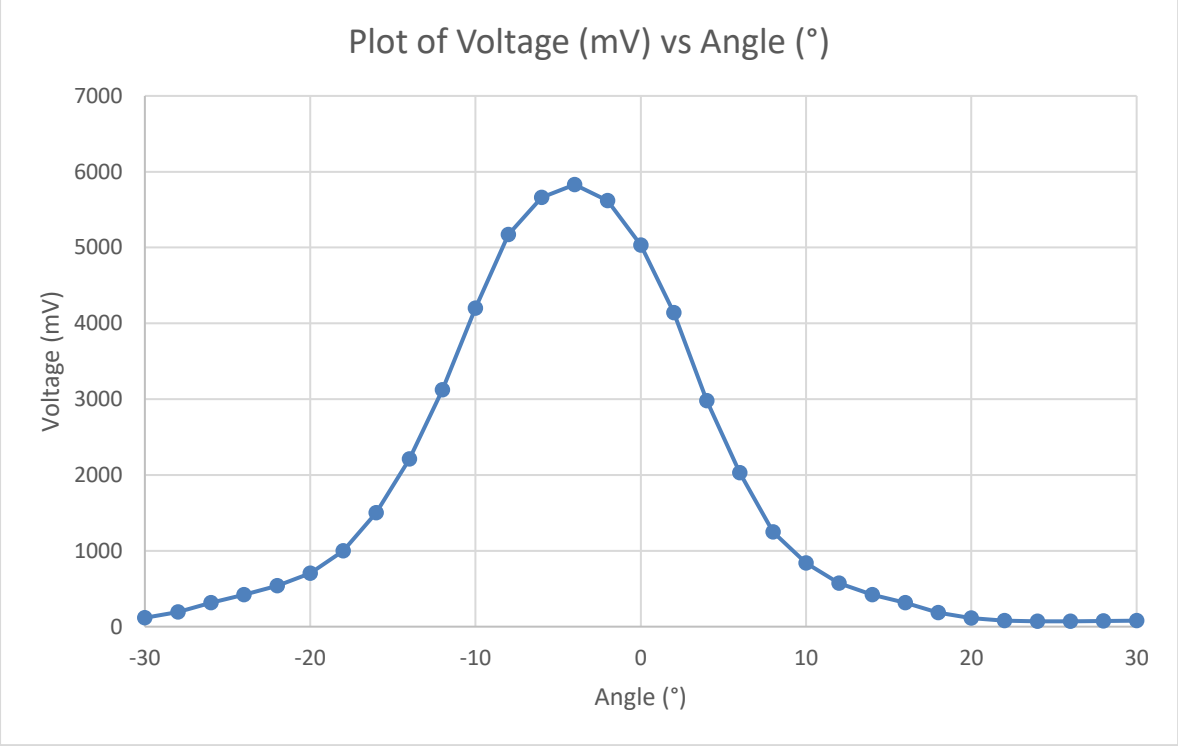

Figure 4.4: Graphic representation of focused data.

Measurements were taken for the fields of view of the LED along another axis of symmetry, the setup of the LED in the rotating device can be seen in figure 3.4.

This data is expected to be similar to the previous configuration of the LED in the device, but it must be completed to ensure the LED is not faulty in all lines of symmetry.

The recorded data is shown below in a table and represented graphically.

Angle (°)	Voltage (mV)	Angle (°)	Voltage (mV)
-90	27.6	5	3730
-85	32.5	10	1310
-80	46.1	15	551
-75	61.5	20	238.5
-70	58.5	25	125.1
-65	34.7	30	95.3
-60	24.9	35	57.3
-55	24.9	40	34.9
-50	27.1	45	28.2
-45	34.2	50	23.3
-40	68.9	55	20
-35	101	60	19.7
-30	80.2	65	25.2
-25	175.6	70	33.5
-20	624	75	34.2
-15	1950	80	28.8
-10	4630	85	21.5
-5	6330	90	17.2
0	6000		

Table 6: Field of view measurements (new symmetry axis) data.

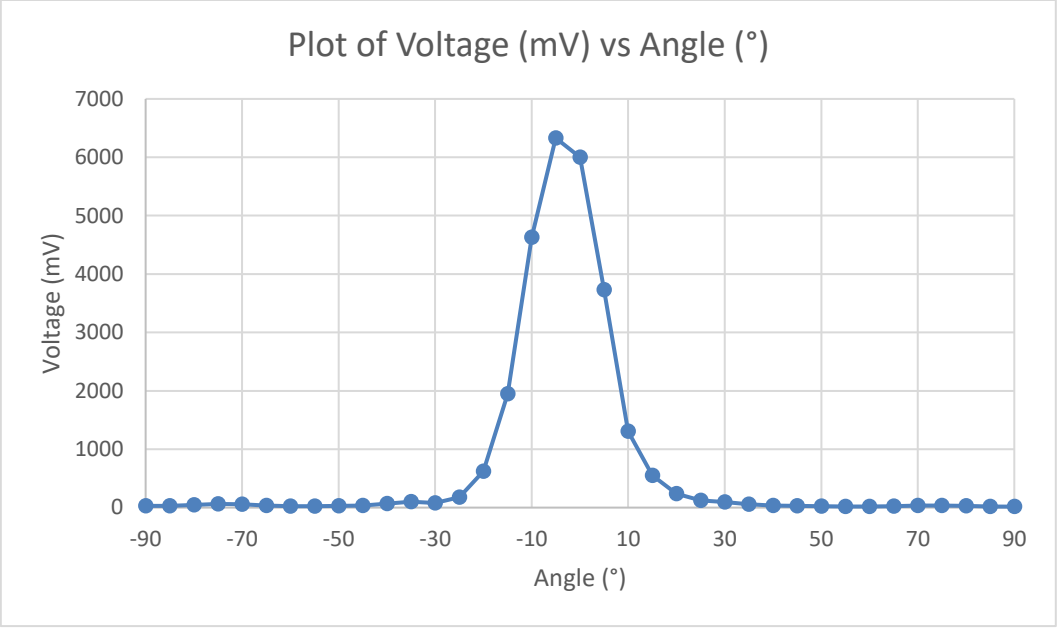

Figure 4.5: Graphic representation of new LED setup in rotating device.

More recordings were taken around the focus of the data at smaller increments to observe a more accurate region of results. This data is shown below.

Angle (°)	Voltage (mV)	Angle (°)	Voltage (mV)
-30	80.2	2	5300
-28	85	4	4260
-26	127.5	6	3130
-24	217.3	8	2090
-22	400	10	1350
-20	625	12	905
-18	984	14	639
-16	1502	16	480
-14	2370	18	341
-12	3490	20	246.5
-10	4770	22	170.2
-8	5670	24	133.6
-6	6210	26	116.6
-4	6330	28	102.2
-2	6220	30	96
0	5950		

Table 7: Focused field of view (new symmetry axis) data.

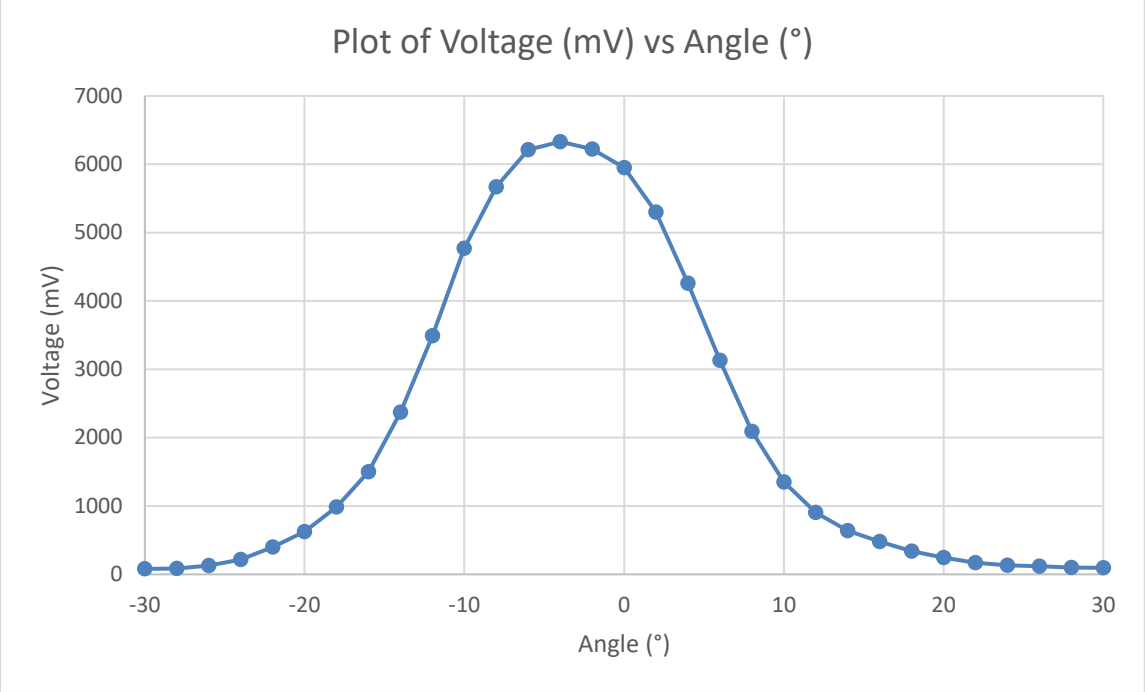

Figure 4.6: Graphic representation of focused data.

It was concluded that the field of view of the LED was responding correctly. The focused data provided more insights into the accuracy of the LED and generated a smoother more expected curve on the plot.

4.2 Live LED testing with python implementation

A python program was created to generate a live data plot of all the channel values and their respective voltage reading in a live field. This program can be seen in the appendix (figure 6.1). The program defines signals to be outputted through the digital pins, these signals can be constant, values, or pulses with specific parameters, it sets up triggering for the analog input channel and continuously acquires samples from the analog input channel and plots the data.

This program was run with the assembled system in figure 3.13. The test was done with no designated light source and no measures were made to prevent light pollution, it was a complete calibration test of the standard lab surroundings. A snapshot of the live plot can be seen below.

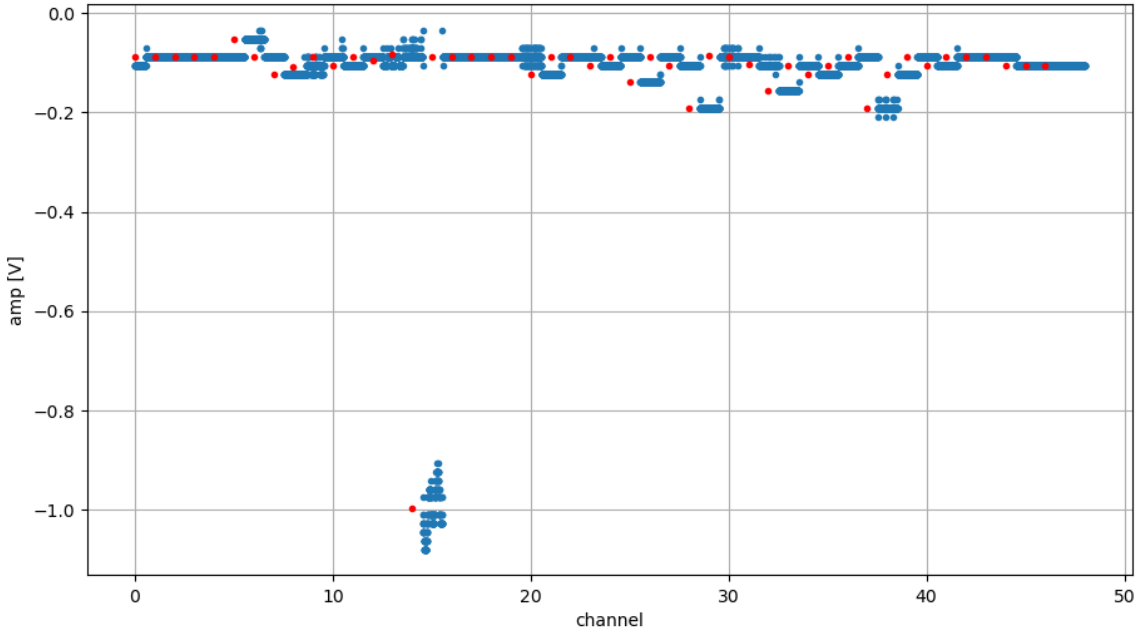

Figure 4.7: Snapshot of live data plot

Analysing the plot above, it can be seen the solar tracking device picked up a small amount of light intensity, but there does seem to be a small peak of around -1V at channel 14. The red dots in the plot represent the mean voltage collected for each channel, and the blue dots are the full samples received from the solar tracking device. It was found that the peak in channel 14 was seen as LED 14 was pointing towards the window in the lab, thus picking up light intensity from the sun in the sky. The other small peaks in the plot found at different channel values were from other sources of light pollution in the lab such as ceiling lights and reflections of lights found on surfaces that would have originated from the sun/ceiling lights.

4.3 Analysing waveform images

Scopy was used for lab testing with the multiplexing configuration. The pattern generator function was used to switch between inputs on the multiplexing board. The pattern generator creates a digital waveform that represents the bits changing for each input of each multiplexing, the waveform created was given gaps of 25ms. The resulting waveform image can be found in the appendix below (figure 6.4).

An apparatus was setup as shown below to analyse waveforms of different pins of the ADALM2000 module using a digital oscilloscope.

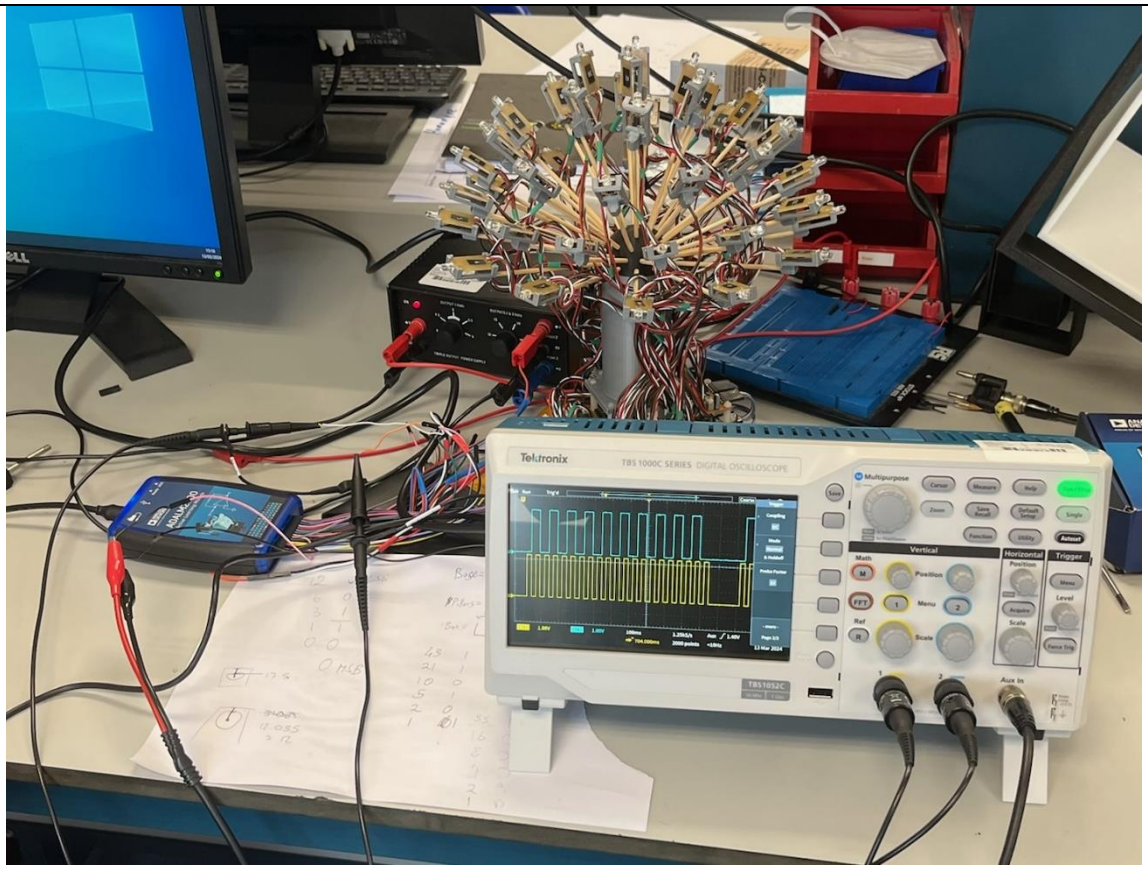

Figure 4.8: Apparatus used to test pins on the ADALM200

The resulting waveforms for individual pins were recorded using the Tektronix software, the waveform images for pins 0 and 1, 0 and 2, 0 and 3, 0and 4, 0 and 5, 0 and 6 and finally 0 and 7 were analysed, these images can be seen below.

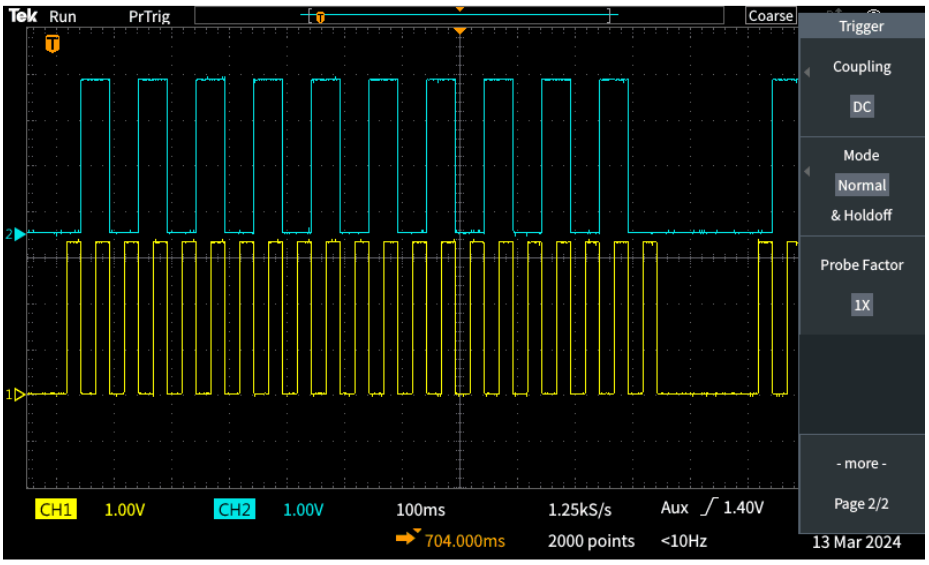

Figure 4.9: Waveform image of pulses from pins 0 – 1.

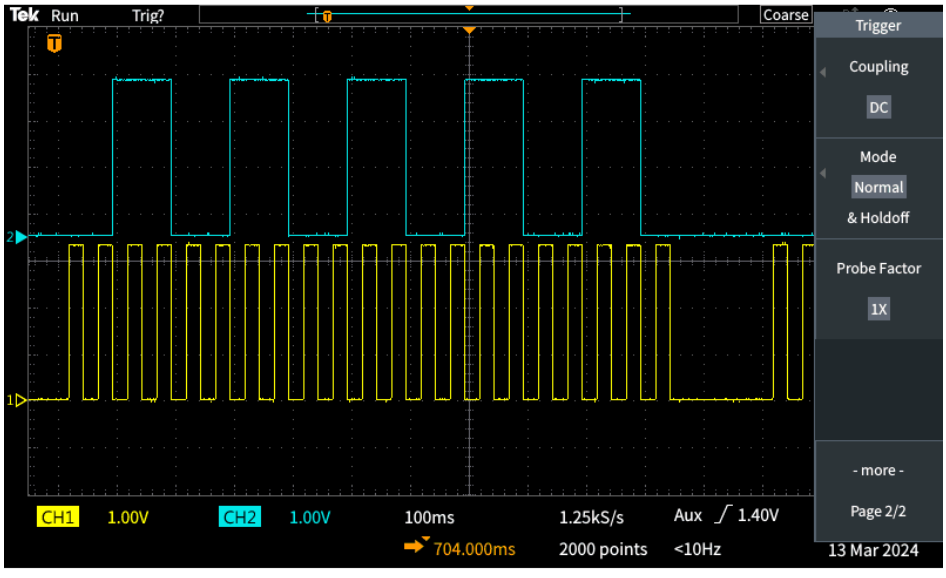

Figure 4.10: Waveform image of pulses from pins 0 – 2.

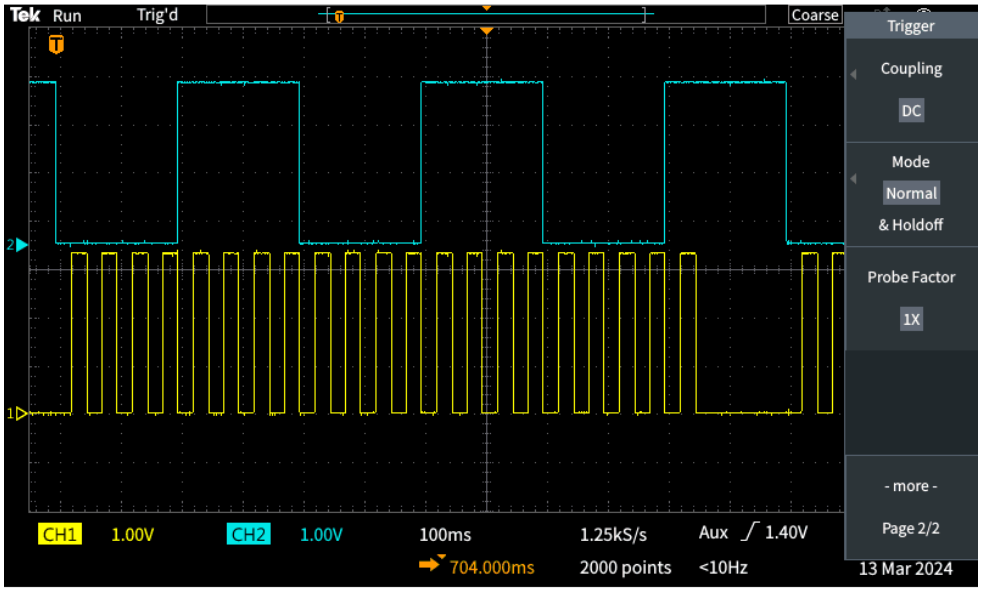

Figure 4.11: Waveform image of pulses from pins 0 – 3.

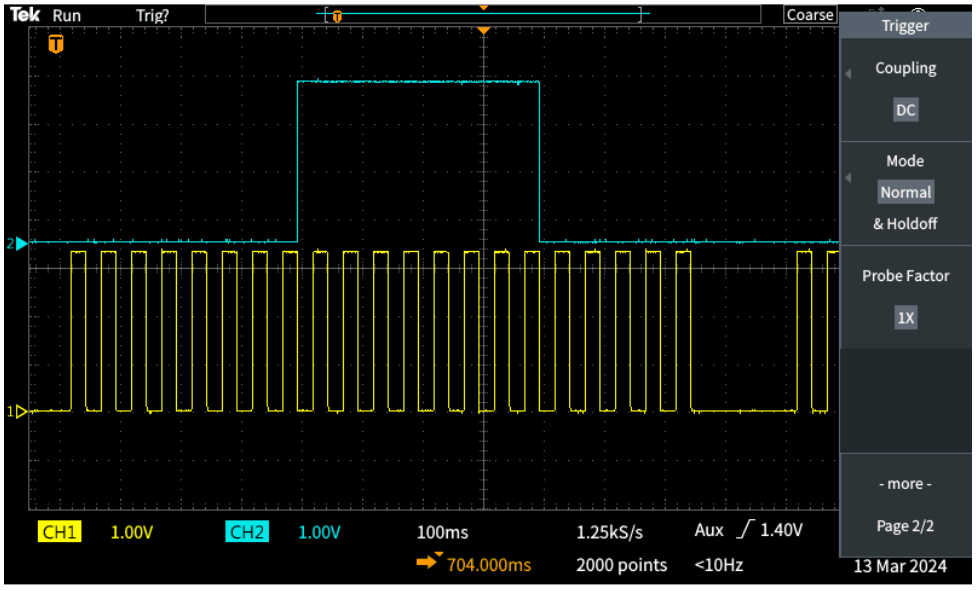

Figure 4.12: Waveform image of pulses from pins 0 – 4.

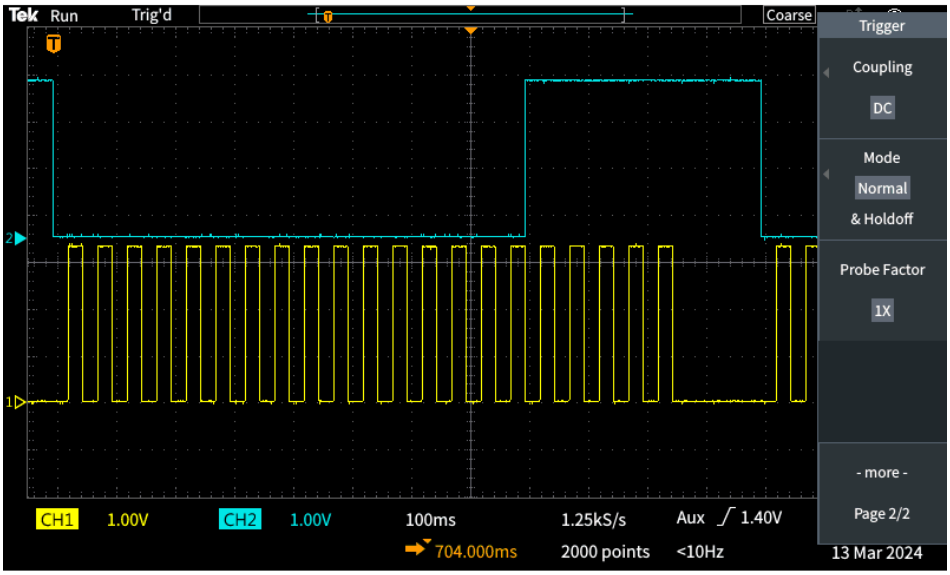

Figure 4.13: Waveform image of pulses from pins 0 – 5.

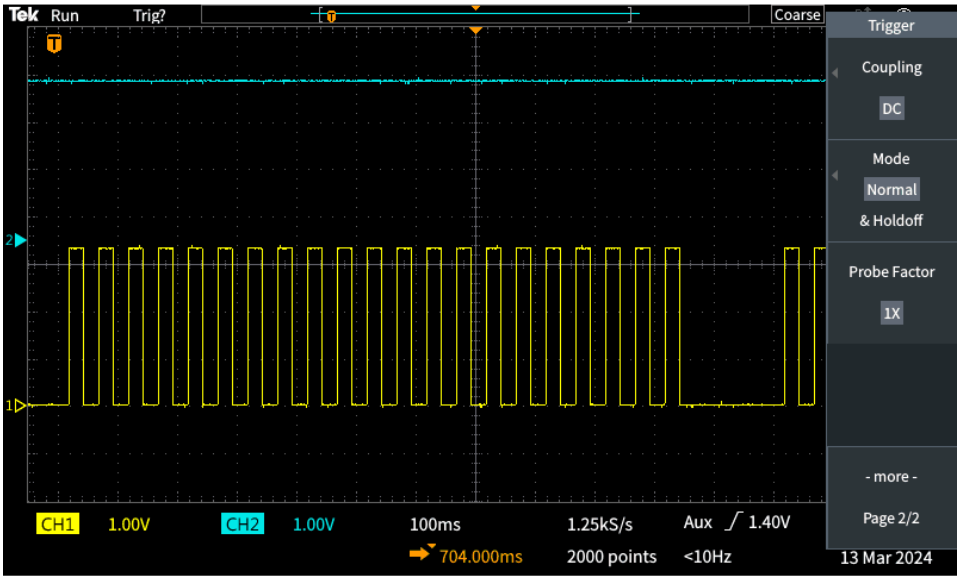

Figure 4.14: Waveform image of pulses from pins 0 – 6.

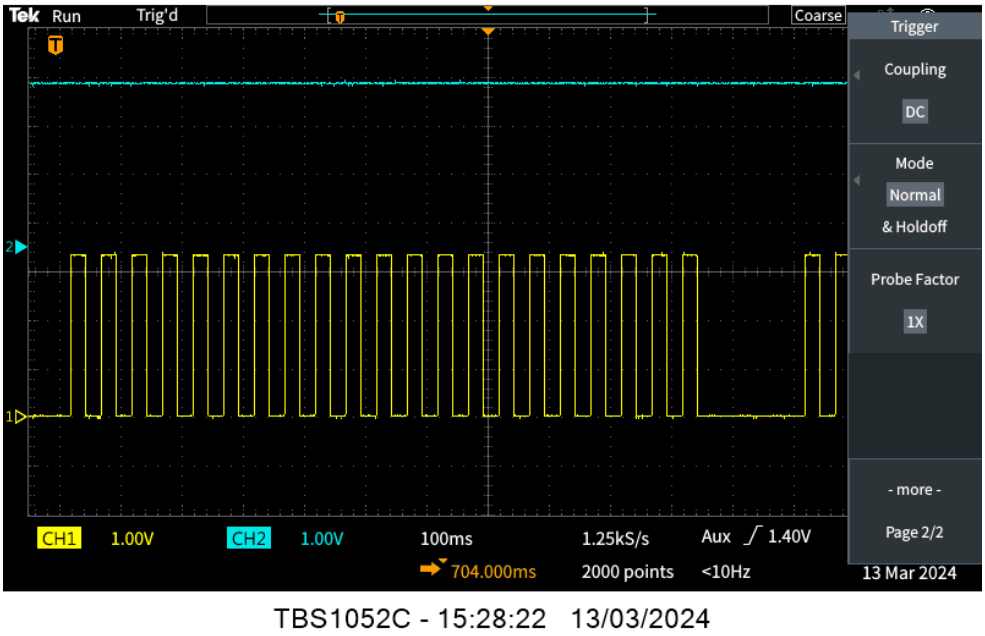

Figure 4.15: Waveform image of pulses from pins 0 – 7.

The waveforms for the pulses above correlated correctly as expected with that of the waveforms that was found on scopy. The yellow waveforms seen in these graphs is the 0 pin, and the blue waveforms are the pins 0 to 7 for their respective graph.

4.4 Laboratory field test

A final laboratory field test was conducted to gain a real understanding of the solar tracking probe and its accuracy in a more controlled and less polluted environment. The full solar tracking system consisting of the solar tracking device, tracking holder, power supplies, microstep driver, zaber RST and ADALM200 module were moved to a more isolated room where there is a discrete amount of light pollution, this was done to collect the most accurate recordings possible. A fixed light source was suspended on a pole directly above the system to be used in the field test.

To maximise accuracy, all lights in the room had been turned off and the windows were covered with a black liner.

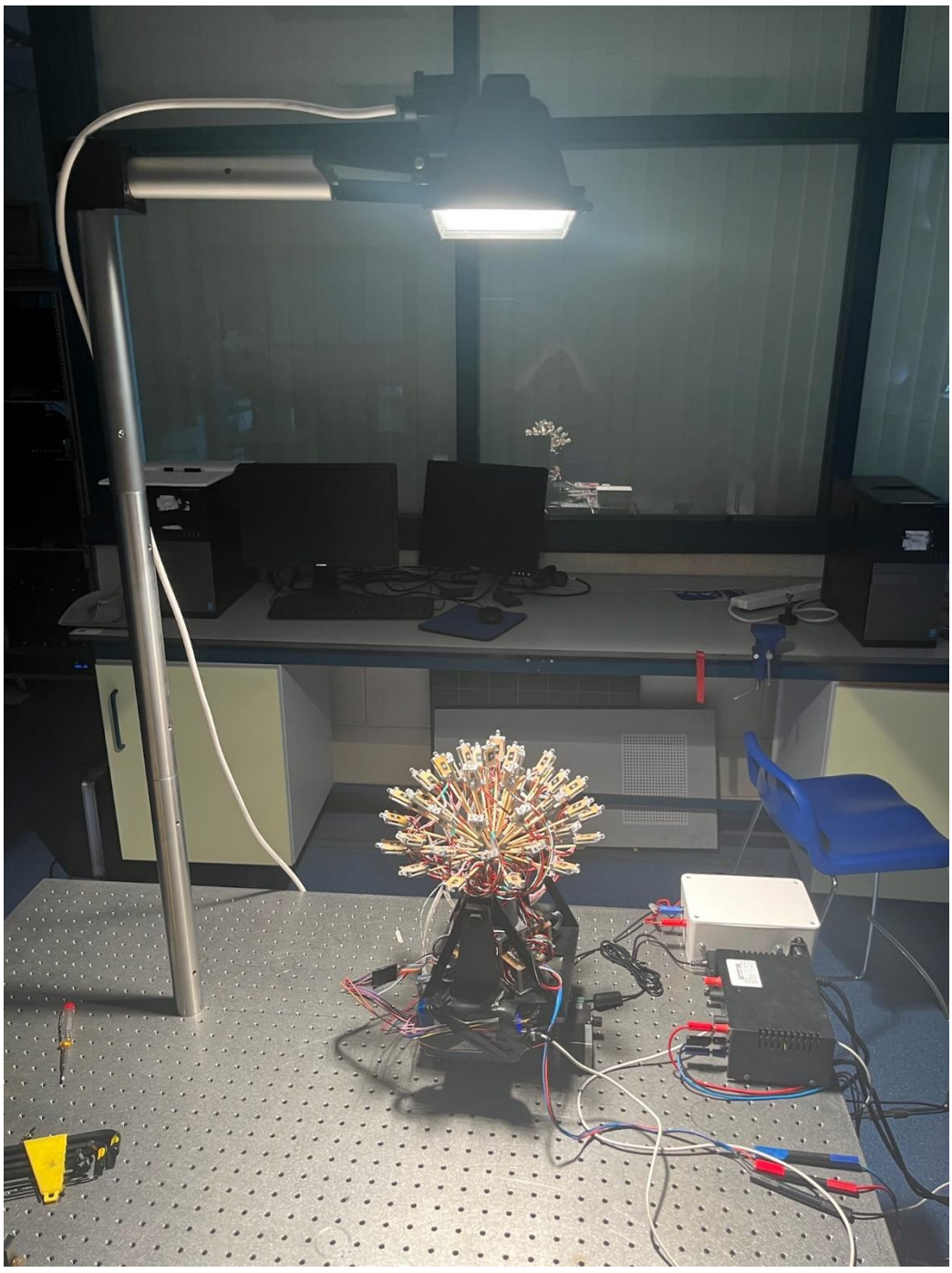

Figure 4.16: Experimental setup for laboratory field test.

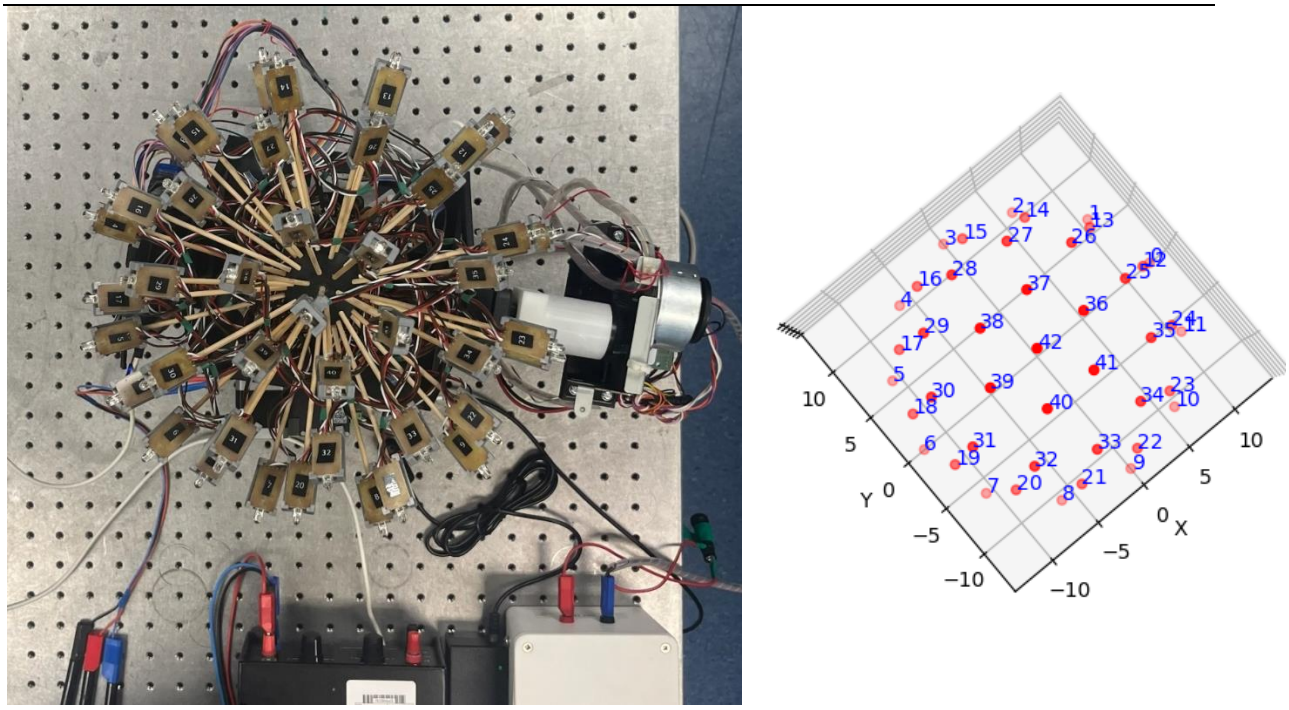

Figure 4.17: Top-down view of orientation of solar tracker during field test.

(Azimuth and Elevation angle = 0°)

To ensure accuracy in testing, a large amount of data was to be recorded. A program was created that is similar to the live plot code, this program instead implemented functions in the code that controlled the zaber RST to replicate the azimuth angle, and functions were also made to control the elevation angle of the device. This code can be seen in the appendix (figure 6.3).

For the field test, the azimuth angle was set to begin at 0° and move in increments of 3°. The range of the azimuth was set to move from 0° to 180°. Once the azimuth angle had changed, the elevation of the device would move. The elevation angle was set to move at increments of 10° and was ranged from -90° to 90°.

To picture this better, beginning at 0° azimuth, the elevation would start at -90° and move at increments of 10° until it got to +90°, then the azimuth would move 3° and the elevation sweep would repeat. At each increment of elevation and azimuth, a measurement of the light intensity on each LED was taken. As you could imagine, this created a large amount of data as there was 60 azimuth recordings, 1080 elevation recordings and 43 recordings of light intensity in the LED's. A snippet of some lines of data is shown below.

	(V)
Elevation	-90
Azimuth	0
0	-0.07
1	-0.07
2	-0.07
3	-0.07
4	-0.07
5	-0.09
6	0
7	1.34
8	0.04
9	-0.06
10	-0.03
11	-0.06
12	-0.06
13	-0.07
14	-0.05
15	-0.07
16	-0.07
17	-0.05
18	-0.05
19	0.09
20	0.03
21	-0.05
22	0
23	-0.07
24	-0.07
25	-0.07
26	-0.07
27	-0.07
28	-0.05
29	-0.07
30	-0.07
31	0
32	0.02
33	-0.05
34	-0.05
35	-0.07
36	-0.07
37	-0.05
38	-0.07
39	0
40	-0.05
41	-0.07
42	-0.05

	(V)
Elevation	0
Azimuth	0
0	0.02
1	0
2	0
3	0
4	-0.05
5	-0.07
6	-0.05
7	-0.05
8	-0.05
9	-0.05
10	-0.07
11	-0.05
12	0.03
13	-0.07
14	-0.05
15	0
16	0.03
17	0.02
18	0.02
19	0.02
20	0.02
21	-0.05
22	0
23	0
24	0
25	-0.05
26	-0.05
27	0.02
28	0.02
29	0
30	-0.05
31	0
32	0
33	-0.05
34	0
35	0
36	1.09
37	0.64
38	0.91
39	0.07
40	0.12
41	0.12
42	9.2

	(V)
Elevation	0
Azimuth	3
0	0.02
1	0
2	0
3	0
4	-0.05
5	-0.07
6	-0.05
7	-0.05
8	-0.05
9	-0.05
10	-0.07
11	-0.05
12	0.03
13	-0.07
14	-0.05
15	0
16	0.03
17	0.02
18	0.02
19	0.02
20	0.02
21	-0.06
22	0
23	0
24	-0.01
25	-0.05
26	-0.05
27	0.02
28	0.02
29	0
30	-0.05
31	0
32	0
33	-0.05
34	0
35	0
36	1.1
37	0.64
38	0.91
39	0.07
40	0.12
41	0.12
42	9.22

Table 8: Snippet of field test data.

	(V)
Elevation	20
Azimuth	15
0	0.05
1	0
2	0
3	0.03
4	-0.05
5	-0.09
6	-0.05
7	-0.05
8	-0.05
9	-0.07
10	-0.07
11	-0.05
12	0
13	-0.07
14	-0.04
15	0.02
16	0.03
17	0.02
18	-0.05
19	-0.05
20	-0.05
21	-0.07
22	-0.05
23	0
24	-0.05
25	0.03
26	0.12
27	0.8
28	0.03
29	0
30	-0.05
31	0.04
32	0
33	-0.05
34	0
35	0
36	1.53
37	5.47
38	0.07
39	0
40	0
41	0.07
42	0.03

	(V)
Elevation	-60
Azimuth	93
0	-0.05
1	-0.07
2	-0.07
3	-0.07
4	-0.07
5	-0.05
6	-0.05
7	0.19
8	0.06
9	-0.05
10	-0.05
11	-0.05
12	-0.05
13	-0.09
14	-0.07
15	-0.07
16	-0.07
17	0
18	0
19	0.23
20	6.52
21	0
22	0
23	0.02
24	0
25	-0.07
26	-0.07
27	-0.07
28	-0.01
29	0
30	-0.05
31	0.92
32	2.65
33	0.03
34	-0.05
35	0
36	-0.05
37	0.02
38	0
39	0.1
40	0.05
41	-0.05
42	-0.05

	(V)
Elevation	30
Azimuth	135
0	0
1	0
2	-0.01
3	0.02
4	-0.05
5	-0.1
6	-0.07
7	-0.05
8	-0.07
9	-0.07
10	-0.07
11	-0.05
12	0.02
13	-0.07
14	-0.01
15	0.03
16	0.03
17	0
18	-0.05
19	-0.05
20	-0.05
21	-0.07
22	-0.07
23	-0.05
24	-0.05
25	0.07
26	1.76
27	1.61
28	0.02
29	0.02
30	-0.05
31	-0.05
32	-0.06
33	-0.05
34	0
35	0
36	0.28
37	0.35
38	0.03
39	0.03
40	0
41	0
42	0.02

Table 9: More snippets of field test data.

The raw data file of all recordings taken in the ;abratory field test can be found in the appendix below (figure 6.5).

The yellow highlighted points in the data are the peak values found in each column. Referring to the setup of the LEDs on the solar tracking device in figure 1.2 it can be seen what LEDs should receive the peak amount of light at different configurations of azimuth and elevation. An easy configuration to calibrate would be when the elevation position is set to 0°. It can be expected that LED 43 would receive the most light in this case as it is pointing directly at the light source above. This can be seen to be true in Table 8, where the azimuth position is set to 0°, and the azimuth is set to 0 and 3°, the peak values of voltage was recorded to be 9.2V and 9.22V coming from LED 42. Upon further inspection of the full data list it is seen, that for every row of data at 0° azimuth, the peak value is seen in LED 42.

Other interesting points of data is seen in Table 9 where the azimuth and elevation angles are set to 15° and 20° respective. The peak value of voltage is seen to be 5.47 on LED 37. Referring to figure 1.2 again, it can be seen that when the device is rotated at these elevation and azimuth angles, LED 37 is pointing directly above at the fixed light source, again backing up the accuracy in these results.

Another python program was created to visualise these sets of data in a polar plot. The code can be modified to choose a row in the full data file and a polar plot of the LED's, and the respective values of voltage were changed to polar co-ordinates and plotted. The full code can be seen below in the appendix (figure 6.4). Some examples of these polar plots for different configurations of azimuth and elevation angles in the final laboratory field test are shown below.

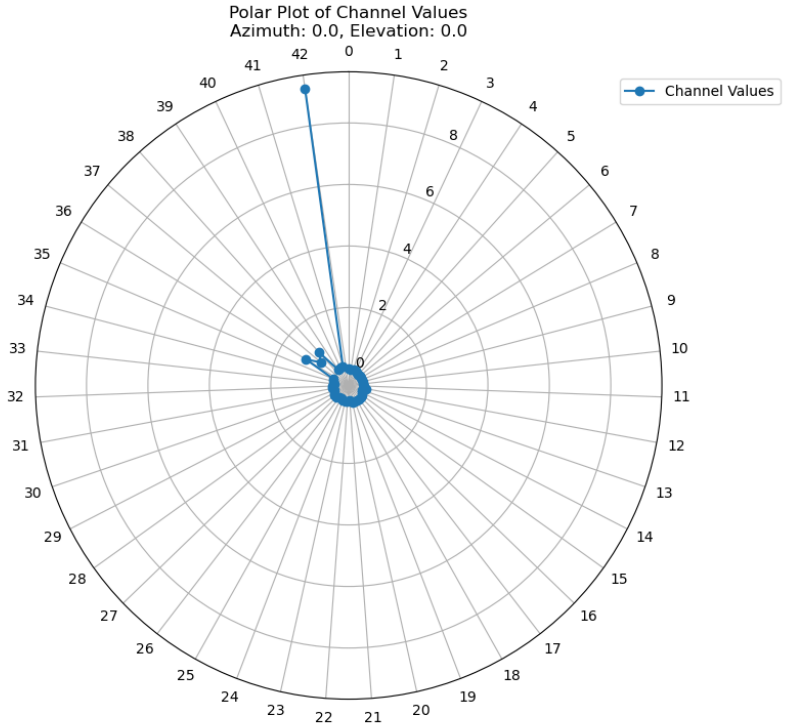

Figure 4.18: Polar plot of Channel values: Azimuth = 0°, Elevation = 0°

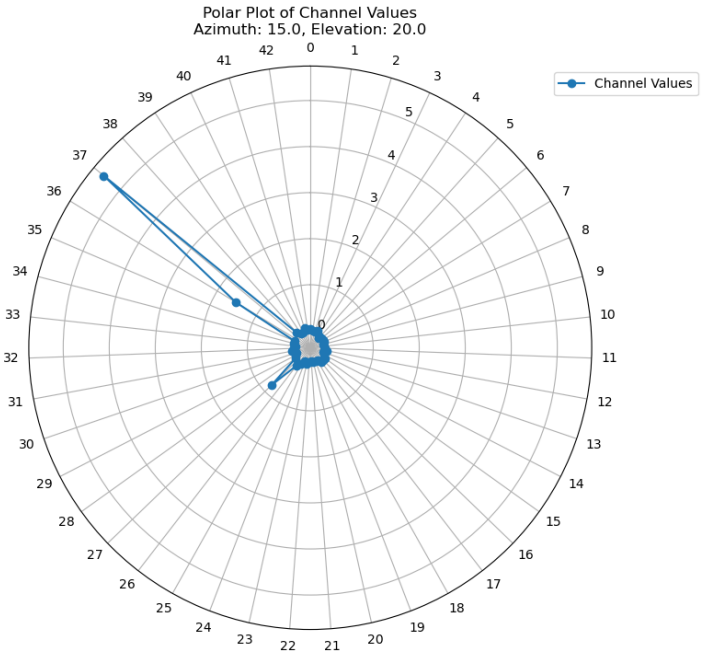

Figure 4.19: Polar plot of Channel values: Azimuth = 15°, Elevation = 20°

A snapshot of the live data was captured at the beginning of the laboratory field test using the code mentioned above referring to figure 4.7. This snapshot was taken when the solar tracking device was in its initial position of azimuth and elevation angles at 0° before testing.

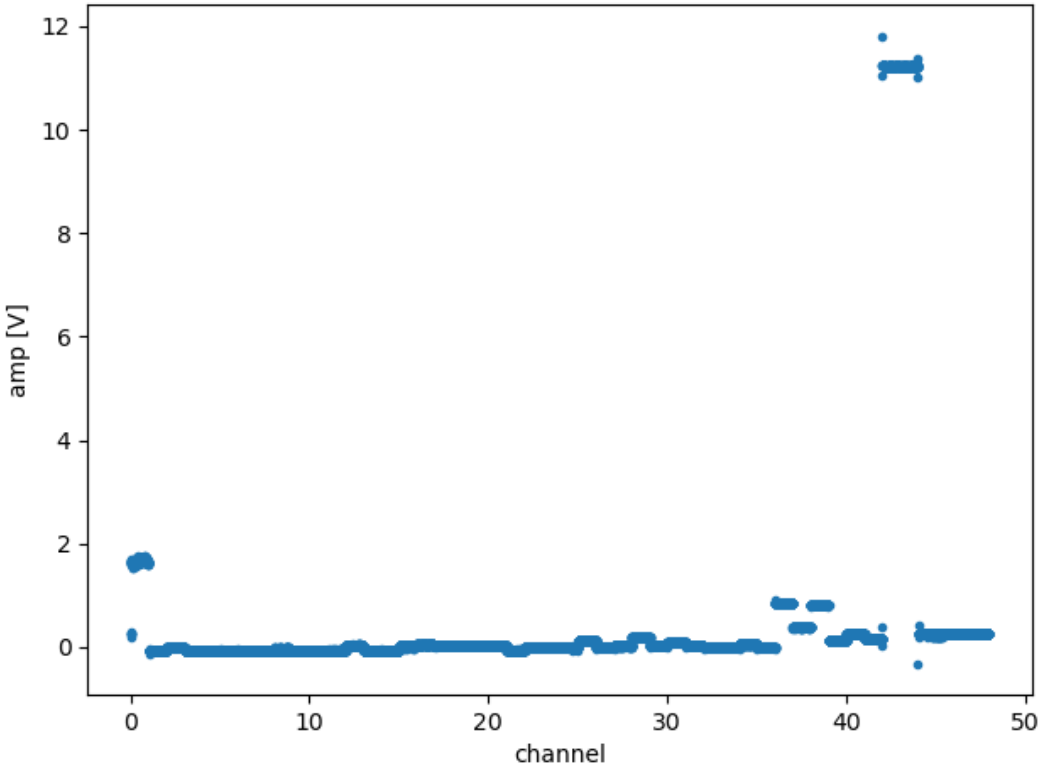

Figure 4.20: Live data plot of initial laboratory field test position.

It can be seen above there is huge spike in light intensity in the 42nd channel which refers to the 42nd LED on the device, there is also some small spikes in the channels in the high 30's. These results coincide with the expected outcome as due to figure 4.17, it can be seen at this position of azimuth and elevation, the corresponding channels that should receive the most light are the high 30s and mostly channel number 42.

Chapter 5.

Discussion & Conclusion

5.1 Errors and implications encountered.

With the nature of this project, there was little error analysis that was seen throughout. Some errors that were seen were in the computer-aided design process. Small errors in the experimental process are mentioned above, such as the imperfections in the microstep driver. This was resolved promptly using a newer microstep driver.

5.1.1 Scrapped solar tracker holder designs.

During this process there were prototypes designed and printed but contained faults and produced errors when printing. An example of one of these is seen below.

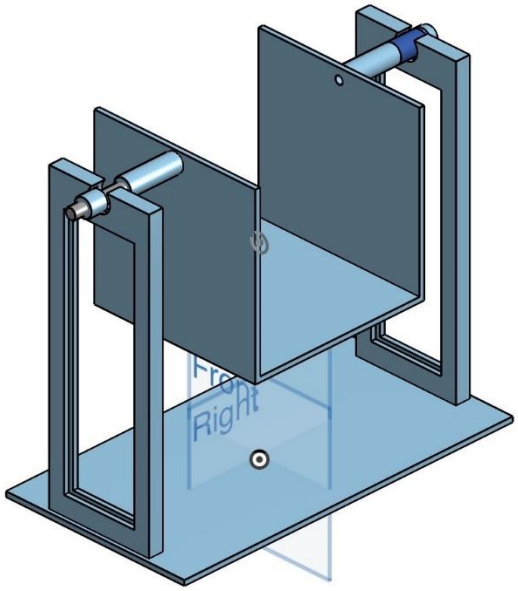

Figure 5.1: Scrapped design of solar tracking holder.

The main error in this design above was in the exterior swing holder. The cut in the centre of the walls were too large leaving the rest of the walls too weak, the bridge between the top

horizontal beams was too long and thin, thus resulting in a failure when printing. Thus, a new design was created to fix this.

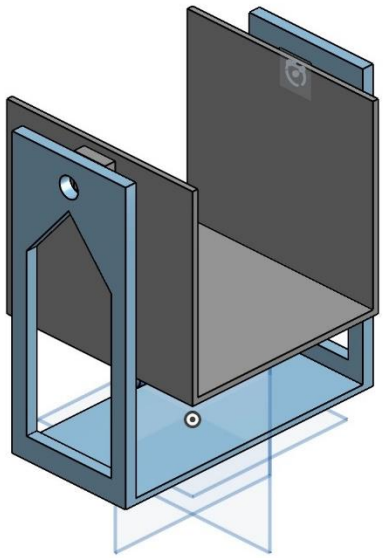

Figure 5.2: Secondary design of solar tracker holder.

This design for the outer holder printed with no errors and worked and fit well, it was later scrapped as a more efficient design was created (seen in figure 3.10) which used less material and provided a base for the support bracket holder to be bolted to.

5.1.2 Rigidity of the system

While calibrating and testing the motion of the full system, implications were seen with swinging of the tracking device. Due to our inexperience in computer aided design, our models were very basic and not the most efficient they possibly could. It was noticed that when the elevation angle would be changed, there would be a slight wobble in the holder. This would cause inaccuracies in our results, thus, to fix the problem a delay of 3 seconds was made after the elevation had been changed and then recordings were taken by the program.

5.2 Improvements to be made for future experimenting.

Although the overall aim of the project was fulfilled and a successful set of results was compiled, there was definitely some improvements that could be made to the system to optimise its output.

Due to time constraints, cable management for the system was not fully prioritised, enough was done to allow the solar tracker to move freely and collect data for all configurations of angles needed without any cable pulling. Although a better attempt could be made to hide wires and shorten cables to create a more compact and easily transportable device. With an easily transportable device, more field testing in more diverse environmental surroundings could be completed.

Another improvement to be made the solar tracker would be the implementation of Smart tracking algorithms. Incorporating advanced algorithms that predict the sun's position based on time, date, and location data can optimize tracking efficiency. Machine learning algorithms can learn from historical data to continuously improve tracking accuracy and adapt to changing weather conditions.

5.3 Conclusion

The aim of this project was to complete an absolute calibration of the solar tracking device – the “Solar hedgehog”. A tracking device holder was successfully designed and printed to aid in this full calibration. Results were gathered on the testing of components of the tracking device and computational programs were created to compile a large amount of data in a laboratory field test that tested the accuracies of the individual LEDs on the device.

This project fulfilled all the aims intended at the beginning of this report but there were some limitations due to time constraints and errors through the course of the project. If I were to start this project again, less time would be spent on designing the solar tracker holder and more time would be spent on other aspects of the design such as a waterproof cover for the system or housing a possible battery to operate the whole system for more adaptability.

Future work on this project can be done by means of outdoor field testing and building upon the above ideas such as smarter tracking algorithms, waterproofing and battery powering.

References

[1] Zach Wendt. “5 Methods of Harvesting Solar Energy.” *Arrow.com*, Arrow.com, 5 May 2020, www.arrow.com/en/research-and-events/articles/5-methods-of-harvesting-solar-energy.

[2] Wesleyan University, University of California, and University of Tennessee. “What Is a Solar Collector? Why Is It Important? Overview and Types.” *Treehugger*, 23 Feb. 2022, www.treehugger.com/what-is-a-solar-collector-5194092.

[3] Boyle, Godfrey, and Open University. *Renewable Energy*. Oxford ; New York, Oxford University Press In Association With The Open University, 2004.

[4] “Solar Geometry - Index of Convalescence.” *www.of-Convalescence.com*, www.of-convalescence.com/entries/solar-geometry#:~:text=Solar%20geometry%20is%20the%20measurement.

[5] Pons, Rafael. “Understanding Azimuth and Elevation.” *PhotoPills*, www.photopills.com/articles/understanding-azimuth-and-elevation.

[6] Patel, Dhavalkumar, and Dr. Bhavesh Patel. “Low Cost and Robust Solar Tracking System Based on Data of Daily and Seasonal Variation in Sun Position Regard to Specific Location on Earth.” *International Journal of Innovative Research in Science, Engineering and Technology*, vol. 03, no. 09, 15 Sept. 2014, pp. 15888–15893, https://doi.org/10.15680/ijirset.2014.0309014.

[7] Repsol. “Solar Trackers: What They Are, Types, and Advantages.” *REPSOL*, 15 Sept. 2023, www.repsol.com/en/energy-and-the-future/future-of-the-world/solar-trackers/index.cshtml.

[8] Analog Devices. “Introduction to the ADALM2000 [Analog Devices Wiki].” *Wiki.analog.com*, wiki.analog.com/university/tools/m2k/devs/intro. Accessed 16 Apr. 2024.

[9] "Zaber Technologies." *Www.zaber.com*, www.zaber.com/products/rotary-stages. Accessed 21 Mar. 2024

[10] Zaber . "Zaber Technologies." *Www.zaber.com*, www.zaber.com/products/rotary-stages/RST/features. Accessed 14 Mar. 2024.

[11] Chai, Wesley. "What Is CAD (Computer-Aided Design)?" *WhatIs.com*, Dec. 2020, www.techtarget.com/whatis/definition/CAD-computer-aided-design.

[12] CENTENT. "OPERATING MANUAL CN0162 MICROSTEP DRIVE ."

CENTENT CN0162 - Operating Manual.

[13] R.T.A. s.r.l. "BSD Series Drivers."

BSD Series Drives specifications.

Chapter 6.

Appendix

```
import libm2k
from libm2kmu import M2KContextManager
import numpy as np
import matplotlib.pyplot as plt
from time import sleep


# channel to use (libm2k.CHANNEL_1 or libm2k.CHANNEL_2)
channel = libm2k.CHANNEL_1

# number of samples to capture
num_samples = 10_750

# the global sampling frequency determines the shortest possible
# interval between samples.  Available values are limited!
desired_sampling_frequency = 10_000
# oversampling ratio gives
# 1 means that all samples are returned,
# 2 that every second one is used (rest are discarded)
oversampling_ratio = 1
# this means that the effective sampling frequency is
# given by:
effective_sampling_frequency = desired_sampling_frequency / oversampling_ratio

# expected minimum value, can be conservative!
desired_minimum_voltage = -15.0
# expected maximum value, can be conservative!
desired_maximum_voltage = 15.0

# if this flag is on, the device will be programmed to use offset range
use_range_offset_function = False
if use_range_offset_function:
    midpoint = (desired_maximum_voltage + desired_minimum_voltage) / 2.0
else:
    midpoint = 0.0 # do not use level pre-shifting
class Constant:
    def __init__(self, value = False):
        self._value = value

    def __call__(self, time):
        return np.full_like(time, self._value)

class Pulses:
    def __init__(self,*, N = 1, F=1, DC = 0.5, delay = 0.0):
        self._N = N
        self._T = 1.0 / F
        self._DC = DC



    def __call__(self, time):
        pulse_number = (time - self._delay) // self._T
        pulse_fraction = ((time - self._delay) % self._T) / self._T

        return (pulse_number >= 0) & (pulse_number < self._N) & \
            (pulse_fraction <= self._DC)

# set overall generation window time is seconds

OVERALL_TIME = 1.2
SAMPLE_RATE = 10_000
BITS_USED = 6
```

```
# entries in this list declare individual bits as inputs (False) or outputs (Tr
# 16 entries altogether
USE_AS_OUTPUT = [True, True, True, True, True, True, True, True,
                 True, False, False, False, False, False, False, False ]

# signals to use for various bits:
# None means ignore (False) -- appropriate for inputs
# Constant(True) -- signal that is always 1
# Constant(False) -- signal that is always 0
# Pulses(N, F, DC, DELAY) -- after a delay of DELAY, represent N pulses at
# frequency F and duty cycle DC

SIGNALS = [ Pulses(delay=0.025,N=21,F=20,DC=0.5),
          Pulses(delay=0.05,N=11,F=10,DC=0.5),
          Pulses(delay=0.1,N=5,F=5,DC=0.5),
          Pulses(delay=0.2,N=3,F=2.5,DC=0.5),
          Pulses(delay=0.4,N=3,F=1.25,DC=0.5),
          Pulses(delay=0.8,N=1,F=0.625,DC=0.5),
          Constant(True),
          Constant(True),
          Pulses(delay=1e-4,N=1,F=1000,DC=0.5),
          None,None,None,None,None,None,None]
    # Constant(True), Pulses(N=50,F=10_000,DC=0.5), None, None, None, None,\
       #None, None,
    #        None, None, None, None, None, None, None, None]

# __Important note__
#
# All of the ADALM2000 code should be within this with statement
# this ensures that the context is correctly deallocated even in the
# case of an error, an exception or explicit user interrruption.
# Without this (pun intended) Python would have to be restarted to allow
# reuse of the hardware.

with M2KContextManager() as ctx:

    dig=ctx.getDigital()
    dig.reset()

    # set the digital output sample rate
    dig.setSampleRateOut(SAMPLE_RATE)

    #  configure all the digital pins
    for i, output_bit in enumerate(USE_AS_OUTPUT):
        if output_bit:
            dig.setDirection(i,libm2k.DIO_OUTPUT)
        else:
            dig.setDirection(i,libm2k.DIO_INPUT)


        dig.enableChannel(i,True)

    sample_rate = dig.getSampleRateOut()
    sample_period = 1.0 / sample_rate

    # determine the number of samples that will be output
    num_samples = int(OVERALL_TIME / sample_period) + 1

    print(f'Digital output sample rate is {sample_rate}.')
    print(f'{num_samples} samples will be output once.')
    # determine the time offsets of individual samples
    sample_times = np.arange(num_samples) * sample_period + 0.5/SAMPLE_RATE

    # all the samples, all zero for now
    samples = np.zeros(num_samples, dtype='int16')
```

```
for i, signal in enumerate(SIGNALS):
    if signal:
        # get the signal values at all sample times
        pin_samples = signal(sample_times)
        # convert to 0 or 1 (int), multiply by the pbit weight
        # and add to the overall signal (all pins)
        samples += 2**i * pin_samples.astype(int)

# output only once !
dig.setCyclic(True)
dig.push(samples.data)
ain=ctx.getAnalogIn()
ain.reset()
ctx.calibrateADC()

ain.enableChannel(channel,True)
ain.setSampleRate(desired_sampling_frequency)
ain.setOversamplingRatio(oversampling_ratio)

# ADALM2000 has a trick up its sleeve, despite only two input gains!
# it can offset input voltages by a programmable amount, before digitising
# This means that the more sensitive range can be used on voltages
# in excess of +/- 2.5 V, as long as the range of expected values is
# narrow enough
# try to set that up if requested
if use_range_offset_function:
    ain.setVerticalOffset(channel,-midpoint)

ain.setRange(channel,desired_minimum_voltage - midpoint,
             desired_maximum_voltage - midpoint)

trigger = ain.getTrigger()
trigger.setAnalogMode(channel, libm2k.EXTERNAL)

# the offset range mode seems to come with a notable settling time,
# that can affect the start of the first high sampling rate captures
# avoid this by adding a short delay
if use_range_offset_function:
    sleep(0.05)
print(f'Wanted {desired_minimum_voltage} to {desired_maximum_voltage}\
      V range')

# determine what we were able to get
gain, (low_limit, high_limit) = ain.getAvailableRanges()[ain.getRange(chann
offset = - ain.getVerticalOffset(channel)

print(f'Got {low_limit + offset} to {high_limit +offset} V effective range\
      ( {gain} gain setting)')

print()



print(f'Wanted {effective_sampling_frequency} S/s sampling rate')
actual_sampling_frequency = ain.getSampleRate()/ain.getOversamplingRatio()
print(f'Got {actual_sampling_frequency} S/s sampling rate')

if actual_sampling_frequency != desired_sampling_frequency:
    print('Available sampling frequencies are: ',\
          ain.getAvailableSampleRates())

# set up the figure and individual plots
# then make it interactive
fig, axs = plt.subplots(1, 1)
fig.set_dpi(150)

plt.ion()

first time = True
```

```
    while True:
        samples = ain.getSamples(num_samples)[channel] # both are always measur
        times = np.arange(len(samples)) / actual_sampling_frequency
        n = len(samples)
        timestep = 1.0 / actual_sampling_frequency
        freqs = np.fft.rfftfreq(n, d=timestep)
        if first_time:
            # create the plots from scratch in the first paSS
            axs.set_xlabel('channel')
            axs.set_ylabel('amp [V]')
            #axs[1].set_xlabel('f [Hz]')
            #axs[1].set_ylabel('amp [dBV]')
            time_plot, = axs.plot(times/0.025, samples,'.')
            #freq_plot, = axs[1].plot(freqs,20.0 * np.log10(np.abs(spectrum)),'
            plt.tight_layout()
            first_time = False
        else:
            # now only update the data. Do not create new plots!
            time_plot.set_xdata(times/0.025)
            time_plot.set_ydata(samples)
           # freq_plot.set_xdata(freqs)
            #freq_plot.set_ydata(20.0 * np.log10(np.abs(spectrum)))
            # rescale the plots for potentially new data
            axs.relim()
            axs.autoscale_view(True, True, True)
            #'axs[1].relim()
            #axs[1].autoscale_view(True, True, True)

        # allow the interactive figure canvas to redraw regularly
        fig.canvas.draw()
        fig.canvas.flush_events()
```

Figure 6.1: Program to produce a live data plot

```
import libm2k
from libm2kmu import M2KContextManager
import numpy as np
import matplotlib.pyplot as plt
from time import sleep
from zaber_motion import Units
from zaber_motion.binary import Connection
from datetime import datetime

class Constant:
    def __init__(self, value = False):
        self._value = value

    def __call__(self, time):
        return np.full_like(time, self._value)

class Pulses:
    def __init__(self,*, N = 1, F=1, DC = 0.5, delay = 0.0):
        self._N = N
        self._T = 1.0 / F
        self._DC = DC
        self._delay = delay

    def __call__(self, time):
        pulse_number = (time - self._delay) // self._T
        pulse_fraction = ((time - self._delay) % self._T) / self._T

        return (pulse_number >= 0) & (pulse_number < self._N) & \
            (pulse_fraction <= self._DC)
```

```
def initialisehedgehog(ctx):
    ain=ctx.getAnalogIn()
    ain.reset()
    ctx.calibrateADC()

def measurehedgehog(ctx):
    # channel to use (libm2k.CHANNEL_1 or libm2k.CHANNEL_2)
    channel = libm2k.CHANNEL_1

    # number of samples to capture
    num_samples = 10_750

    # the global sampling frequency determines the shortest possible
    # interval between samples.  Available values are limited!
    desired_sampling_frequency = 10_000
    # oversampling ratio gives
    # 1 means that all samples are returned,
    # 2 that every second one is used (rest are discarded)
    oversampling_ratio = 1



    # expected minimum value, can be conservative!
    desired_minimum_voltage = -15.0
    # expected maximum value, can be conservative!
    desired_maximum_voltage = 15.0



    # set overall generation window time is seconds

    OVERALL_TIME = 1.2
    SAMPLE_RATE = 10_000



    # entries in this list declare individual bits as inputs (False) or outputs
    # 16 entries altogether
    USE_AS_OUTPUT = [True, True, True, True, True, True, True, True,
                     True, False, False, False, False, False, False, False ]

    SIGNALS = [ Pulses(delay=0.025,N=21,F=20,DC=0.5),
              Pulses(delay=0.05,N=11,F=10,DC=0.5),
              Pulses(delay=0.1,N=5,F=5,DC=0.5),
              Pulses(delay=0.2,N=3,F=2.5,DC=0.5),
              Pulses(delay=0.4,N=3,F=1.25,DC=0.5),
              Pulses(delay=0.8,N=1,F=0.625,DC=0.5),
              Constant(True),
              Constant(True),
              Pulses(delay=1e-4,N=1,F=1000,DC=0.5),
              None,None,None,None,None,None,None]

    dig=ctx.getDigital()
    dig.reset()

    # set the digital output sample rate
    dig.setSampleRateOut(SAMPLE_RATE)
```

```
#  configure all the digital pins
for i, output_bit in enumerate(USE_AS_OUTPUT):
    if output_bit:
        dig.setDirection(i,libm2k.DIO_OUTPUT)
    else:
        dig.setDirection(i,libm2k.DIO_INPUT)

    dig.enableChannel(i,True)

sample_rate = dig.getSampleRateOut()
sample_period = 1.0 / sample_rate

# determine the number of samples that will be output
num_samples = int(OVERALL_TIME / sample_period) + 1

# determine the time offsets of individual samples
sample_times = np.arange(num_samples) * sample_period + 0.5/SAMPLE_RATE

# all the samples, all zero for now
samples = np.zeros(num_samples, dtype='int16')
for i, signal in enumerate(SIGNALS):
    if signal:
        # get the signal values at all sample times
        pin_samples = signal(sample_times)
        # convert to 0 or 1 (int), multiply by the pbit weight
        # and add to the overall signal (all pins)
        samples += 2**i * pin_samples.astype(int)

# output only once !
dig.setCyclic(True)
dig.push(samples.data)
ain=ctx.getAnalogIn()

#ain.reset()
# ain.enableChannel(channel,False)

ain.enableChannel(channel,True)
ain.setSampleRate(desired_sampling_frequency)
ain.setOversamplingRatio(oversampling_ratio)

ain.setRange(channel,desired_minimum_voltage, desired_maximum_voltage)


trigger = ain.getTrigger()
trigger.setAnalogMode(0, libm2k.EXTERNAL)
trigger.setAnalogExternalCondition(0, libm2k.RISING_EDGE_ANALOG) #
trigger.setAnalogDelay(0) #
ain.stopAcquisition()
samples = np.asarray(ain.getSamples(num_samples)[channel]) # both are alway

n = len(samples)
channel_values=[]


for channel_number in range (43):
    start_index=8+channel_number*250+25
    channel_values.append(samples[start_index:start_index+200].mean())
```

```
    return np.asarray(channel_values)

def initialisezaber():
    with Connection.open_serial_port("COM8") as connection:

        device_list = connection.detect_devices()
        #print("Found {} devices".format(len(device_list)))
        device = device_list[0]
        device.home(unit = Units.ANGLE_DEGREES)
    return

def setzaber(position):
    with Connection.open_serial_port("COM8") as connection:
        #connection.enable_alerts()

        device_list = connection.detect_devices()
        #print("Found {} devices".format(len(device_list)))
        device = device_list[0]
        device.move_absolute(position,unit = Units.ANGLE_DEGREES)
        print('Zaber Position: ',device.get_position(unit = Units.ANGLE_DEGREES

def measuremotor(ctx):
    class Constant:
        def __init__(self, value = False):
            self._value = value

        def __call__(self, time):
            return np.full_like(time, self._value)

    class Pulses:
        def __init__(self,*, N = 1, F=1, DC = 0.5, delay = 0.0):
            self._N = N
            self._T = 1.0 / F
            self._DC = DC
            self._delay = delay

        def __call__(self, time):
            pulse_number = (time - self._delay) // self._T
            pulse_fraction = ((time - self._delay) % self._T) / self._T

            return (pulse_number >= 0) & (pulse_number < self._N) & (pulse_frac


    Number_of_steps = 1550
    StepRate = 50
    OVERALL_TIME = Number_of_steps/StepRate *1.05
    direction = True
    SAMPLE_RATE = 10_000
    BITS_USED = 2

    USE_AS_OUTPUT = [False, False, False, False, False, False, False, False,
                     False, True, True, False, False, False, False, False ]


    SIGNALS = [None,None,None,None,None,None,None,None,None,
               Pulses(N=Number_of_steps/2, F=StepRate/2),Constant(direction),Non

    dig=ctx.getDigital()
    dig.reset()
```

```
    dig.setSampleRateOut(SAMPLE_RATE)

    #  configure all the digital pins
    for i, output_bit in enumerate(USE_AS_OUTPUT):
        if output_bit:
            dig.setDirection(i,libm2k.DIO_OUTPUT)
        else:
            dig.setDirection(i,libm2k.DIO_INPUT)

        dig.enableChannel(i,True)

    sample_rate = dig.getSampleRateOut()
    sample_period = 1.0 / sample_rate

    # determine the number of samples that will be output
    num_samples = int(OVERALL_TIME / sample_period) + 1


    # determine the time offsets of individual samples
    sample_times = np.arange(num_samples) * sample_period + 0.5/SAMPLE_RATE

    # all the samples, all zero for now
    samples = np.zeros(num_samples, dtype='int16')
    for i, signal in enumerate(SIGNALS):
        if signal:
            # get the signal values at all sample times
            pin_samples = signal(sample_times)
            # convert to 0 or 1 (int), multiply by the pbit weight
            # and add to the overall signal (all pins)
            samples += 2**i * pin_samples.astype(int)

    # output only once !
    dig.setCyclic(False)
    dig.push(samples.data)


    return

elevationsteps = 0


def initialiseelevation(ctx):

    ps=ctx.getPowerSupply()
    ps.reset()
    ps.enableChannel(0,True)
    ps.pushChannel(0,5)

    global elevationsteps
    elevationsteps = 0

    return

MICROSTEPSPERROTATION = 800*(720/736.09)
StepRate = 200
```

```
def setelevation(position,ctx):
    global elevationsteps
    newelevationsteps = int(position/360 * MICROSTEPSPERROTATION)
    changeelevationsteps = newelevationsteps - elevationsteps

    if changeelevationsteps == 0:
        return

    Number_of_steps = abs(changeelevationsteps)

    OVERALL_TIME = Number_of_steps/StepRate *1.2

    direction = (changeelevationsteps > 0)
    SAMPLE_RATE = 10_000

    USE_AS_OUTPUT = [False, False, False, False, False, False, False, False,
                     False, True, True, True, False, False, False, False ]


    SIGNALS = [None,None,None,None,None,None,None,None,None,
               Pulses(N=Number_of_steps, F=StepRate),Constant(direction),None,

    dig=ctx.getDigital()
    dig.reset()

    # set the digital output sample rate
    dig.setSampleRateOut(SAMPLE_RATE)

    #  configure all the digital pins
    for i, output_bit in enumerate(USE_AS_OUTPUT):
        if output_bit:
            dig.setDirection(i,libm2k.DIO_OUTPUT)
        else:
            dig.setDirection(i,libm2k.DIO_INPUT)

        dig.enableChannel(i,True)

    sample_rate = dig.getSampleRateOut()
    sample_period = 1.0 / sample_rate

    # determine the number of samples that will be output
    num samples = int(OVERALL TIME / sample period) + 1

    # determine the time offsets of individual samples
    sample_times = np.arange(num_samples) * sample_period + 0.5/SAMPLE_RATE

    # all the samples, all zero for now
    samples = np.zeros(num_samples, dtype='int16')
    for i, signal in enumerate(SIGNALS):
        if signal:
            # get the signal values at all sample times
            pin_samples = signal(sample_times)
            # convert to 0 or 1 (int), multiply by the pbit weight
            # and add to the overall signal (all pins)
            samples += 2**i * pin_samples.astype(int)

    # output only once !
    dig.setCyclic(False)
    dig.push(samples.data)
    sleep(OVERALL_TIME)
    elevationsteps = newelevationsteps
```

```
    print('Elevation Position: ', f'{(newelevationsteps/(200/90)):.2f}')
    return


with M2KContextManager() as ctx:

    initialisehedgehog(ctx)
    initialisezaber()
    initialiseelevation(ctx)

    azimuthpositions = np.arange(0,20, 10)
    elevationpositions = np.arange(-30,30, 10)

    #print(elevationsteps)


    data_list = [] # Initialize an empty list to store data

    for azimuth in azimuthpositions:
        setzaber(azimuth)
        for elevation in elevationpositions:
            setelevation(elevation, ctx)
            sleep(3)
            channel_values = measurehedgehog(ctx)
            print(channel_values)
            data_list.append(np.concatenate(([elevation, azimuth],\
                                             channel_values)))

data_array = np.array(data_list) # Convert data to NumPy array



# Save the data to a CSV file using NumPy

current_datetime = datetime.now().strftime("%Y-%m-%d %H-%M-%S")
header = 'Elevation, Azimuth,' + ','.join(str(i) for i in range(43))
np.savetxt(f'data-{current_datetime}.csv', data_array, delimiter=',',\
           header=header)
```

Figure 6.2: Main program used to automate system and collect data

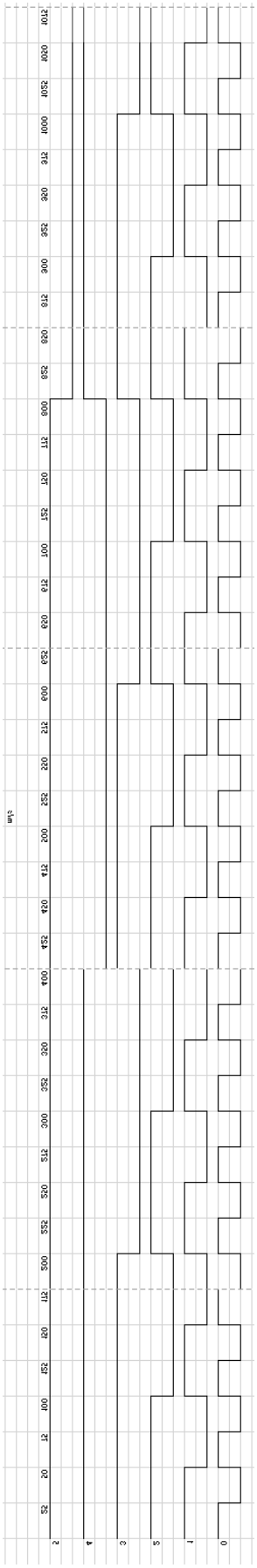

Figure 6.3: Waveform images

```python
import csv
import matplotlib.pyplot as plt
import numpy as np

def create_polar_plot(filename, row_number):
    with open(filename, 'r') as file:
        reader = csv.reader(file)
        for _ in range(row_number-1):  # Skip rows until the desired row
            next(reader)
        row = next(reader)  # Read desired row
        elevation = float(row[0])  # Elevation value from the row
        azimuth = float(row[1])    # Azimuth value from the row
        channels = list(map(float, row[2:45]))# Convert channel values to float

    # Define angles for the channels
    num_channels = len(channels)
    angles = np.linspace(0, 2*np.pi, num_channels, endpoint=False)

    # Plotting
    fig, ax = plt.subplots(subplot_kw={'projection': 'polar'})
    ax.plot(angles, channels, marker='o', linestyle='-', label=\
            'Channel Values')

    # Add channel values as labels around the plot
    ax.set_xticks(angles)
    ax.set_xticklabels(range(43))

    # Add title with azimuth and elevation values
    ax.set_title(f'Polar Plot of Channel Values\nAzimuth: {azimuth}, \
                 Elevation: {elevation}')


    # Move the legend outside the plot
    ax.legend(loc='upper right', bbox_to_anchor=(1.2, 1))
    # Set the direction of the zero angle to be at the top (North)
    ax.set_theta_zero_location('N')
    ax.set_theta_direction(-1)  # Reverse direction for theta

    plt.show()

# Assuming your CSV file is named "data.csv"
filename = "data-2024-04-22 17-25-57.csv"
row_number = 457 #Row in excel to plot
create polar plot(filename, row number)
```

Figure 6.4: Program used to create polar plots of collected data

data-2024-04-22
17-25-57.csv

Figure 6.5: Raw data file of laboratory field test recordings