

GLASGOW CALEDONIAN UNIVERSITY

MEng Group Research Project

MMH723842-24-AB-GLAS

**Design and Implementation of a Photodiode
Array-Based Analogue 2D Sun Sensor**

word count: xxx

by Zac McCaffery, Alexandru Belea,
Sebastian Alexander, William Kong, Nassor Salim,

Date: April 17, 2025

Contents

Abstract	12
1. Acknowledgements	13
2. Introduction	14
2.1. Problem Statement	14
2.2. Aim of the Project	14
2.3. Objectives of the Project	14
3. Literature Review	16
3.1. CubeSat Design	16
3.2. PSD Enabled Sun Sensor	16
3.3. Mechanical Design and Analysis	18
3.4. Photodiode Simulation and Signal Analysis	19
3.5. IoT Communication Enhancement with LEO Satellites	20
4. Methodology	22
4.1. Prototype Development	22
4.1.1. Lifecycle	22
4.1.2. Signal Conditioning Circuitry	30
4.1.3. OpAmp Noise	31
4.1.4. Enclosure Design 3D print	35
4.2. Data Acquisition System	48
4.2.1. Functional Requirements	48
4.2.2. Design Approach	49
4.2.3. Technical Specifications	50
4.2.4. Testing Strategy	60
4.2.5. Evaluation of design	61
4.3. Renewable Energy Demonstrator Testbench	61
4.3.1. LED Controller reprogramming	62
4.3.2. Arch Movement Reprogramming	63
4.4. Software Model	65
4.4.1. Functional Requirements	65
4.4.2. Theory and Concept	65

4.4.3. Implementation	69
4.4.4. Data Analysis and Evaluation	76
4.4.5. Model Results	81
4.5. Material Analysis and Selection	82
4.5.1. Material Selection and Requirements	82
4.5.2. External Factors	82
4.5.3. Internal Factors	83
4.5.4. PCB Material Selection	84
4.5.5. CubeSat Chassis Material	89
4.5.6. Quality Assurance and Reliability Standards in Space PCB Manufacturing	91
4.6. Thermal Analysis of the PCB	92
4.6.1. Hypothesis	92
4.6.2. Thermal Analysis Parameters	92
4.6.3. Thermal results	93
4.6.4. Finite Analysis Evaluation 1 - Under 200 °C in space	93
4.6.5. Results Evaluation	95
4.7. CubeSat Chassis Design	95
4.7.1. CubeSat Chassis Design 1	96
4.7.2. CubeSat Chassis Design 2	96
5. Results	100
5.1. Sensor Characterization	100
5.2. Amplification Performance	100
5.3. Photodiode Angular Response	100
5.4. Enclosure Effectiveness	100
5.5. Data Acquisition System (DAQ) System Evaluation	101
5.6. Comparative Analysis	101
5.6.1. Breadboard vs. Stripboard Results	102
5.6.2. Iteration Improvements Analysis	102
5.6.3. Performance Against Design Requirements	102
5.6.4. Design Evolution Assessment	102
5.7. System Limitations And Considerations	102
5.7.1. Renewable Energy Demonstrator (RED) testbench limitations and impact	102
5.7.2. Aperture placement accuracy	104
5.7.3. Model Limitations and Future Work	104
6. Conclusions	105

7. Future Work	106
Bibliography	107
A. Appendix - DAQ	113
A.1. Arduino DAQ full code	113
A.1.1. Arduino C++ Code	113
A.1.2. PC-side Python Serial Receive Script	115
A.2. Derivation of TIA Voltage Out	121
B. Appendix - Software Model Code	122
B.1. Section 1 Title	122
B.1.1. subsectiontitle	122
B.2. Section 2 Title	122
B.2.1. subsectiontitle	122
C. Appendix - Photos of Lab Work	123
C.1. Prototype Images	123
C.1.1. BreadBoard Prototype	123
C.1.2. Building the Prototype	124
C.1.3. Prototype Testing	126
C.1.4. RED testbench	127
C.1.5. Solar Lab Prototype Testing	128
D. Appendix - Material Analysis	133
D.1. Material Analysis - ANSYS	133
D.1.1. PCB Comparison	133
D.1.2. Aluminium Comparison	133
E. Appendix - CAD and 3D Printing	144
E.1. CAD Modelling and 3D printing	144
E.1.1. Design Iterations	144
E.1.2. CubeSat Chassis CAD Modelling	148
F. Appendix - Renewable Energy Demonstrator (RED) testbench code	151
F.1. Appendix - RED LED Strip Code	151
F.1.1. LED Strip Code for automatic transition	151
F.1.2. LED Strip Code for Manual 5 location transition	153
F.1.3. ARCH controller C++ code	156
G. Appendix - Meeting Notes	159
G.1. Meeting Notes	159

H. Appendix Software Model Code	165
H.1. Main Script: <code>main.py</code>	165
H.2. <code>plane.py</code>	199
H.3. <code>line.py</code>	205
H.4. <code>intersectionCalculations.py</code>	207
H.5. <code>areas.py</code>	208
H.6. <code>config.py</code>	211
H.7. <code>config.json</code>	212
H.8. <code>arcRotation.py</code>	215
H.9. <code>analyse_results.py</code>	219
H.10. <code>interface.py</code>	225
H.11. <code>requirements.txt</code>	234
H.12. <code>test_arc_rotation.json</code>	234
H.13. <code>test_directly_below.json</code>	239
H.14. <code>test_no_intersection.json</code>	242
H.15. <code>test_off_center.json</code>	245
H.16. <code>test_with_aperture.json</code>	249
H.17. <code>utils_io.py</code>	252
H.18. <code>utils_metrics.py</code>	252
H.19. <code>utils_plot.py</code>	253

List of Figures

4.1. Diagram of Photodiode Array with Apertures	23
4.2. Photo Of BreadBoard Prototype	26
4.3. Photo Of Stripboard Prototype on Fritzing	28
4.4. Photo Of Stripboard Prototype	29
4.5. Photo of aperture placement on screen protector glass	29
4.6. TIA and Post Amplification Circuit in Altium Designer	32
4.7. First Design Iteration of the Enclosure	36
4.8. Second Design Iteration of the Enclosure	36
4.9. Third Design Iteration of the Enclosure	37
4.10. Final Design Iteration of the Enclosure	38
4.11. SLA Printer	39
4.12. Creality Ender V3 SE	41
4.13. Single Head FDM Process Layout	42
4.14. FDM Parameters	42
4.15. Flowchart of the FDM Process	43
4.16. Various Infill Patterns Used for FDM Processes	43
4.17. Design Sliced for Printing	45
4.18. Bed Adhesion Failure	45
4.19. Test Print After Temperature Adjustments	46
4.20. Part Printed with Brim	46
4.21. Improved Bed Adhesion from Test Print Attempts	47
4.22. Dimensional Inaccuracy of the Holes after Printing	48
4.23. Signal Noise Analysis, oscilloscope AC coupled	49
4.24. Flowchart Arduino DAQ C++ program	53
4.25. FSM Arduino C++ LOOP	54
4.26. Python Script Flowchart	57
4.27. Mapping of the s-plane onto the z -plane using the bilinear transformation [1, p.130]	59
4.28. Frequency Response of 4-pole Butterworth Low-Pass Filter (Cutoff: 2.0 Hz, Order: 4, Fs: 500 Hz)	60
4.29. The RED Testbench modified to fit our sensor	61

4.30. Flowchart of the C++ program on the Arduino Controlling the Arch and other peripherals	63
4.31. Sensor Toplogy	66
4.32. Rigid Arc Scanning Mode - Tilt Angles Left (0°) Right ($0^\circ, 45^\circ$)	71
4.33. Horizontal and Vertical Scanning Modes	71
4.34. Model Configuration Interface	76
4.35. <code>plot_hit_percentage()</code> Output Plot	78
4.36. <code>sensor_surface_plots()</code> Output Plot	79
4.37. Run Time and Hit Gain Efficiency Output Plot	81
4.38. Runtime and efficiency trends with increasing ray count. Left: Runtime increases linearly with ray count. Middle: Marginal gain in hit accuracy. Right: Cost in seconds per 1% hit gain.	81
4.39. FR-4 PCB [2]	85
4.40. Alumina Oxide PCB [3]	86
4.41. PTFE/Teflon sheets [4]	87
4.42. Copper foils [5]	88
4.43. Kapton PCB [6]	88
4.44. PCB model with Parameters applied	93
4.45. Results under 200°C radiation	94
4.46. Results under -200°C radiation	95
4.47. Front view of the first CubeSat chassis design	96
4.48. Angular view of the first CubeSat chassis design	96
4.49. Front view of 1U Cubesat Skeleton Chassis [7]	97
4.50. Angular view of 1U Cubesat Skeleton Chassis [7]	97
4.51. Second chassis design without PCBs	98
4.52. Chassis with the aperture	99
5.1. Model and Physical Results Comparison	101
A.1. Solution Deriving TIA Vout	121

List of Tables

4.1.	Electrical Components Used in Amplification Circuit	24
4.2.	Components Used for DAQ System	24
4.3.	S5971 Photodiode Specifications	24
4.4.	Operational Amplifier Specifications	25
4.5.	Arch and RGB LED Coordinates in Hemispherical System	62
4.6.	Geometric Vector Definitions	66
4.7.	Definition of vectors and symbols used in ray and plane calculations	69
4.8.	Optical Simulation Components	70
4.9.	Simulation Output: Summary of Each Arc Position	77
4.10.	Sensor Output: Ray Hits Per Sensor at Each Arc Position	77
4.11.	Summary of Analysis Functions	81

List of Equations

1. DC Offset Voltage at TIA Output Due to Dark Current	25
2. Total DC Offset After Post-Amplification	25
3. Transimpedance Amplifier Output Voltage	30
4. Photocurrent as Function of Optical Power	30
5. TIA Output with Photodiode Response	30
6. Low-Pass filter formed from Photodiode capacitance and Rf	30
7. Thermal Current Noise Calculation for Feedback Resistor	31
8. Johnson Noise from TIA feedback resistor	31
9. Amplified Voltage Noise with Gain of 10^6	31
10. Secondary Amplification Gain Calculation	32
11. Amplifier Feedback Resistor Calculation	33
12. Secondary Amplification Low Pass Filter Calculation	33
13. Default ADC Clock Frequency Calculation	50
14. Optimized ADC Clock Frequency Calculation	50
15. Default ADC Conversion Time	50
16. Optimized ADC Conversion Time	51
17. Total Sampling Time for 4 Channels (Default)	51
18. Total Sampling Time for 4 Channels (Optimized)	51
19. Maximum Theoretical Sampling Frequency (Default)	51
20. Maximum Theoretical Sampling Frequency (Optimized)	51
21. Actual Limited Sampling Frequency	51
22. Total Effective Data Rate	51
23. Frequency Response of 4th-order Butterworth low-pass filter with 2.0 Hz cutoff	59
26. Ray Generation	67
27. Ray projection	67
32. Rodrigues Rotation Formula Matrix	68
33. Skew-symmetric matrix	68
34. Overall Hit Results (%)	77
35. Cost per Gain accuracy	80

ADC	Analog to Digital Converter
ADCS	Attitude determination and control system
ATP	Acquisition, Tracking and Pointing
CCD	Charge-Coupled Device
CSV	Comma Separated Values
CTE	Coefficient of thermal expansion
DAQ	Data Acquisition System
FOV	Filed of View
GSFC	Goddard Space Flight Centre
LEO	Low Earth Orbit
OPS	Optical Position Sensor
PSD	Position Sensitive Detector
PV	Photovoltaic
RED	Renewable Energy Demonstrator
SGP4	Simplified General Perturbations 4
SMPS	Switch-Mode Power Supply
TIA	Transimpedance Amplifier
TLE	Two-line element set
VLC	Visible Light Communication
WSN	Wireless Sensor Networks
SMD	Surface Mount Device
OpAmp	Operational Amplifier
PCB	Printed Circuit Board
PLA	Polyactic Acid
LED	Light Emitting Diode
EDA	Electronic Design Automation

LUT Look-Up Table

EPS European Project Semester

Abstract

This research project explores the design, implementation, and testing of a prototype of a cost-effective photodiode array-based analogue 2D sun sensor for attitude determination in Low Earth Orbit (LEO) nanosatellites, with a focus on the CubeSat variety. As the commercialisation of space continues to grow, the demand for low-cost, reliable attitude determination systems for small satellites that cannot accommodate the expensive digital camera systems used in larger commercial missions. A comprehensive review of existing sun sensor technologies, CubeSat designs, mechanical considerations and signal analysis methods.

Based on this foundation, a prototype was developed utilizing four photodiodes arranged in a T-shaped configuration with appropriate apertures, coupled with transimpedance and secondary amplification circuitry to process the photodiode signals. The prototype was housed in a custom-designed 3D-printed enclosure to ensure proper positioning of the photodiodes and protection of the electronic components. A software model was created in parallel to simulate ray projection and intersection calculations, allowing for the prediction of sensor response under various light conditions. This model provided valuable insights for optimizing the physical design and interpreting experimental results. Additionally, material analysis was performed to evaluate suitable materials for space deployment, with polyimide identified as the optimal PCB material due to its balance of thermal stability, radiation resistance, and mechanical properties. Thermal analysis using ANSYS confirmed that the selected components would function reliably within the extreme temperature range of space environments from 200 °C to –200 °C. A Data Acquisition System (DAQ) based on an Arduino microcontroller was implemented to record and process the sensor data, incorporating digital filtering to eliminate noise and enhance signal quality. Testing was conducted using a Renewable Energy Demonstrator (RED) as a testbench to position a light source at precise angles. The research demonstrates the feasibility of developing a low-cost sun sensing solution for nanosatellites that balances simplicity with adequate performance for attitude determination in space applications. The findings contribute to the growing field of small satellite technology and offer potential pathways for future improvements in sun sensor design for space missions

1. Acknowledgements

We would like to express our sincere gratitude to our supervisors, Dr. Roberto Ramirez-Iniguez and Geraint Bevan, for their invaluable guidance and unwavering support throughout this project.

Our appreciation extends to the 3rd Floor EEE Lab Technicians and Dr. Carlos Gamio-Roffe, whose technical expertise and assistance were instrumental in the successful construction of the prototype.

We are particularly grateful to the European Project Semester RED Team members—Nikolay Ivanov Shopov, Stef Hannisse, and Samridhi Gupta—for generously allowing the use of their Renewable Energy Demonstrator as a testbench. Their contribution provided an ideal platform for positioning light sources during the testing phase of this project.

2. Introduction

2.1. Problem Statement

With the ever-increasing commercialization of the space and satellite industry there is a growing need for a cost-effective method of attitude tracking for smaller satellite missions of such as CubeSat as these missions are purpose built for very specific objectives. Whilst the larger commercial satellite missions make use of expensive digital camera systems for tracking purposes, this is not feasible for much smaller CubeSat setups. CubeSats are defined from 1 unit to 12 – where 1U is a 10x10x10 cm satellite. Consequently, there is a demand for a cost-effective and easily implementable attitude tracking system that can provide accurate measurements for CubeSat missions, such as a Position Sensitive Detector (PSD) using photodiodes.

2.2. Aim of the Project

"To investigate and develop a cost-effective and reliable sun sensing solution suitable for Low Earth Orbit (LEO) nanosatellite attitude determination."

2.3. Objectives of the Project

To investigate the design of a sun sensing system for nanosatellites, used in orientation determination, through detection of its relative position to the sun using analogue sensors located on the satellite's body. Our goal is to create a system which balances cost-effectiveness and simplicity. To achieve this, we will create a software model of the analogue sensor(s) to simulate the system's ability to track the sun from various angles in orbit. After which, we aim to build a physical prototype and use a movable light source to simulate the sun's movement, allowing comparison between the real sensor's performance against our simulations. Although the physical prototype will be built using non-space-grade materials, one of the objectives is to look at and analyse materials required for building a space-grade PCB and sensor. For this step, the Mechanical side of the team will perform Printed circuit board (PCB) and aperture device finite analysis using ANSYS to determine resilience to environmental factors such as stress and thermal simulation. The application of signal processing will be explored to provide usable data, filter out

noise, and improve the system's accuracy. This approach aims to develop a cost-effective and reliable, in-house sun sensing solution specifically for nanosatellites operating in Low Earth Orbit. Major Objective points:

- **Conduct literature review:**

- Analyse existing research on sun sensing technologies, with a focus on PSD-based analogue sensors and their applications in nanosatellites.
- Identify current challenges, best practices, and advancements in attitude determination in Low Earth Orbit. Use these insights to guide the design and optimisation of the proposed sun sensing system.

- **Develop software model:**

- Simulate the performance of the PSD-based analogue sun sensor in tracking the sun's position from various angles in Low Earth Orbit.

- **Design and fabrication of physical prototype:**

- Integrate analogue sun sensor components, test and validate its performance under controlled conditions.

- **Compare simulated and experimental results:**

- Establish evaluation methodology between simulated and experimental test results to ensure that topology evaluation is applicable.

- **Optimise sensor topology:**

- Research and evaluate various configurations of analogue sun sensing systems to maximise sun detection accuracy and minimise blind spots.

- **Investigate environmental factors:**

- Evaluate the material requirements of the PCB and aperture device.

- **Implement signal processing algorithms:**

- Investigate the filtering of noise to enhance the signal-to-noise ratio and otherwise ensure the acquisition of usable data for accurate sun position determination.
- Implement data handling which optimises scanning rates and efficiently processes the analogue signal data for real-time attitude determination.

- **Document results and overall cost-effectiveness:**

- Develop criteria for final evaluation of sun sensing systems, on which to base the final presentation of project findings.

3. Literature Review

3.1. CubeSat Design

Puig-Suari, Turner and Ahlgren published an IEEE paper in 2001 with the help of their students at California Polytechnic State University exploring a need for micro satellites for use by universities in an ever-expanding space programme. They provide as a solution a standard satellite form-factor that will bring down the cost of both manufacture and deployment of satellites by smaller entities: the CubeSat. The paper identifies a key component for the success of this form factor a need for a standard CubeSat deployer mechanism which can deploy several satellites safely and develop such a platform, called Poly Picosatellite Orbital Deployer or P-POD. They point out the need and provide microsatellite size and shape of the CubeSat form factor [8].

Sai balaji et al. performed a study using MATLAB simulation of several attitude control algorithms to look at the ability to control a CubeSat of size 1U. They also simulated sensors such as sun sensors, magnetometer, and gyroscope. They concluded that it is possible to operate the satellite using a magnetorquer type actuator and an array of mathematical models and algorithms: it would take 2000 seconds for a 1U satellite to stabilize at 505km, 98° degree attitude in orbit with the methods utilized by them [9].

Incentivised by the rapidly increasing use of LEO, Lopez-Calle and Franco perform a quantitative comparative study on the catastrophic failure of CubeSats and Nanosats from radiation exposure due to the harsh environment of space versus failure due to collisions in the increasingly busy Low Earth Orbit (LEO). The authors concluded that while sustained damage and damage protection from radiation exposure used to be and currently still is the most crucial factor in protecting LEO microsatellites, increasingly the risk of debris collisions is becoming more important and will become the most important in the following 50 to 70 years. The authors conclude that microsatellite designers need to move their focus more towards defence from debris impacts as these, even if not resulting in catastrophic failure of the satellite, they will impact the attitude of the satellite [10].

3.2. PSD Enabled Sun Sensor

The need for position estimation based on light sources preceded current requirements in microsatellites. Qian, Wang, Busch-Vishniac and Buckman describe in 1993 a position

sensitive detector (PSD) method using a single two-dimensional lateral-effect PSD capable of keeping track of multiple light sources at the same time. Their method of tracking multiple light sources involves modulating each light source, LEDs, of interest to a different frequency. They succeeded in correctly tracking two light sources modulated at 10kHz and the other at 5kHz. They point out that the light sources can correctly be tracked even in the presence of background light and that several light sources can be tracked, with the only restrictions being the bandwidth of the PSD and the sampling time of the sample and hold device being used. Although their method of tracking several lights might be out of scope of a sun sensor, the methods of design may be of some use to the design of a PSD.

Similarly, Guanghui and his colleagues developed the AirLink-E100 system which makes use of a PSD and describe it in a 2007 paper. This system is not directly related to sun sensing; however, their finding may be of some use. In a position sensor on earth, they point out, due to background light the PSD does not work with the precision necessary. They introduce methods of achieving better precision using analogue and digital signal processing. They modulate the light to be sensed, in this instance a laser, with a square wave, in essence turning the laser on and off. This allows for sensing of the background noise (when the laser is off) and subtracting the noise from the next time period when the laser is on. The authors conclude that this is a sound method of filtering out background noise in PSD devices where the light source can be modulated[11].

Building on this and other work, Ortega, Lopez-Rodriguez, Ricart et al. describe in a 2010 paper a miniaturized two axis sensor with a $\pm 60^\circ$ FOV, totalling 120° , and angle accuracy better than 0.15° . Their method is directly aimed at sun sensing. They not only design, fabricate and characterize the sensor, but also successfully integrate it in a Spanish nano-satellite NANOSAT-1B. The team used monolithically integrated silicon photodiodes in a crystalline silicon substrate protected by a glass cover. The entire size of the sensor is $3\text{cm} \times 3\text{cm}$ and weighs in at just 24 grams. The NANOSAT-1B which launched in 2009 contained three of these sensors[12].

Ortega et al.'s research directly aligns with the paper's objectives, rendering their work highly relevant, and was replicated by Dwik and Somasundaram's research modelling and simulating a PSD using MATLAB[13]. The authors point out the superiority of PSD over Charge-Coupled Devices (CCDs) because of the higher resolution and rapid response time. The researchers modelled two-dimensional photodiode arrays providing four output currents using included photodiode element in MATLAB. They also modelled simplified versions with single diode and one-dimensional array. They conclude that a PSD using photodiodes is a viable method of detecting the location of a light source from the current change readings of the diodes. They point out that for use in a fully working Acquisition, Tracking and Pointing (APT) system, further work is necessary that is not covered in their paper, such as signal conditioning.

Furthering the previous work, Delgado et al. showed in a 2013 paper the design, fabrication and characterization of a solar sensor that takes advantage of subdivision of the field of view (FOV) into 4 quadrants. The research team was able to develop high-precision sensors with an FOV of $\pm 60^\circ$ and fine resolution of 0.05° while providing a coarse resolution of 0.5° . This high precision is achieved by splitting the sensor into four sub sensors, each concentrating on a different range of angles. They named the technology Sensosol and the paper states it will be used onboard the SeoSat satellite from Inta corporation[14].

3.3. Mechanical Design and Analysis

Although the CubeSat specifications are strict in reference to size and shape of nanosatellites that aim to respect these specifications, it allows the freedom for designers to choose many characteristics. Therefore, mechanical analysis is required, with a focus on thermal and structural characteristics. To this end, Ullah, Rehman, Bari and Reyneri published a paper in 2017 raising the struggle of heat dissipation in CubeSats due to their small size not allowing the installation of heat-dissipating radiators. The team considered all thermal resistances of the CubeSat panels either as separate layers or similar materials combined in their simulation. They conducted both simulation and real measurements of the AraMiS-C1 satellite developed by the Torino Polytechnic and found that the simulated model correctly aligned with real world measurements. They conclude that their model can therefore be used to model any microsatellite following the CubeSat standard. They also concluded that the thermal resistance measured was exceptionally low and therefore could be safely used on satellites to be deployed[15].

Similarly, Raslan, Michna and Ciarcia performed a thermal simulation of a CubeSat in a 2019 paper. Their goal was to discover the required framework to maintain the thermal stability of a CubeSat in orbit, with the overarching goal of creating a CubeSat that will contain a mammalian tissue sample for gathering experimental data of the effects of microgravity and space radiation on the sample. The CubeSat, therefore, must be able to maintain a very narrow range of temperatures without large fluctuations so that the experiment remains valid. They aim to find external coating materials and attitude control that limit these fluctuations. The mission will involve a 6U sized CubeSat containing a biohousing. With a black-chrome plated metal coating in combination with solar panels on the sun exposed side the team was able to maintain 37 degrees with only 2 degrees deviation in the biohousing. The other sides of the satellite all were covered in solar panels in the simulation. They conclude that they have identified a suitable single-coat material that in combination with PID attitude control algorithms is able to maintain the temperature within an admissible range. They also conclude that this can be done for a wide range of orbits and exposure time. They point out future research will focus on finding

other coat types that maintain temperature without attitude control as changing attitude to maintain temperature consumes satellite power, which is a valuable resource[16].

Concentrating on the structural resistance of the CubeSats, Dhariwal, Singh and Kushwaha performed a structural analysis using ANSYS software and released their findings in a 2023 paper analysing the structural behaviour of 1U CubeSats under various loads, static, modal random vibration, and shock loads. The team claims their paper establishes a methodological framework for CubeSat structural analysis and can be used for future work. They conclude that because the stress applied to the aluminium alloy 6061 used did not go above the yield strength, it is safe to assume the material operated safely and can be used on the CubeSats structure. They also conclude that their analysis was accurate and point out a need for a physical test on a real 1U CubeSat structure[17].

This review of carefully selected research papers serves as a robust foundation for the work that is to be conducted in this project. The critical insights and methodology described for thermal and structural analysis of CubeSats by the teams' previous work, as well as CubeSat designs and PSD sensor work conducted as mentioned above, is important and helpful for the project's successful contribution to this research.

3.4. Photodiode Simulation and Signal Analysis

A paper by Fuada et al. released in 2017 investigates the received power characteristics of commercially available photodiodes used as receivers in Visible Light Communication (VLC) systems with a line-of-sight (LOS) channel. The authors used MATLAB simulation to analyse how various parameters affect the received power, including:

- Transmitter semi-angle (half power)
- Distance between transmitter and receiver
- Room size
- Receiver Filed of View (FOV)
- Optical filter gain
- Lens refractive index

They also point out 6 key considerations when choosing a photodiode for VLC applications:

1. Surface area: A larger surface area (e.g. 10mm x 10mm) can support mobility in the VLC system, but this needs to be balanced against the impact on cut-off frequency and susceptibility to ambient light noise.

2. Generated short current: the photodiode should generate sufficient current ($>100\ \mu\text{A}$) when exposed to light, as this affects the required gain and bandwidth of the amplifier circuit.
3. Wavelength detection capabilities: The photodiode needs to be sensitive to the visible light spectrum (380nm to 780nm) for VLC applications.
4. Cut-off frequency: A high cut-off frequency (in the GHz range) supports high-speed data transfer, but this often requires sacrificing a larger surface area.
5. Rise time: Fast rise time (in the nanosecond range) is also desirable for high-speed VLC, but again this trades off with surface area.
6. Dark current and junction capacitance: The photodiode should have low dark current and low junction capacitance to minimize noise and maximize response time.

The paper notes that it is difficult to find a single commercially available photodiode that optimizes all 6 of these factors simultaneously. This often requires making trade-offs or using custom photodiode designs for the specific VLC application. The results show that factors like distance, room size, FOV, and LED power have a linear relationship with the received power at the photodiode. Additionally, the optical filter gain and lens index play an important role in determining the received power characteristics. The authors note that this study was limited to the LoS channel and does not take into consideration indirect illumination[18].

Nathanael A. Fortune writes a paper in 2021 meant to help scientists with common signal processing tasks when handling experimental data. The paper provides examples of using the Numerical Python (Numpy) and Scientific Python (SciPy) packages, as well as interactive Jupyter Notebooks, to accomplish tasks such as interpolation, smoothing, propagation of uncertainty, curve fitting, plotting functions and data, and determining the goodness of fit. The goal is to enable an interactive, exploratory approach to data analysis while ensuring the original data is freely available and the resulting analysis is readily reproducible. The paper includes sample Jupyter notebooks containing the Python code used to carry out these tasks, which can be used as templates for analysing new data[19]. This paper should prove useful in the numerical simulation of the PSD sensor.

3.5. IoT Communication Enhancement with LEO Satellites

The use of LEO satellites, including microsatellites, can be extended to the current expansion of Internet of Things (IoT) devices. To this end, Koukis and Tsoussidis investigated

the use of satellites for IoT sensors and devices. They created simulations using OM-NeT++ software and real sensor data from Smart Santander testbed. They concluded that their initial hypothesis was correct, and an increase of LEO satellites is inversely proportional to ping loss and round-trip time of packets sent between a LEO constellation and IoT equipment on the ground. They also concluded that the placement of ground stations led to improved communication even with a lower number of satellites. They close by saying they did notice instances where signal was deteriorated, and that further work is necessary[20].

4. Methodology

4.1. Prototype Development

4.1.1. Lifecycle

This section provides an overview of the Prototype Development Lifecycle.

Conceptualization and Requirements Definition

- The prototype must have four photodiodes in an xy pattern with respective circuitry required to output 0-5 Volts that will be read by an Arduino based DAQ. The circuit must be able to react to light intensity changes, however the change will be at low frequency (below 1Hz) as a satellite attitude is considered to change only gradually.
- While the prototype may not have a high accuracy, it is hoped that it will be enough to measure light position changes roughly, even if at a low accuracy of 10° or 20° but this will remain to be seen.
- The prototype within the scope of this paper will show the ability to detect the position of light at normal room conditions, therefore it does not need to withstand temperature changes or radiation that a final product would require if deployed in space.
- Interface requirements: the prototype electrical output needs to be compatible with the Arduino Analog to Digital Converter (ADC) input. Therefore, the signal shall not go below 0 Volts or exceed 5 volts.
- Size and weight are not of high importance, but the device must fit in the testing equipment, which is the Renewable Energy Demonstrator arch. Preferably a height not higher than 5cm.

Theoretical Design

The prototype will be composed of three parts: a stripboard containing the components for the amplification circuit, a 3D printed Photodiode enclosure which will allow placing the photodiodes in the correct positions, as the photodiodes have the legs at a smaller

distance than the stripboard, and need to be placed quite close to eachother, with the third part being the Arduino based DAQ. The third part to the

Sun Sensor Geometry and Aperture Design was decided to be in an orthogonal, T shape, providing an x-y layout with two photodiodes in the x direction and two photodiodes in the y direction. Combined with an aperture design that covers one half of each diode, as represented in Figure 4.1. This configuration is similar to Ortega et al. in their paper attempting to miniaturize a two axis Sun Sensor[12].

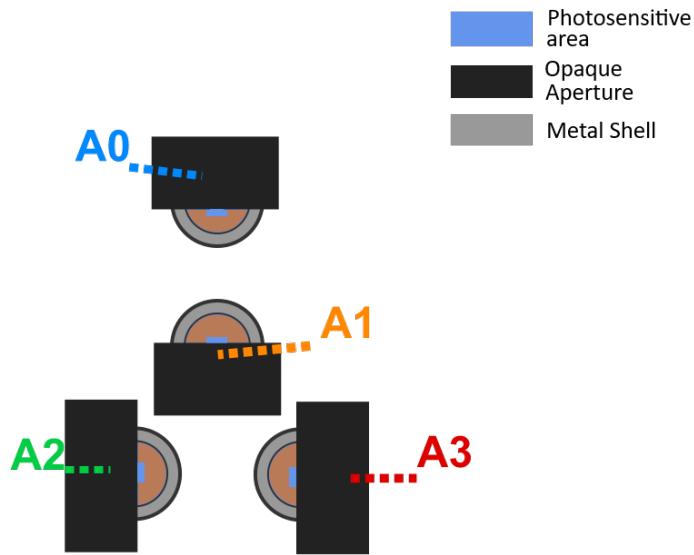


Figure 4.1.: Diagram of Photodiode Array with Apertures

Component Selection

The components chosen for the design of the prototype are detailed below. They were first placed on a BreadBoard and tests were performed to ensure the functionality of the circuit before moving them to a stripboard which would ensure the components are physically more stable and reduce the chance of faulty connectoins. Therefore, the same components were used for both BreadBoard and StripBoard prototyping. The circuit design and testing is gone into more detail in Section 4.1.2.

Photodiodes were researched and several options were found, most of which were quite expensive, such as 2D PSD type sensors like the Hamamatsu S5990 but were both prohibitivly expensive and Surface Mount Device style (SMD), which would have been harder to prototype but would allow for much higher resolutions, while also complicating the

Table 4.1.: Electrical Components Used in Amplification Circuit

Component Type	Comp. Part Name	Component Value	Amount
Op-Amp	TI LM324-N	—	2
Photodiode	Hamamatsu S5971	—	4
Resistor	—	1 MΩ	4
Resistor	—	150 kΩ	4
Resistor	—	10 kΩ	4
Capacitor	—	1 μF	4
Stripboard	Veroboard	—	1
Screw terminal block	—	2-input	3

Table 4.2.: Components Used for DAQ System

Component Type	Component Part Name	Component Value	Amount
Microcontroller	Arduino Uno	—	1
Cable	USB cable	—	1
Computer	Laptop/Python	—	1

project due to the more complex nature of PSDs. A decision was made to base the project on 1D photodiodes, and four Hamamatsu S5971 were purchased, which offered a good compromise in price and specifications:

Table 4.3.: S5971 Photodiode Specifications

Parameter	Value
Spectral response range (λ)	320 to 1060 nm
Peak sensitivity wavelength (λ_p)	920 nm
Photosensitivity S (A/W) at λ_p	0.64
Photosensitivity S (A/W) at 780 nm	0.55
Photosensitivity S (A/W) at 830 nm	0.6
Short circuit current I_{sc}	1.0 μ A
Dark current I_d (Typical)	0.07 nA ^{*3}
Dark current I_d (Maximum)	1 nA ^{*3}
Cutoff frequency f_c	0.1 GHz ^{*3}
Terminal capacitance C_t ($f=1$ MHz)	3 pF ^{*3}
Noise equivalent power NEP ($V_R=10$ V, $\lambda=\lambda_p$)	7.4×10^{-15} W/Hz ^{1/2}

$$V_R = 10 \text{ V}$$

Although the higher versions such as S5972-3 have better specifications, such as frequency cutoff of 1GHz and lower Dark current, these were not needed for our project, higher frequency cut off not needed due the static nature of the light source and dark current, while it could affect a case where one of the diodes is fully in the dark, a voltage offset would be noticeable, but with a voltage resolution restricted by the Arduino DAQ

to 4.88mV/step, it was deemed acceptable:

$$\begin{aligned}
 V_{\text{offset-TIA}} &= I_d \times R_f \\
 &= 0.07 \text{ nA} \times 1 \text{ M}\Omega \\
 &= 70 \mu\text{V}
 \end{aligned} \tag{1}$$

$$\begin{aligned}
 V_{\text{offset-total}} &= V_{\text{offset-TIA}} \times \text{Gain}_{\text{post-amp}} \\
 &= 70 \mu\text{V} \times 16 \\
 &= 1.12 \text{ mV}
 \end{aligned} \tag{2}$$

Equation 2 above shows the final dark current would be a maximum of 1.12mV which is below what the ADC can detect.

Operational Amplifier (OpAmp) choice was once again not a complicated choice due to the same arguments mentioned above re. photodiode selection: low frequency signal and reduced DAQ resolution. The LM324-N is a low-cost OpAmp which provides acceptable performance. The advantages in choosing this OpAmp is its ability to function

Table 4.4.: Operational Amplifier Specifications

Parameter	Value
DC Voltage Gain	100 dB
Unity Gain Bandwidth	1 MHz
Supply Voltage Range (Single)	3 V to 32 V
Supply Voltage Range (Dual)	$\pm 1.5 \text{ V}$ to $\pm 16 \text{ V}$
Supply Current	700 μA
Input Bias Current	45 nA
Input Offset Voltage	2 mV
Input Offset Current	5 nA
Input Common-Mode Voltage Range	Includes Ground
Differential Input Voltage Range	Equal to Supply Voltage
Output Voltage Swing	0 V to $V^+ - 1.5 \text{ V}$

Internally frequency compensated for unity gain

Temperature compensated

on a single pole Power Supply, it is rather cheap while still offering 1MHz Unity Gain Bandwidth and is recommended for DC Gain which the type of signal our project aims to amplify. For example, the slew-rate characteristic is not mentioned on the datasheet because it is aimed at low frequency operation.

BreadBoard Testing

Once the circuit design was completed as seen in Figure 4.6, the components were placed on a BreadBoard to test the real circuit, as seen in Figure 4.2. There were several

iterations, at first with only the Transimpedance Amplifier (TIA) and one photodiode- a design that when tested, resulted in a low Voltage output when testing with only the Light Emitting Diode (LED) light from the RED testbench. This is due to the low light power of LEDs. A decision was made to add a secondary amplification circuit which raised the Voltage to a maximum 5V as designed. Further details on design in Section 4.1.2. The final BreadBoard design can be found in Figure 4.2.

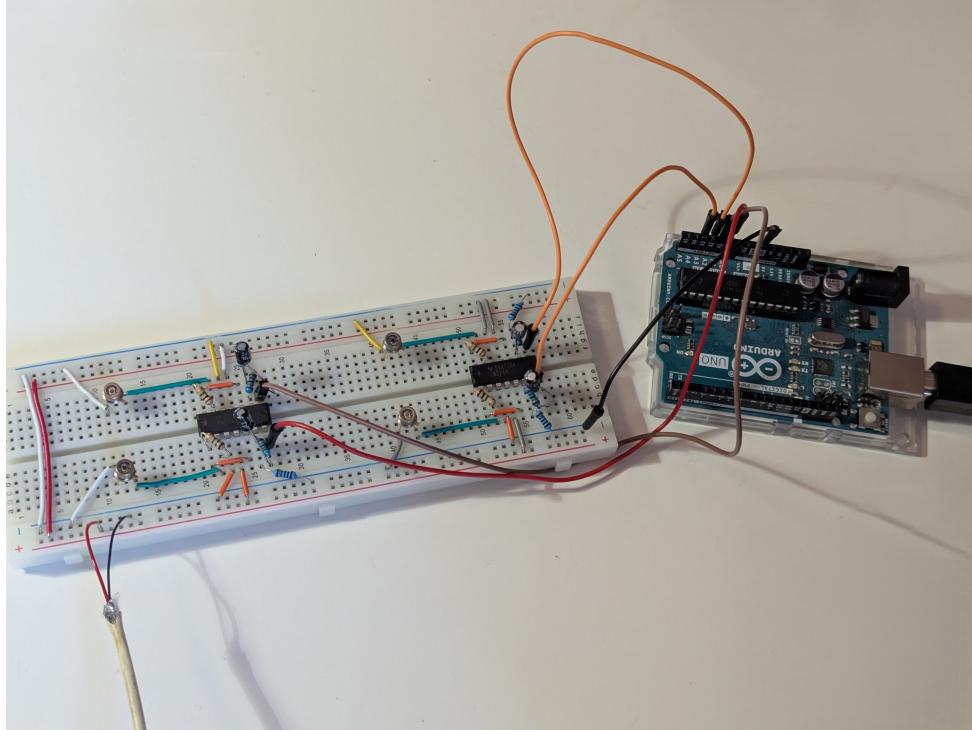


Figure 4.2.: Photo Of BreadBoard Prototype

Renewable Energy Demonstrator (RED)

The RED device borrowed from the EPS team [21] had to be modified to be used as our testbench providing illumination from desired angles in a repeatable fashion. Details on the modification can be found in 4.3. After modifying it to move the light in our desired location on the hemispheric plane both azimuth and elevation angles could be controlled, it was discovered that the servomotors in the RED were not always delivering the correct angles as requested in the code. It was discovered that "helping" the arch move towards the desired location, such as 90° from 0° by pushing with our hand, allowed the arch to move to the correct position. The reasons for this could be several, such as servomotors with insufficient torque, or damaged gear train. Due to time limitations it was decided not to further debug the device, instead a decision was made to use a protractor and manually "help" the arch when needed to reach the required angle. A further issue with the RED device was that it would "randomly" change the location, which was only meant to be triggered by a button press. The reason for this was likely the same reason our recorded

signals had a lot of noise: interference from the RED power source. However this was rare enough that it did not impede our testing. A debouncing technique in software could likely have fixed this issue, if it was going to be fixed. Further, while overall the RED was very useful for our project, the servomotor issues causing the arch to not reach the correct angle easily and further affected by some control deadband - an inability of the servomotors to reliably make small changes in angle, meant that certain tests could not be performed, such as taking measurements at every 5° in order to record readings across the whole hemispere, which would allow creating a Look-Up Table (LUT) that would be used to correctly read the location of the light.

Design Refinement

Secondary Amplification was added to the circuit which was a major redesign for signal conditioning. This allows reading of the analog signal with the Arduino ADC, at the full range of 0V to 5V, as the Arduino ADC is 10 bit. This gives a resolution of $2^{10} = 1024$ steps, otherwise the readings would be from 0 V to about 300 mV, with a much lower resolution between steps - the DAQ would be using a resolution of only the region covered between 0V and 0.3V, ie. $1024/16 = 64$ steps. Details regarding this can be found in Section 4.1.3. Further, the Secondary Amplifier circuit was modified with the addition of a feedback resistor to reduce noise, as described in Section 4.1.3.

The Photodiodes were first placed in a square pattern around the circuit, which helped BreadBoard prototyping and would have also simplified the Stripboard design. However, it was quickly determined that the photodiodes must be as close to each other as possible, due to the light in the RED testbench being closer to a "single point" spreading out, as opposed to the light coming from the sun, which is assumed to be parallel - in which case the distance relative between the photodiodes would not matter. More details about the lightsource on the RED testbench in Section 4.3 . This meant that soldering the photodiodes to the stripboard was not an option, due to the small distance between the photodiodes, the diodes having 3 pins (anode, cathode and case) at small distances in triangle pattern and the relatively large distance between strips on the Veroboard stripboard (2mm). The decision was therefore made to move the photodiodes to their own 3D printed enclosure, which went through its own redesign as described under Section 4.1.4. This meant soldering wires to the photodiode pins and ensuring they do not shortcircuit using heat shrink tubing.

The **Stripboard Prototype Development** was done in two stages to avoid making too many mistakes that would be too hard to fix: the design was first implemented using the Software Fritzing which is a free and Open Source Software suite for Electronic Design Automation (EDA) that allows planning on different BreadBoards and Stripboards. The

final CAD Stripboard design is reproduced in Figure 4.3. When soldering the components to the actual stripboard, besides some errors such as soldering a jack on the wrong stips, and having to de-solder and re-solder, the stripboard prototyping was quite straight forward due to having tested the design on the BreadBoard and creating the plan in Fritzing as mentioned. However, during testing, the readings seemed off and upon further debugging it was found that some flux remained on the stripboard, causing unwanted conductance between strips, creating a parallel resistance with one of the feedback resistors, which was causing the incorrect amplification. After cleaning the stripboard with support from Dr. Carlos, who had solvent in his office, the Stripboard prototype functioned as intended.

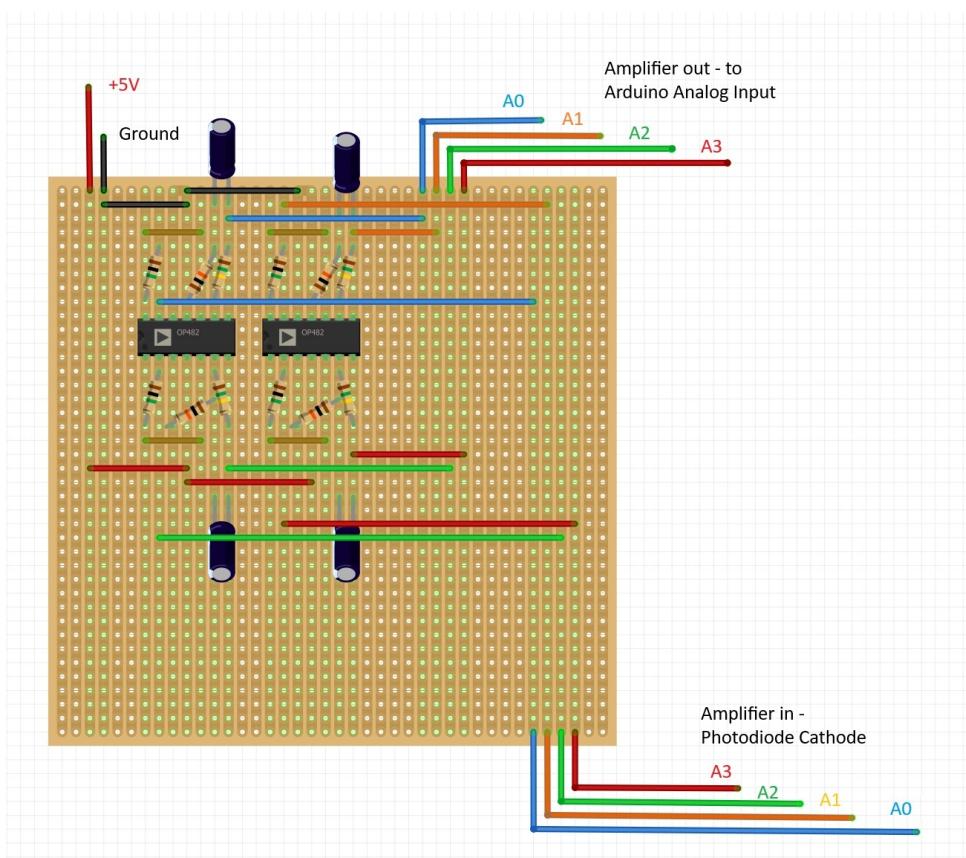


Figure 4.3.: Photo Of Stripboard Prototype on Fritzing

The **Aperture** on the 3D printed photodiode enclosure was harder to implement. Initially a 3D printed aperture was planned but this was not able to be produced. Another plan was to glue plexiglas to the enclosure and apply black electrical tape to it, an idea which was changed because the plexiglas available was considered too thick at 5mm. The worry was that light scattering in the plexiglas, refraction and especially lateral displacement due to the thickness, especially at a larger angle such as when the light is close to the horizon tangent point. The difference in displacement between zenith and when close to horizon was a worry. A solution was found by purchasing a screen protector glass for



Figure 4.4.: Photo Of Stripboard Prototype

mobile phones. This glass was very thin, at under 1mm, and is fabricated to be very clear and transparent so as not to affect the quality of the mobile phone screen it is meant to protect. The black electrical tape was then placed on top of this glass as seen in Figure 4.5. As can be seen the apertures are not fully straight. This is because they were placed by hand.

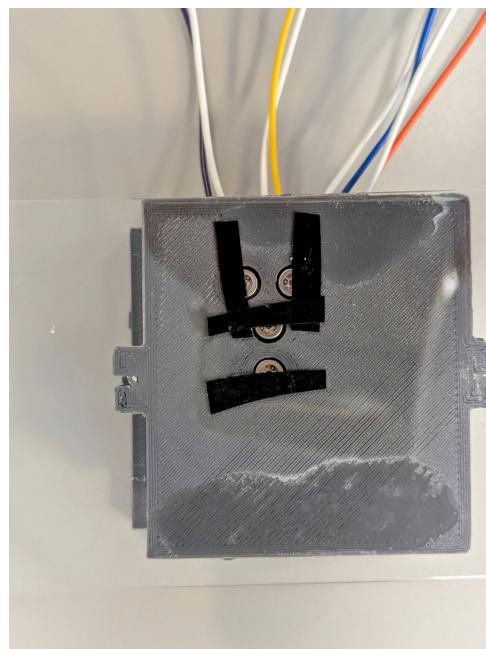


Figure 4.5.: Photo of aperture placement on screen protector glass

4.1.2. Signal Conditioning Circuitry

Photodiodes produce a certain amount of current when light hits the depletion region. Therefore, a larger depletion region is desirable, to capture more light and in turn produce more current. For this purpose the photodiode in our circuit is reverse-biased as can be seen in Figure 4.6 [22, p.155].

Transimpedance Amplifier (TIA)

A reverse-biased photodiode allows a current to flow from the cathode to anode which is connected to ground. Adding a resistor in series with the photodiode and measuring the voltage across the resistor would be a form of I-to-V conversion, however it would distort the reading due to non-zero impedance due to the input current [23, p.233]. This current is instead converted to a Voltage using a TIA with the following relationship derived as in Appendix Figure A.1:

$$V_{\text{out}} = -I_{\text{ph}} \cdot R_f \quad (3)$$

$$I_{\text{ph}} = P \cdot R_{\lambda} \quad (4)$$

Where P is Light Power (W) and R_{λ} is Responsivity (A/W).

$$V_{\text{out}} = -(P \cdot 0.5 \text{ A/W}) \cdot 1 \text{ M}\Omega \quad (5)$$

The TIA circuit makes use of an OpAmp as seen in Figure 4.6 that provides very high input impedance ($1\text{G}\Omega$) and allows the amplification of the signal without disturbing the photodiode current, therefore not affecting the readings. The gain of the circuit is simply $A = -R_f$ [23, p.535]. The inverting input is used in this configuration, which converts the negative current flowing from the cathode to the anode of the photodiode, into a positive voltage. A TIA usually requires a feedback capacitor in parallel with the feedback resistor, this is because the photodiode capacitance forms a low-pass filter with the feedback resistor. However, for our purpose this was not necessary due to the DC-like signal. The low-pass filter forming for our $1\text{M}\Omega$ resistor and 3pF terminal capacitance of the photodiode:

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot 1 \text{ M}\Omega \cdot 3 \text{ pF}} = 53,051 \text{ Hz} \quad (6)$$

This shows a low-pass filter of around 53kHz is formed, which can be ignored for our DC current.

An OpAmp circuit introduces a **phase-shift**, and with two OpAmps the phase-shift would be disturbed even more in an AC circuit. However as the signal we are dealing with is DC, phase shift can be ignored, it would just show up as a certain amount of delay in the signal amplitude change when the light position changes.

4.1.3. OpAmp Noise

Operational Amplifiers introduce some noise that must be taken into account. Previously we discussed the photodiode dark current, which was decided could be ignored as it was below our ADC resolution of 4.88mV.

The **Short Circuit Noise Current** at input can be calculated as such (at 298K) [23, p.439]:

$$\begin{aligned} i_{n,thermal} &= \sqrt{\frac{4kT}{R_f}} \\ i_{n,thermal} &= 1.28 \times 10^{-10} \cdot R_f^{-\frac{1}{2}} \\ i_{n,thermal} &= 0.128 \text{ pA}/\sqrt{\text{Hz}} \end{aligned} \tag{7}$$

This would give an input noise of 0.128pA at 1Hz.

The **Johnson-Nyquist Noise** at input from the feedback resistor is:

$$\begin{aligned} e_n &= \sqrt{4kTR} \\ e_n &= \sqrt{4 \times 1.38 \times 10^{-23} \times 298 \times 1 \times 10^6} \\ e_n &= \sqrt{1.64 \times 10^{-14}} \\ e_n &= 1.28 \times 10^{-7} \text{ V}/\sqrt{\text{Hz}} \\ e_n &= 128 \text{ nV}/\sqrt{\text{Hz}} \end{aligned} \tag{8}$$

Multiplying the Johnson by the gain of the circuit:

$$\begin{aligned} e_{n,out} &= e_n \times \text{Gain} \\ &= 128 \text{ nV}/\sqrt{\text{Hz}} \times 1 \times 10^6 \\ &= 128 \text{ nV} \times 10^6 \text{ nV}/\sqrt{\text{Hz}} \\ &= 128 \text{ mV}/\sqrt{\text{Hz}} \end{aligned} \tag{9}$$

We see that the noise is dominated by the Johnson Noise of 0.128V. While this is significant, in real test results it did not affect our readings, presumably because we added both a feedback capacitor on the secondary amplifier and further digital low pass filtering after sampling the signal.

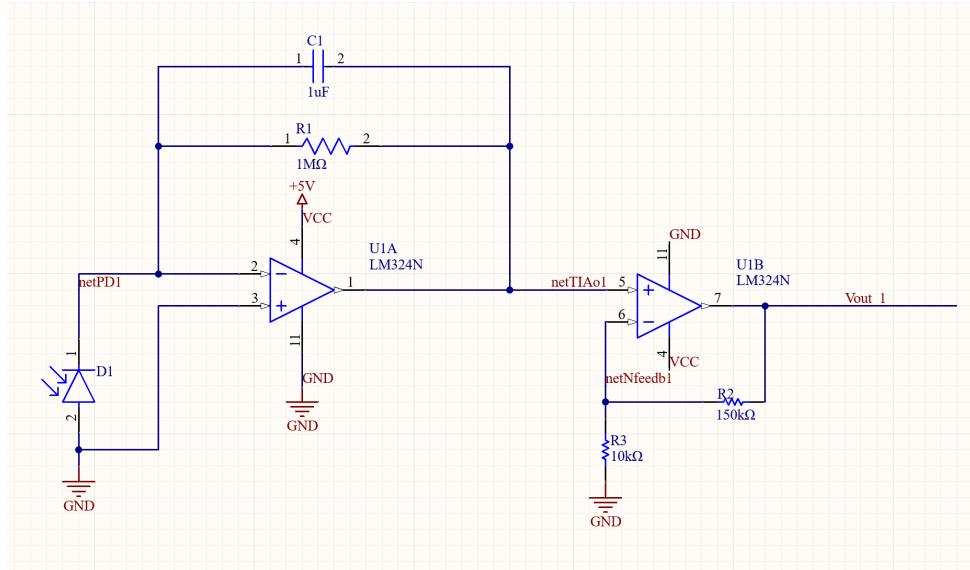


Figure 4.6.: TIA and Post Amplification Circuit in Altium Designer

Secondary Amplification

Testing showed that even using a $1M\Omega$ resistor, the output Voltage was too low (around 310mV) at our RED testbench' LEDs maximum brightness, as explained in Section 4.1.1. To raise the maximum Voltage to the desired maximum of the ADC of 5V, a higher feedback resistor could be used, however this would introduced noise and would require more complicated TIA with feedback capacitors. Due to the LM324-N having 4 OpAmps, the decision was taken to implement a Secondary Amplification circuit. The non-inverting OpAmp configuration was chosen to maintain the voltage positive, which also means there is no need for a dual power supply and keeps the Voltage positive for the Arduino ADC. A simple calculation was made to figure out the required Gain of the circuit:

$$A = \frac{\text{required Voltage}}{\text{measured}} = \frac{5 \text{ V}}{0.31 \text{ V}} = 16.1 \quad (10)$$

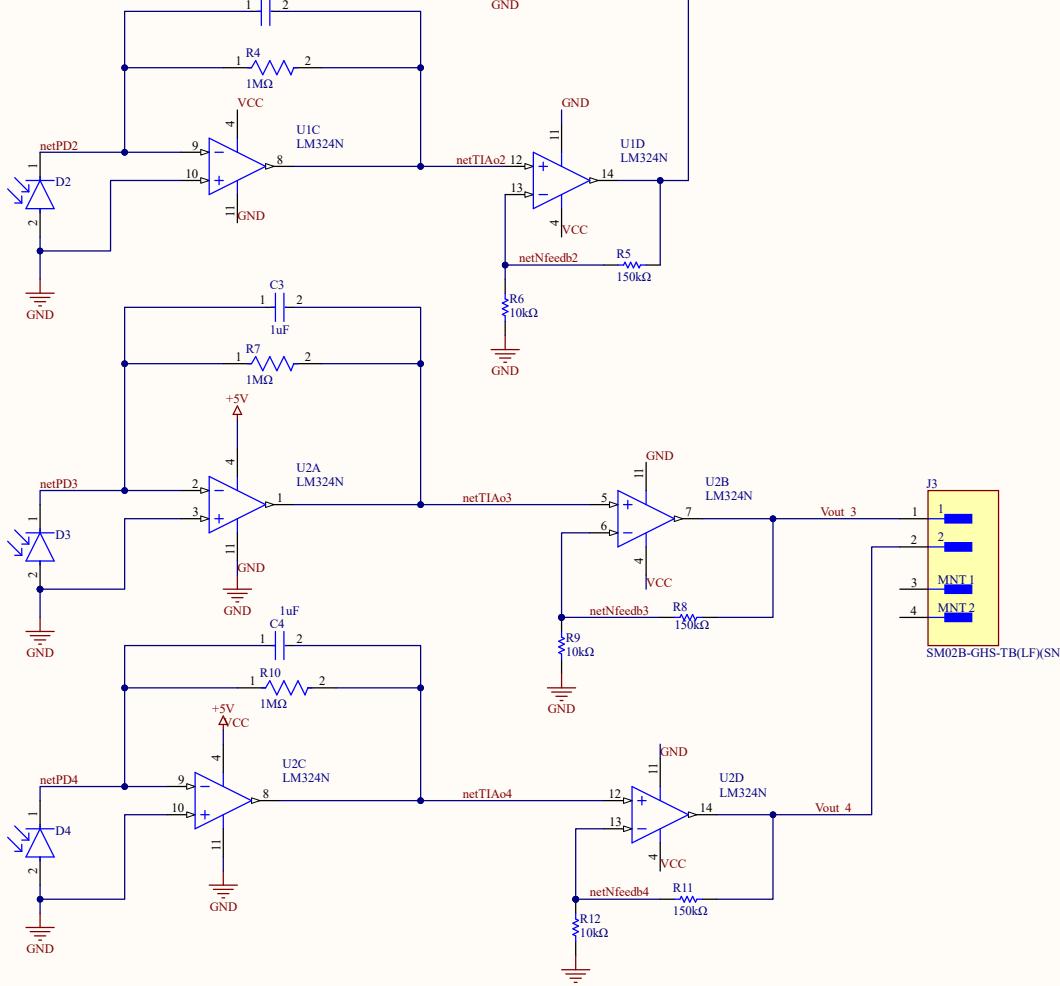
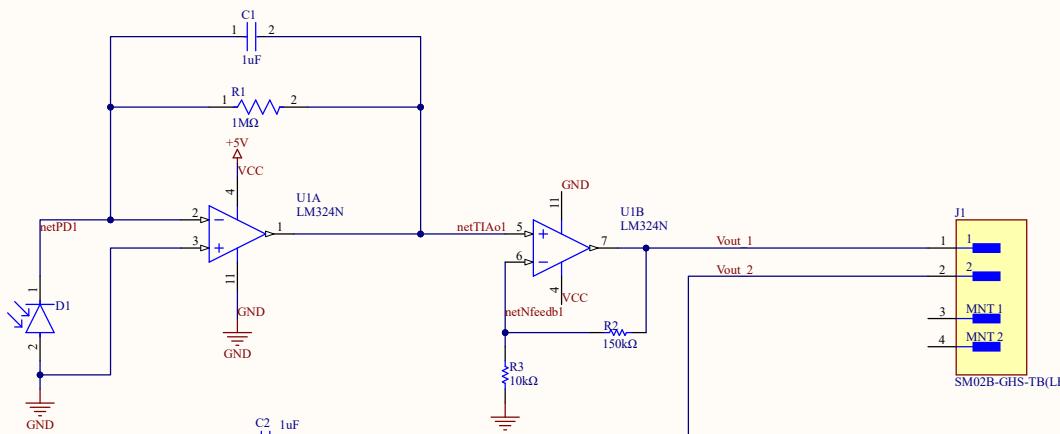
Knowing the gain required, the feedback resistor was calculated by choosing a $10k\Omega R_1$ and rearranging the gain equation:

$$\begin{aligned}
A &= 1 + \frac{R_f}{R_1} \\
16 &= 1 + \frac{R_f}{10 \text{ k}\Omega} \\
16 - 1 &= \frac{R_f}{10 \text{ k}\Omega} \\
15 &= \frac{R_f}{10 \text{ k}\Omega} \\
R_f &= 15 \times 10 \text{ k}\Omega \\
R_f &= 150 \text{ k}\Omega
\end{aligned} \tag{11}$$

This provides a gain $A = 16$ which is very close to the Gain required in Equation 10. Further it must be stated that the resistors used have a tolerance of 10% - therefore the actual final gain will fluctuate by that much. Once the design was tested on a BreadBoard, it was transferred to a stripboard as pictured in Figure 4.4. Later in the design during testing, a decision was made to add a $1\mu\text{F}$ capacitor in parallel with the feedback resistor of the Secondary Amplifier. This creates a low-pass filter on the output as seen in Eq. 12.

$$\begin{aligned}
f_c &= \frac{1}{2\pi RC} \\
f_c &= \frac{1}{2\pi \cdot 150 \text{ k}\Omega \cdot 1 \mu\text{F}} \\
f_c &= \frac{1}{2\pi \cdot 150 \cdot 10^3 \cdot 1 \cdot 10^{-6} \text{ s}} \\
f_c &= \frac{1}{2\pi \cdot 150 \cdot 10^{-3} \text{ s}} \\
f_c &= \frac{1}{0.942 \text{ s}} \\
f_c &= 1.061 \text{ Hz}
\end{aligned} \tag{12}$$

This does mean that we are now restricting the design to not be able to show Voltage change rates at higher than 1Hz, and testing with a moving light source will have to be restricted to a frequency at least half of 1Hz. Otherwise, the rate of change of Voltage will appear gradual and not represent the real signal.



Title: Analogue 2D light sensor		
Size A3	Number	Revision 1
Date: 4/16/2025	Sheet of	
File: D:\Documents\prototype.SchDoc	Drawn By:	Alexandru Belea

4.1.4. Enclosure Design 3D print

This section provides an overview on the design and fabrication of the enclosure for the prototype. The enclosure is a critical component that houses the electronic components and provides protection against environmental factors. The design process involves several steps, including conceptualization, modeling, and fabrication.

Conceptualization

The design of the testing rig aperture underwent several iterations to optimize its performance and manufacturability. This iterative process approach allowed for the systematic refinement

Initial Concept Development An initial sketch design of the aperture consisted of a rectangle geometry “T-shaped” array of holes for the photodiodes to sit in with the main body being 105 mm long, 114 mm wide, and 1.5 mm gap in-between for a stripboard to be fitted in, these initial sketches can be seen in Appendix E.

CAD Software Selection and Workflow Fusion 360 was employed for the CAD of the testing rig aperture, specifically for fabrication via 3D printing. Its capabilities in parametric modelling and integrated CAD/CAM workflows facilitated precise design execution and a streamlined transition to additive manufacturing, ensuring compliance with functional specifications. The software’s simulation tools also enabled design optimisation for effective performance under testing conditions.

Design Iterations and Refinements

Initial Design For the initial design first made an outlined shape of the box with width measurements of 115mm x 114mm and thickness of 1.5mm as can be seen in Figure 4.7. For the photo diode array used hole tool provided in fusion 360 and based the measurement values on the manufacturers datasheet for the photodiodes used for this project.

Iteration 1 Following the initial design sketches and CAD model, the first iteration of the enclosure was created as can be seen in Figure 4.8. The design focused on ensuring that the photodiodes were securely mounted and that the enclosure provided adequate protection for the electronic components. Iteration 1 involved a design refinement to facilitate modularity and interchangeability of photodiode array segments. Fusion 360’s pattern tool was employed to replicate the photodiode hole arrangement across four distinct segments, ensuring consistent alignment and ease of component replacement.

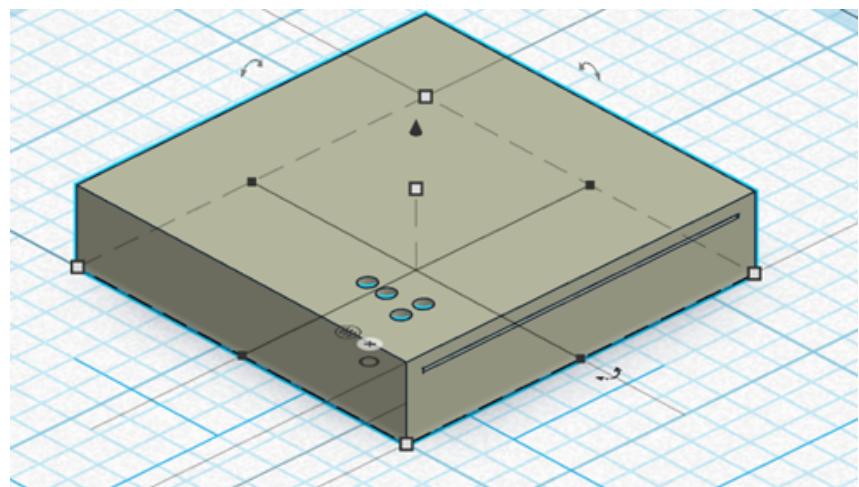


Figure 4.7.: First Design Iteration of the Enclosure

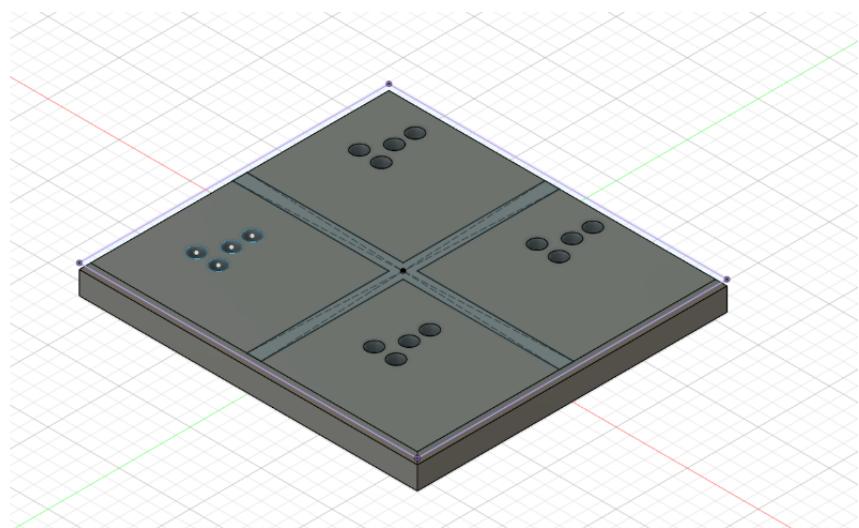


Figure 4.8.: Second Design Iteration of the Enclosure

Iteration 2 In iteration 2 reduced iteration 1's size down to one of the segments which measured out at 52mm length, 57mm and 10mm height with the focus being the housing for the photodiode array, shown in Figure 4.9

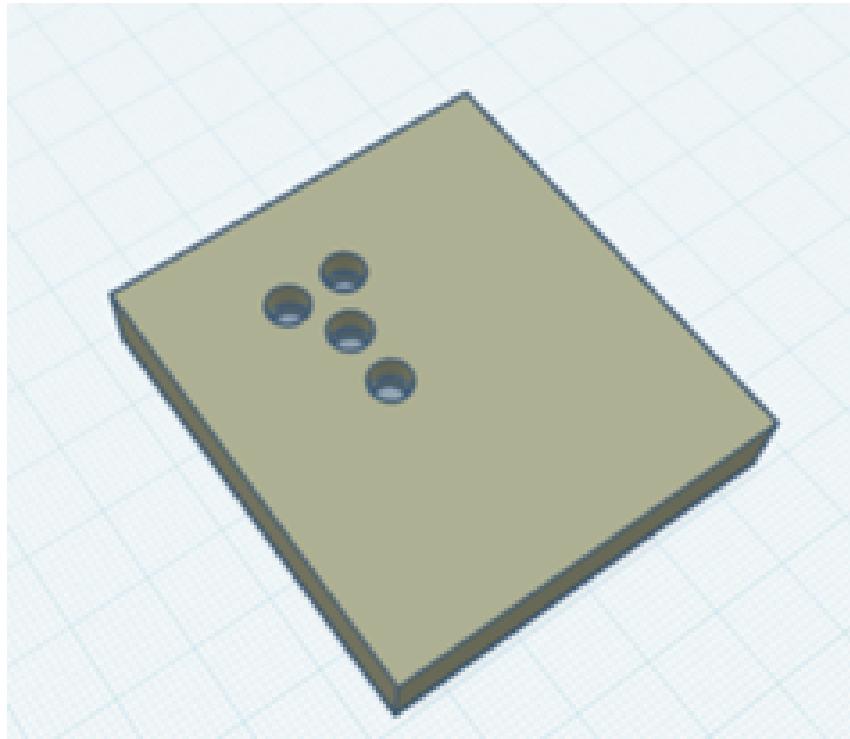


Figure 4.9.: Third Design Iteration of the Enclosure

Refinements To address dimensional incompatibility from previous iterations, the CAD model was revised. In iteration 3, the extruded height was increased by [insert height dimension] to achieve a proper fit. This adjustment was made while maintain the ease of fabrication and assembly. The final aperture design as depicted in Figure 4.10 while also utilizing a rail system such as a T rail to allow, incorporates the necessary dimensional adjustments and structural refinements identified through the print-and-test iterations. This design ensures proper mounting and functionality within the testing rig. With the final, validated design, the CAD model was prepared for the final 3D printing fabrication, ensuring optimal orientation to accurately replicate the refined dimensions.

Parametric Deesign Considerations

The initial design measured 105 mm long, 114 mm wide, and 20mm depth and 1.5 mm gap. Subsequent work resulted in revisions, with the second redesign extending the dimensions to 150 mm in length, 115 mm in breadth, and 10 mm in thickness for testing. The final design was improved to be 52 mm long, 57 mm wide, and 10 mm height with a rail system added for adjustable height. These improvements were made to improve the design for stability.

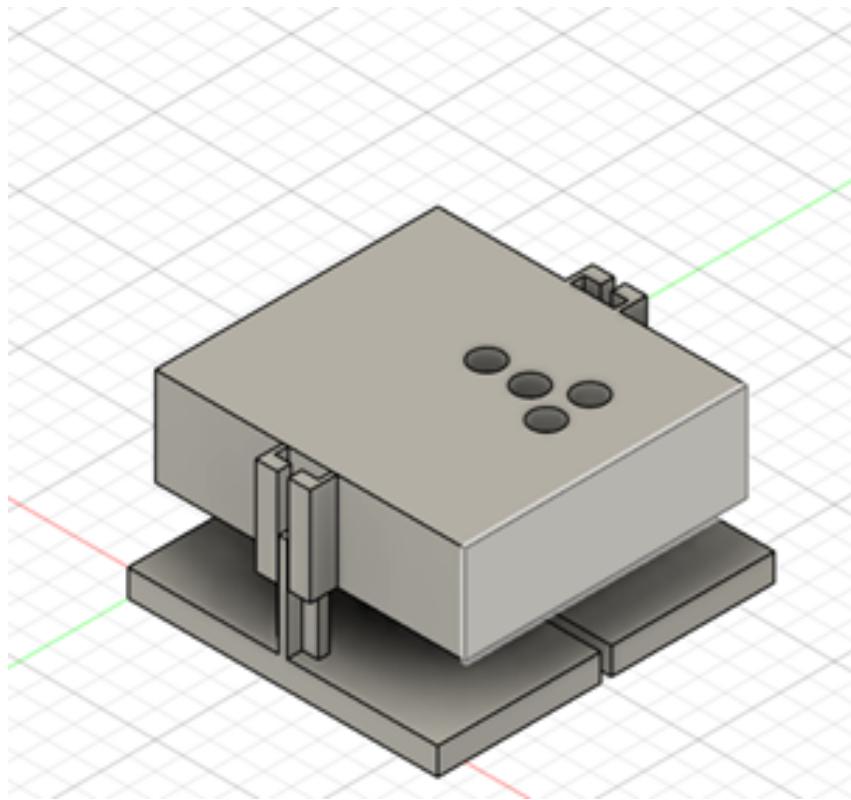


Figure 4.10.: Final Design Iteration of the Enclosure

Overview of 3D Printing methods

Fused Deposition Modelling (FDM) Fused deposition modelling is a popular Additive manufacturing (AM) Method because of its fast production, cost efficiency, and ease of access. Broad material adaptation and capability to produce complex components [24]. It works by using a material extrusion process, where a continuous filament of thermoplastic or composite material is used to construct 3D parts. The polymer filament is forced over the nozzle and fed over the build plate or previously solidified substance, and the product is built through a layer-by-layer method while maintaining a steady speed and pressure [25].

Stereolithography (SLA) Stereolithography is a 3D printing technology that uses liquid photopolymer that polymerises when exposed to a laser. The worktable moves up and down after being immersed in resin. The photopolymer hardens after being selectively irradiated with a laser beam. The printed object becomes a detailed physical model from the bottom up [26].

Selective Laser Sintering (SLS) Carl Deckard developed selective laser sintering (SLS) in 1987, and it is among the best powder-based AM methods [27]. The solid structure is created in this method by sintered powder particles using a laser source [28]. This SLS process uses two chambers: the building chamber is used for printing, and the feed



Figure 4.11.: SLA Printer

chamber loads the powder onto the bed using a roller. First, a roller in the feed chamber distributes the powder evenly onto the constructed chamber base plate. The building chamber is heated to a temperature below melting point before the carbon dioxide (CO₂) laser is shone on the powder to cure it. Next, the building chamber descends a little, and the feed chamber applies the powder over the printed layers. After construction is finished, the extra powders that served as a supporting framework in the building chamber are removed, and the extra material is recycled. This process for creating high-density prototype goods is flexible and economical [29]. However, compared to the SLS method, the operation cost is expensive, and the product quality is bad because of the high power of the laser input [30].

Justification for FDM printing

For this project, FDM was decided as the main manufacturing method for our CAD design this is due to benefits such as:

Accessibility and Affordability In addition to its low cost and ease of use, FDM's ability to produce complicated geometries reinforces its standing as the leading option for small-scale prototyping. The capacity to create elaborate designs with different layer heights

can greatly improve the precision of microscopic features, which is critical in applications like microfluidics and biomedical devices [31]. SLA may provide better surface smoothness and precision, but it is more expensive and requires more post-processing, making it less accessible for rapid prototyping [32]. While SLA and SLS are valuable alternatives, FDM's capabilities make it an indispensable tool for designers and engineers looking for fast and cost-effective solutions in their projects.

Post-processing Requirements Furthermore, the environmental impact of 3D printing processes is becoming a more essential factor for designers and engineers. While FDM uses recyclable thermoplastic materials, other techniques, such as SLA, frequently rely on resins, which may offer disposal issues due to their chemical makeup [33]. As the industry progresses towards more sustainable practices, optimising material selection not only lowers costs but also reduces environmental footprints, making FDM a more appealing alternative in this aspect. Furthermore, developments in biodegradable filaments are increasing FDM's attractiveness, allowing for prototypes that line with eco-friendly goals while maintaining performance and quality. Thus, FDM's mix of cost, adaptability, and low environmental effect places it at the forefront of the growing additive manufacturing industry.

Material Selection

Overview of Material Properties The material properties of 3D printed parts are critical for their performance in various applications, for FDM printing, there are various filament materials such as acrylonitrile butadiene styrene (ABS), Polylactic acid (PLA), Polyetheretherketone (PEEK), polypropylene (PP), and Thermoplastic Polyurethane (TPU) [24].

Cost Considerations When selecting materials for 3d printing, cost is a significant factor. PLA is one of the most cost-effective option available as its one of the more widely used filaments in FDM due to its thermal and rheological properties [24], which make it easier to manufacture parts.

3D Printer Selection and Specifications

The Creality Ender 3 SE(Figure 4.12) was selected as the Fused Deposition Modelling (FDM) printer for this project due to its efficiency and reliability. Its single-extrusion design, coupled with an automated Z-offset levelling system, significantly reduces recalibration requirements, ensuring consistent print quality and minimising downtime.

The layout of the FDM process is shown in Fig. Here the filament is stored in the spool roller and is connected directly to the extrusion head i.e. extrusion head. This head will move in X and Z-directions while the build plate will move in the Y-direction. Electric



Figure 4.12.: Creality Ender V3 SE

motors will control the position of the moveable liquefier [34]. Generally, for this process material filaments are used for both the supporting and built material of the print.

This FDM technique generally consist of three sections being the pre-processing, production and post processing. In the pre-processing stage the product is created using CAD Software and saved in STL format. Before slicing the file, key parameters such as slicing settings, build orientation and machine temperature are considered. These crucial parameters Influence the final products mechanical properties. Figure 4.14 shows the necessary process parameters. After this step, slicing is performed using the Software Ultimaker Cura and the tool path is generated as G-code which is a computer numerical control code used to manage the extrusion process.

Print Settings and Parameters

Infill Pattern The infill pattern offers internal support to the 3D print as it develops each layer. Printing layers without an infill pattern would be time-consuming and result in material slumping over empty areas [35]. The figure below displays several infill patterns, including triangular, grid, cubic, honeycomb, concentric, rectilinear, rectangular, octet, and wiggly. For this project, the cubic infill pattern was selected due to its advantages that align with our project's requirements. Firstly, the cubic pattern provides reliable mechanical properties ensuring robust structural support and strength, which is necessary

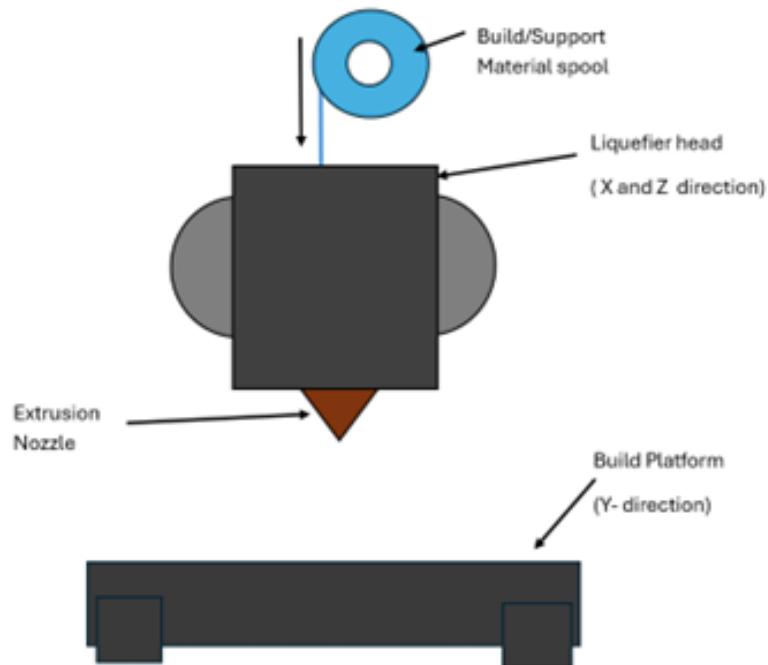


Figure 4.13.: Single Head FDM Process Layout

FDM Process

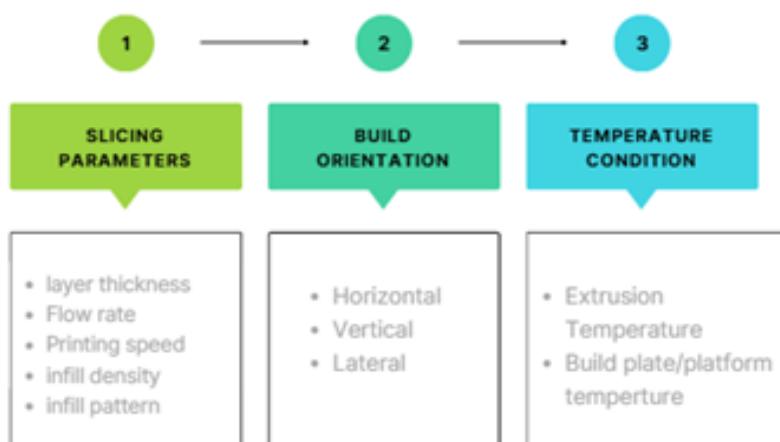


Figure 4.14.: FDM Parameters

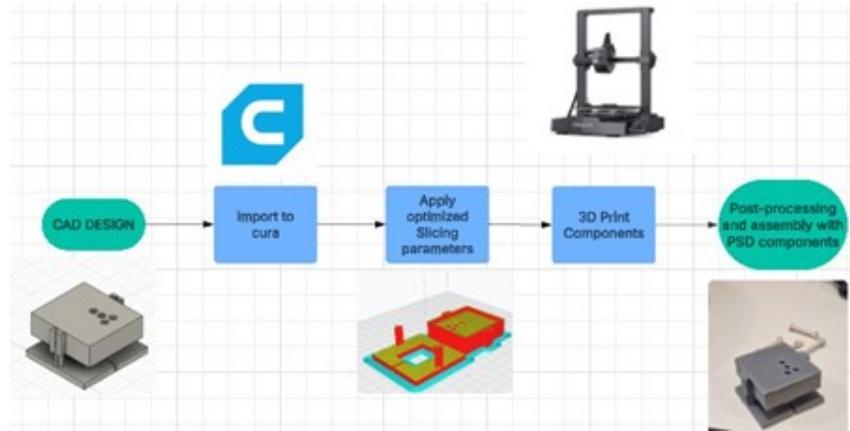


Figure 4.15.: Flowchart of the FDM Process

for the housing component of the design. Secondly, the cubic infill pattern is efficient in terms of print speed. It allows for faster printing compared to other infill patterns without compromising its structural efficiency. These benefits make the cubic infill pattern an ideal choice for achieving the desired balance between strength and efficiency in the printed parts

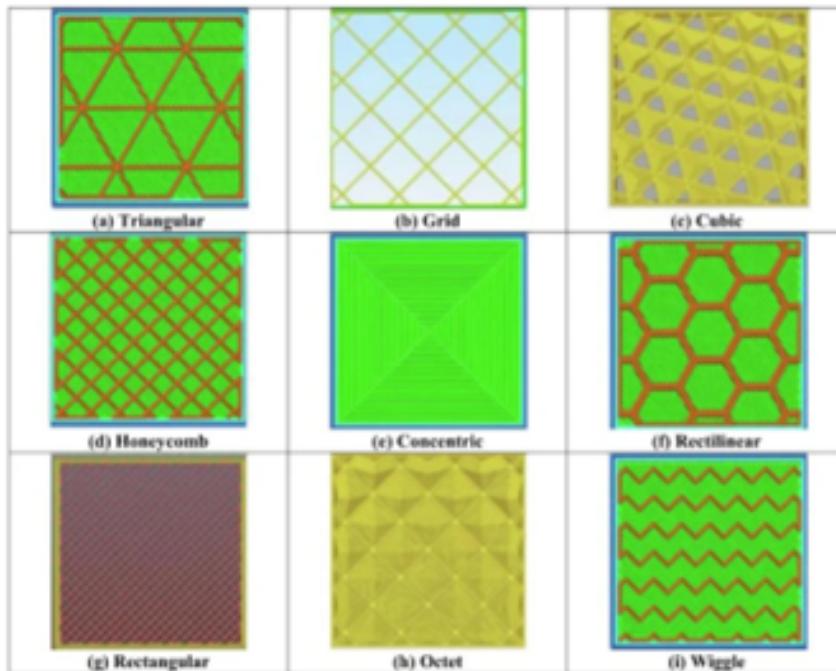


Figure 4.16.: Various Infill Patterns Used for FDM Processes

Infill Density Mechanical performance of 3D-printed components is largely determined by the infill density, which dictates the amount of material used internally [24]. Groza and Shackelford [36] categorise infill patterns into three distinct types, each impacting the mechanical properties and print efficiency of 3D-printed components. The ‘solid normal’ infill, characterised by its dense interior, yields robust mechanical performance. Conversely,

the ‘sparse’ infill pattern prioritises reduced printing time and material consumption by incorporating gaps and utilising a unidirectional raster. The ‘sparse double dense’ infill, while also focusing on minimising printing time and material usage, employs a crosshatch raster pattern. However, for the fabrication of the testing rig aperture in this project, a 15% infill density was employed, which lies within the sparse categorisation, to achieve a balance between structural integrity and material efficiency.

Printing Speed Printing speed is the nozzle’s pass-through speediness on the build platform when printing. The printing speed determines how long the product takes to produce. In addition, the printing speed has a maximum effect on the deformation of the product since, during production, this quick printing could produce significant residual stresses[24]. For this project a nozzle speed of 180 mm/s was chosen, as it showed the best performance than other nozzle speeds when doing test calibrations.

Operating Temperature Rajan et al. (2022) emphasise the vital role of operating temperatures in 3D printing, particularly nozzle temperature (extrusion temperature) and bed temperature (build platform temperature). Prior to printing, the nozzle is heated to a temperature sufficient to melt the filament, allowing extrusion. Simultaneously, the build platform is heated to a bed temperature that promotes good adhesion and reduces warping during printing [24].

Nozzle Temperature The choice of 200°C for PLA extrusion was determined through a combination of filament manufacturer guidelines and experimental optimisation. This temperature facilitated adequate melt flow, enabling proper layer bonding and minimising the risk of under-extrusion. Lower temperatures resulted in insufficient melt flow, leading to weak layer adhesion, while higher temperatures increased the potential for stringing and warping. Therefore, 200°C was identified as the optimal setting to achieve consistent and reliable print quality for the testing rig aperture.

Build Orientation To optimise the dimensional accuracy of the photodiode hole array and minimise the need for extensive support structures, the testing rig aperture was printed in a horizontal orientation. This orientation, as shown in Figure, placed the top surface, containing the critical photodiode holes, directly on the build plate. This reduced the potential for surface imperfections caused by support structures on these critical features and facilitated a more uniform thermal distribution during printing.

Iterations and Improvements

There were two main challenges that we came across during the printing process, which were bed adhesion and dimensional inaccuracy.

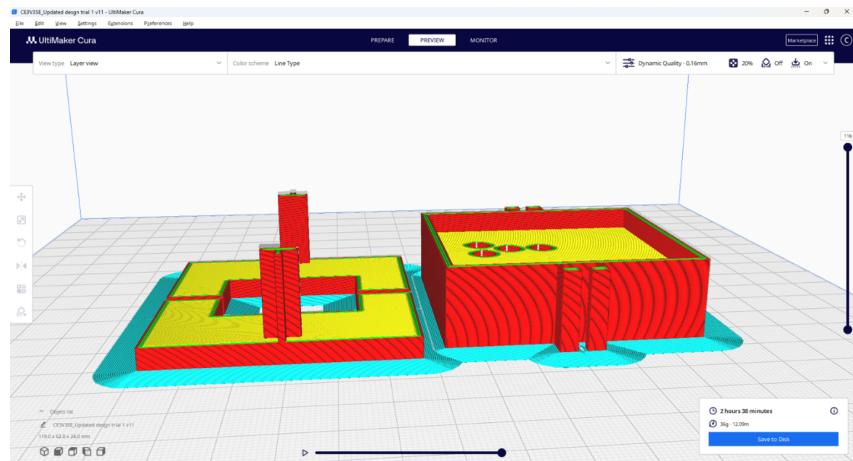


Figure 4.17.: Design Sliced for Printing

Bed Adhesion For the bed adhesion issue shown in Figure 4.18, we found that the source of the issue was temperature related. Firstly, the build plate temperature was 45°C which was lower than that what is recommended when printing with PLA which ranges from 50°C - 70°C. Additionally the extrusion temperature was 190°C which is on the lower end of the recommended range which is typically 190°C - 220°C [37] after some testing to find the ideal which was found to be at 60°C. So, after some readjustments to the build plate and extrusion temperature, a first layer was able to stick to the build plate without any warping happening shown in Figure 4.19. Although the first layer print worked, there was still warping on the corners of the prints, as well as some 'wisping' on the surface, so research was conducted to find a solution, which was to use a brim on the print shown 4.20. This is an extra layer of material printed around the base of the print object. This increased the surface area contact with the build plate, improving adhesion and stability during printing. These methods significantly reduced bed adhesion issues, leading to more successful and reliable prints as can be seen in Figure 4.21



Figure 4.18.: Bed Adhesion Failure

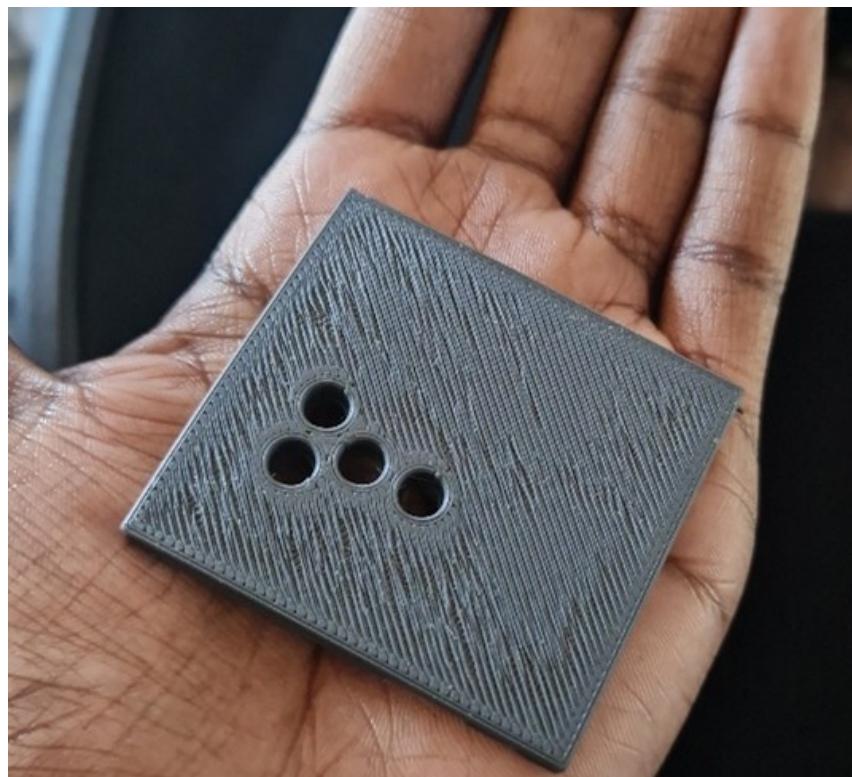


Figure 4.19.: Test Print After Temperature Adjustments

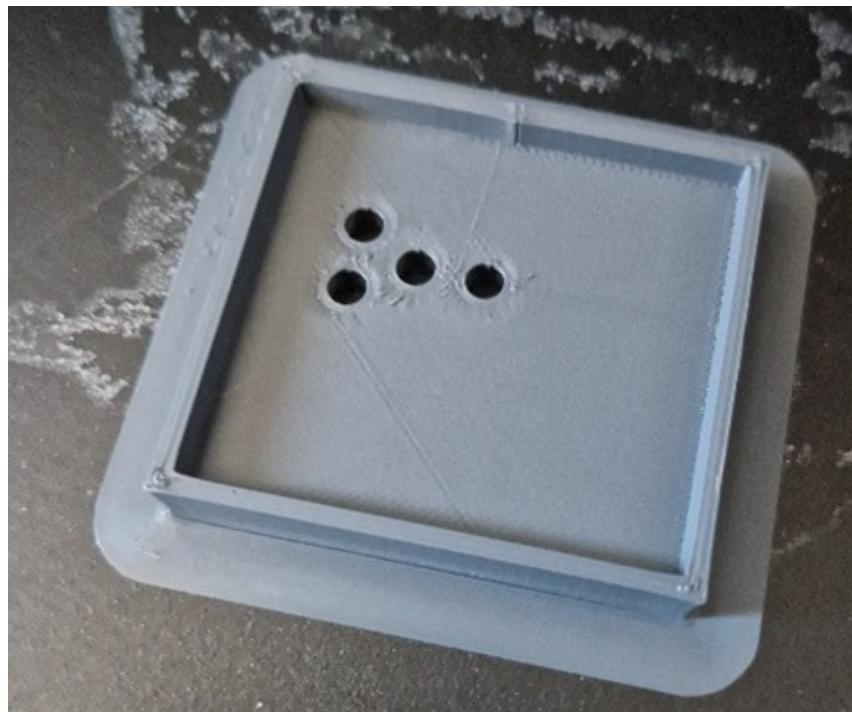


Figure 4.20.: Part Printed with Brim

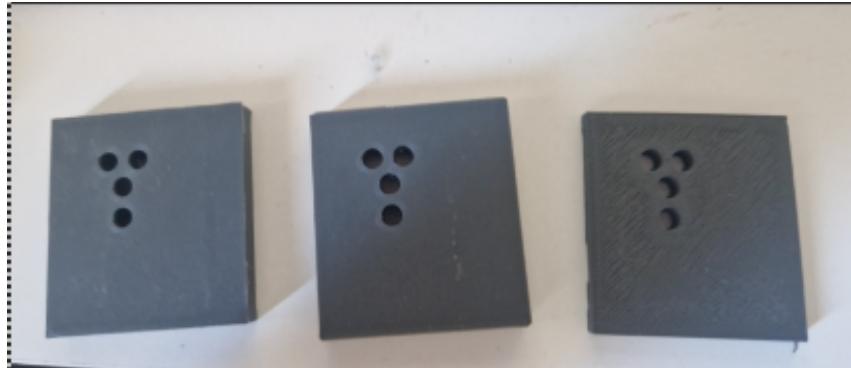


Figure 4.21.: Improved Bed Adhesion from Test Print Attempts

Dimensional Inaccuracy Another one of the issues encountered in this project was the dimensional inaccuracies specifically the hole sizes for the test rig in which the CAD model did not translate entirely to the 3D printed part. After doing some fault finding it was found that over time as the printing process continued some of the printed layers had begun to cool, thus shrinking down the holes made for the photodiode array, causing them to not fit as can be seen in Figure 4.22. To address dimensional inaccuracies between printed apertures and electronic components, an iterative calibration process was conducted using Ultimaker Cura's hole expansion feature. This technique, which involves adjusting the horizontal dimensions of the print, is essential for several reasons. Primarily, it compensates for material shrinkage, a common occurrence during FDM printing, ensuring dimensional accuracy. Furthermore, by fine-tuning the dimensions of slots and holes, it facilitates improved fit and alignment of components like photodiodes. Ultimately, this adjustment enhances the precision of the printed part, ensuring that critical components are placed accurately, thereby preserving the design's intended functionality. Through this process, an expansion of 0.16mm was determined to be optimal, ensuring precise integration of electronic components.



Figure 4.22.: Dimensional Inaccuracy of the Holes after Printing

4.2. Data Acquisition System

4.2.1. Functional Requirements

The output signal from the photodiode array amplifier is required to be converted to digital form for post-processing. This requirement is filled by designing a Digital Acquisition System (DAQ) capable of recording the signal from the four photodiode circuits simultaneously. The choice of design was conceived by analyzing the analog signal and determining some basic requirements of the Analog to Digital Converter (ADC) the DAQ must possess.

Analog Signal Characteristics

- The signal is four channel, one per photodiode, and between 0 and 5 Volts, as the TIA and post amplification was designed specifically for this output.
- Close to DC frequency, i.e., static in nature, due to light intensity remaining static under most tests. One test is performed at 0.2Hz, which is still very low frequency, with the light completing a semicircular arc once in 130 seconds (26 positions of 5 seconds each).
- Later in testing it was found that the signal is impacted by interference of 400mVpp

at a frequency fluctuating from 160kHz to 180kHz from the RED testbench power supply, as pictured in Figure 4.23.

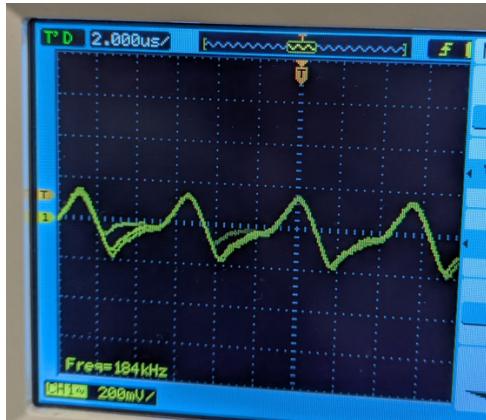


Figure 4.23.: Signal Noise Analysis, oscilloscope AC coupled

CSV Data Structure and Format Specification is as follows:

The output of the DAQ is to be saved in Comma Separated Values (CSV) file format, with columns as follows: Sample(nr.), Time(ms), A0(V), A1(V), A2(V), A3(V). This allows for easy post-processing and plotting.

4.2.2. Design Approach

The characteristics of the signal being low frequency, combined with the requirement to read all four signals simultaneously and in sync, meant two things: the Sampling Rate could be quite low due to Nyquist theorem telling us that the sampling rate must be at least twice the frequency of the signal being sampled, in order to maintain the original signal without aliasing [1, p. 146]. Therefore a low performing ADC is acceptable for a signal changing at under 1Hz. And secondly, the DAQ must support sampling from at least four analog inputs. These requirements meant that a cheap Arduino based DAQ could fit perfectly the needs of the project: it is powered by the Atmega328P which has an included ADC of 15 ksps [38, p.205]. And the Arduino Nano has four analog inputs.

Arduino Programming

The Arduino-based DAQ will require both a C++ program written for the Arduino itself, as well as a program or script on the PC receiving the digitized signal, this is because the Arduino lacks both the memory requirements and capability to store the recorded digitized signal to some internal memory.

The Arduino C++ Program must be able to listen to commands from the user on the PC receiving, start a recording, and immediately transmit to the PC over serial communication.

4.2.3. Technical Specifications

Arduino Code

The Arduino Code which uses the Arduino ADC is formed of the setup() function triggered once at the start/reset of the device and a standard continuous loop triggered after setup completes. Inside the loop, two if statements check for instructions from the PC script. The recording time limit is hardcoded as a global function. Figure 4.24 shows the algorithm as a Flowchart that checks for Serial data in, waits for a command to start recording, and if recording time has reached the preset limit, it stops recording, sends the last values to the Python script on the PC and a "recording_stopped" command. A FSM diagram is also available in Figure 4.25. The pseudocode used while designing the Arduino side of the DAQ system, is available in Listing 4.1. The final code is available in Appendix A.1.1.

The Atmega328P does not have a separate ADC clock input, therefore the CPU clock is used by first dividing by a default rate of 128, this divider is changed to 16 by changing bits 2-0 to 100, as per [38, p.219]. This increases the clock speed available to the ADC for a higher sampling rate. This results in a 1MHz clock signal to the ADC (16MHz/16) which seemed needed when dealing with multiplexing four analog inputs to a single ADC. The process is as follows:

Original ADC Clock Speed (with default prescaler of 128):

$$f_{\text{ADC-default}} = \frac{f_{\text{CPU}}}{\text{Prescaler}_{\text{default}}} = \frac{16 \text{ MHz}}{128} = 125 \text{ kHz} \quad (13)$$

Optimized ADC Clock Speed (with modified prescaler of 16):

$$f_{\text{ADC-optimized}} = \frac{f_{\text{CPU}}}{\text{Prescaler}_{\text{optimized}}} = \frac{16 \text{ MHz}}{16} = 1 \text{ MHz} \quad (14)$$

Conversion Time Calculations: ADC requires approximately 13 clock cycles for each conversion [38, p.208] Optimized ADC Clock Speed (with modified prescaler of 16):

$$T_{\text{conversion-default}} = 13 \times \frac{1}{f_{\text{ADC-default}}} = 13 \times \frac{1}{125 \text{ kHz}} \approx 104 \mu\text{s} \quad (15)$$

$$T_{\text{conversion-optimized}} = 13 \times \frac{1}{f_{\text{ADC-optimized}}} = 13 \times \frac{1}{1 \text{ MHz}} \approx 13 \mu\text{s} \quad (16)$$

Time required to sample all 4 analog inputs:

$$T_{\text{4channels-default}} = 4 \times T_{\text{conversion-default}} = 4 \times 104 \mu\text{s} \approx 416 \mu\text{s} \quad (17)$$

$$T_{\text{4channels-optimized}} = 4 \times T_{\text{conversion-optimized}} = 4 \times 13 \mu\text{s} \approx 52 \mu\text{s} \quad (18)$$

Maximum theoretical sampling frequency for all 4 channels:

$$f_{\text{sampling-max-default}} = \frac{1}{T_{\text{4channels-default}}} = \frac{1}{416 \mu\text{s}} \approx 2.4 \text{ kHz} \quad (19)$$

$$f_{\text{sampling-max-optimized}} = \frac{1}{T_{\text{4channels-optimized}}} = \frac{1}{52 \mu\text{s}} \approx 19.2 \text{ kHz} \quad (20)$$

Actual limited sampling frequency (based on minSampleInterval = 2ms):

$$f_{\text{sampling-actual}} = \frac{1}{2 \text{ ms}} = 500 \text{ Hz per channel} \quad (21)$$

Effective data rate across all channels:

$$\text{Data Rate} = 4 \text{ channels} \times 500 \text{ Hz} = 2000 \text{ samples/second} \quad (22)$$

In real testing the actual sampling rate was closer to 330Hz for 5 second recordings or 100Hz for a 2 minute recording - after some investigation the only explanation was the relativly small size of the transmission buffer implemented by the Serial C++ library. The buffer is of only 64 bytes, and when it fills, the function Serial.write() (used by println()) will block the write untill there is space in the buffer[ref:arduino.cc/serial.write]. As our line of text is quite long "498,5000,0.059,0.054,0.073" for example has 26 characters

(last line of a 5 second recording). For larger recording length, where the first and second column, Sample and Time, can get quite large, the sampling rate decreased considerably, but was kept constant (around 10ms for a 2 minute recording). Presumably due to optimization in the Arduino Serial Hardware/Software or compiler, it remains constant at 10ms. However this was not investigated further as for our near-DC signal, even 10ms was a fast enough sampling rate for our DC-like signal.

```
1 recordingDuration = 5000 // for how long to record in milliseconds
2
3 minSampleInterval = 2      // control how fast to sample to avoid
                           // relying on Arduino performance
4
5 // Initialize serial communication
6 // Initialize analog inputs
7 // Setup ADC
8
9 // Infrom PC listening on Serial Connection: Arduino is ready to
   record
10 Serial.print("Arduino_DAQ_Ready")
11 // enter the loop
12 void loop(){
13     //listen for command from PC script:
14     String command = Serial.read()
15     //set system state
16     if (command == "START"){
17         // Send header of csv
18         Serial.println("Sample,Time(ms),A0(V),A1(V),A2(V),A3(V)")
19         // keep track of system state
20         state = recording
21         // keep time
22         startTime = currentTime()
23         // send confirmation
24         Serial.println("recording in progress")
25     }
26     // check if recording
27     if (state == recording){
28         // check if within recording period
29         currentTime = currentTime()
30         elapsedTime = currentTime - startTime
31         if(currentTime <= recordingDuration){
32             // Also check not recording too fast
33             if(currentTime - lastSampleTime >= minSampleInterval){
34                 sampleCount++;
35                 // Start each row with sample count and time of sample
36                 String currentCSVrow = String(sampleCount) , string(
37                     elapsedTime)
38                 // Multiplex through all analog inputs
39                 for (int = 0;i<4;i++){
40                     // Read analog input i
41                     analogValue = analogRead(i);
42                     // Convert analog value to voltage
43                     voltage = map(analogValue, 0, 1023, 0, 5);
44                     // Print the data to the serial port
45                     Serial.print(voltage);
46                     if (i < 3) {
47                         Serial.print(",");
48                     }
49                 }
50             }
51         }
52     }
53 }
```

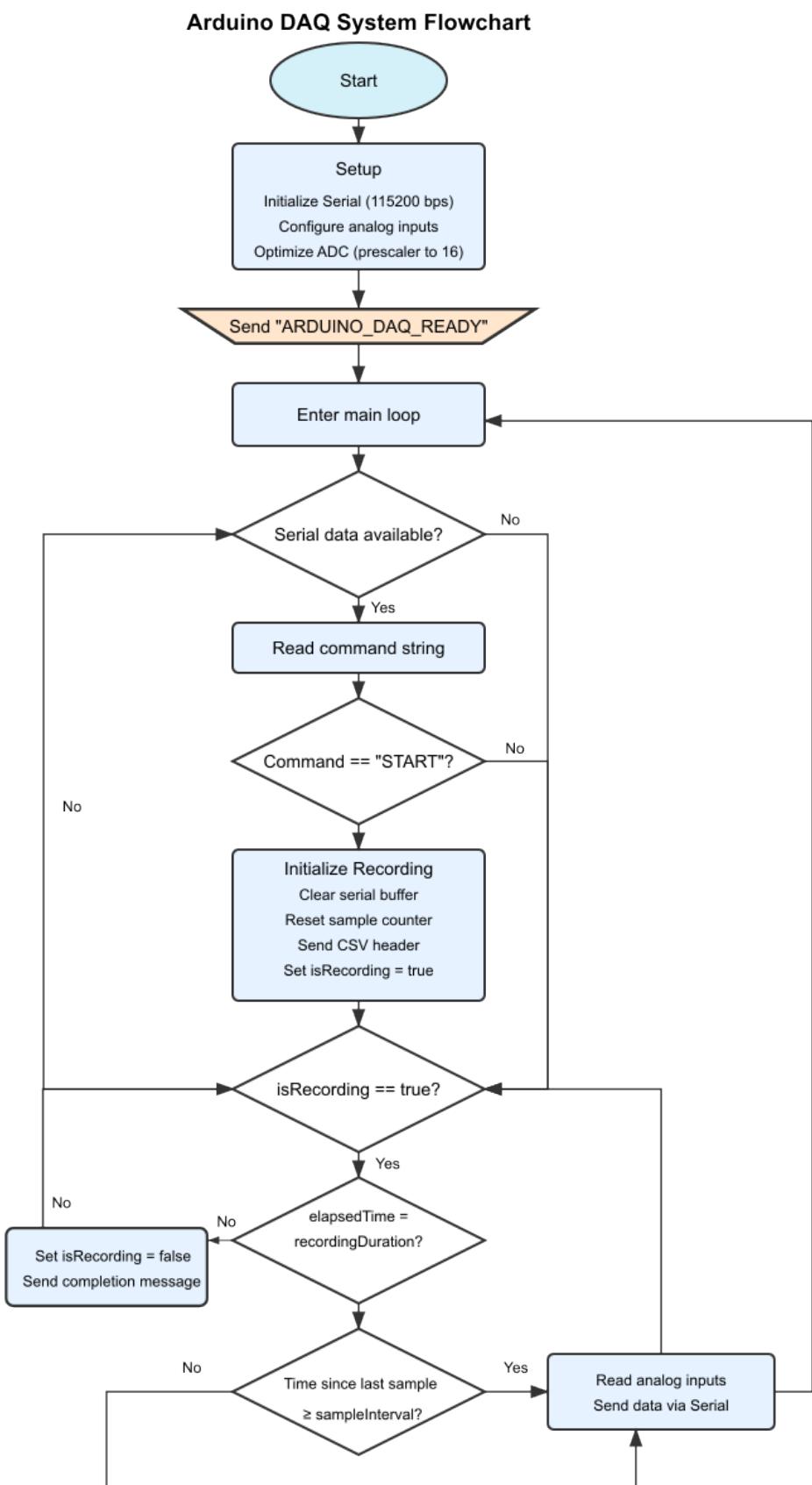


Figure 4.24.: Flowchart Arduino DAQ C++ program

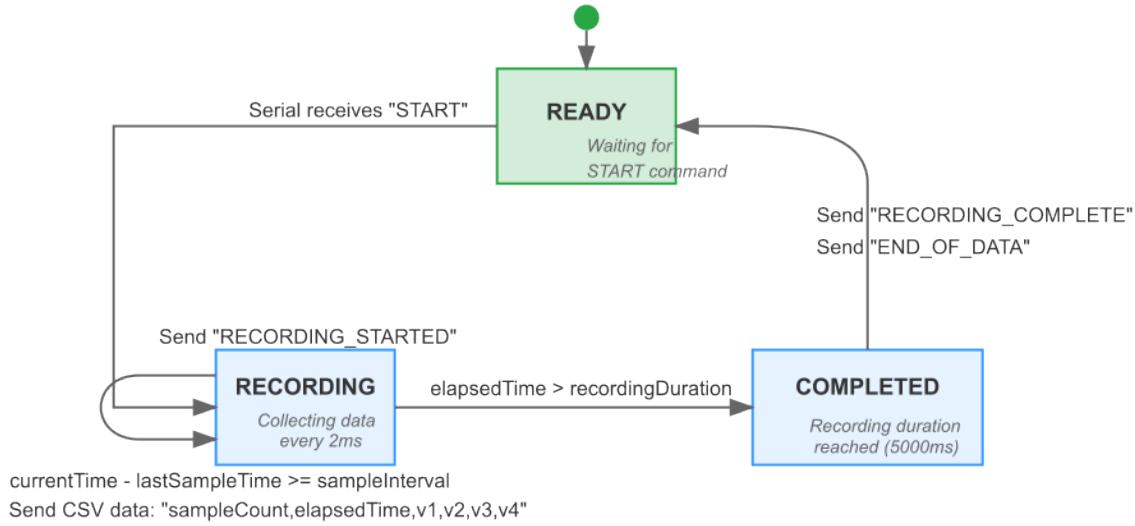


Figure 4.25.: FSM Arduino C++ LOOP

```

39         //read raw values
40         rawValue = analogRead(analogInputs[i]);
41         // compute real value
42         voltage = rawValue * 5/1023
43         // add value to current row to send
44         currentCSVrow += String(voltage)
45     }
46     // Send completed row to PC
47     Serial.println(currentCSVrow)
48 }
49 }
50 }
51 // end recording
52 else{
53     state = notRecording
54     // tell PC recording finished
55     Serial.println("Recording_finished")
56 }
57 }

```

Listing 4.1: Arduino DAQ PseudoCode

Sampling Rate Details Several factors restrict the sampling rate:

Sample Interval Setting The most direct limitation is the `sampleInterval` constant set to 2ms in the code. It was meant to avoid having random sampling rates based on the number of computations required. This means samples are taken no more frequently than every 2 milliseconds (500 Hz theoretical maximum) of all four channels. The "jump" to sample the next channel is not limited in the code, but it will take 13 clock cycles, (ie. around 13 μ s at 1MHz ADC clock) to switch to the next channel.

ADC Prescaler Configuration The ADC prescaler is set to 16 (from the default of 128) with this line:

```
ADCSRA = (ADCSRA & 0xF8) | 0x04;
```

This increases the ADC clock to $16\text{MHz}/16 = 1\text{MHz}$. With each conversion taking 13 ADC clock cycles, the theoretical maximum sampling rate is about 76.9kHz for a single channel.

Serial Transmission Overhead Each sample requires formatting and sending data over serial:

```
String dataString = String(sampleCount) + "," + String(elapsedTime);
// ... format and add voltage values ...
Serial.println(dataString);
```

This string creation and serial transmission takes some time to process as mentioned earlier.

Serial Baud Rate The code uses 115200 bps, which limits how quickly data can be transmitted. Each sample in this format might be around 30-40 bytes, which means ~3000-3800 samples/second theoretical maximum throughput.

String Operations The use of the Arduino `String` class is memory-intensive and can cause fragmentation over time, potentially causing the slowdowns noticed during testing with larger timeframes (and longer strings).

Python Script

The digitized signal must be interpreted and saved on the PC. This is done via a python script which listens to the Serial port from the arduino. The signal then also required cleaning from RF interference discovered during testing. In Figure 4.26 a flowchart is produced showing the way the script works: after initial setup that sets up the Serial Communication, the script waits for a "DAQ_READY" signal from the Arduino. Once this is received, a csv file is created and the script sends a "START" signal which the

Arduino interprets and starts sampling and sending the data. The script receives each line and saves it in the new CSV, and continues to record data until the Arduino sends a "RECORDING_COMPLETE" signal - which will happen when the recordingDuration is reached. At this point the python script performs the following post-processing steps:

The filter_and_save_data() function' purpose is mainly to correctly interpret the CSV. It takes a csv with the raw voltage values, saves them into a Pandas DataFrame for easier manipulation and sends each channel to the apply_lowpass_filter() function which will be described below. filter_and_save_data() pseudocode is produced in Listing 4.2.

```
1  FUNCTION filter_and_save_data(filename)
2      // Load data from CSV file into a table structure
3      data_table = READ_CSV(filename)
4
5      // Streamline the data by converting text to numbers
6      FOR EACH column IN data_table
7          CONVERT column values to numeric type
8          IF conversion fails for any value
9              REPLACE with NaN (Not a Number)
10         END FOR
11
12     // Remove any rows containing NaN values
13     REMOVE all rows with NaN values from data_table
14
15     // Calculate sampling frequency
16     time_differences = CALCULATE differences between consecutive time
17     values
18     typical_time_difference = FIND median of time_differences
19     sampling_frequency = 1000.0 / typical_time_difference // Convert ms
20     to Hz
21
22     // Process each data channel
23     channel_list = ["A0(V)", "A1(V)", "A2(V)", "A3(V)"]
24
25     FOR EACH channel_name IN channel_list
26         IF channel_name EXISTS in data_table
27             filtered_values = APPLY_LOWPASS_FILTER(original_values,
28             sampling_frequency)
29             ADD new column named channel_name + "_filtered" with
30             filtered_values
31         END IF
32     END FOR
33
34     // Save results to new file
35     new_filename = REMOVE_EXTENSION(filename) + "_filtered.csv"
36     WRITE data_table TO new_filename
```

Python DAQ Script Flowchart

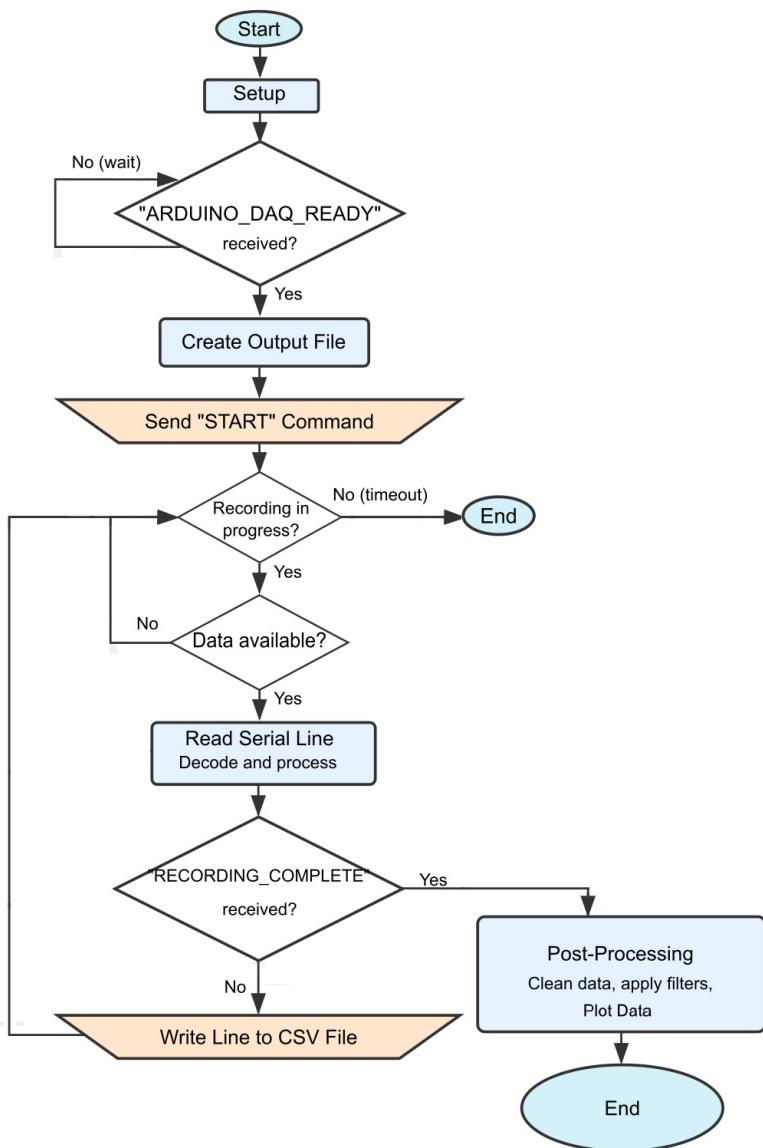


Figure 4.26.: Python Script Flowchart

```

33
34     RETURN new_filename
35 END FUNCTION

```

Listing 4.2: Python filter_and_save_data() PseudoCode

The **apply_lowpass_filter()** function was created and integrated into the script once it was observed that the RED testbench used was introducing noise as seen in Figure 4.23. The benefit of using an already built testbench that could place the light at exact positions repeatedly, meant it would make sense to accept the noise and just filter the data, as the signal of interest was very low frequency while the noise was around 170kHz. The filter could be relatively simple, due to the large frequency difference between noise and signal. A 4th order Butterworth IIR filter was deemed acceptable and a low cut-off frequency of 1Hz or 2Hz was used for the static readings. This was found to be acceptable for the test involving light location at a frequency of 0.2Hz the transition of the light from one position to another was still visibly sharp. The post-processing is not the only reason light transition would not appear instantaneous on the Voltage graph, another reason is the amplification circuit which has a 1Hz cutoff frequency via the feedback capacitor on the secondary-amplification OpAmp circuit. The pseudocode of the function that creates the low-pass digital filter and filters the data is reproduced in Listing 4.3. The butter() function from the library signal is used, from the scipy package [39] which is a free and open source library offered to the scientific community. Before feeding the cutoff frequency to the filter, it is normalized to the Nyquist rate, which means it is between 0 (DC) and 1 (Nyquist frequency), and therefore the filter can be applied no matter the sampling rate. As Schafer and Oppenheim explain in "Discrete-Time Signal Processing" Third Edition: "The frequency scaling or normalization in the transformation from $X_s(j\Omega)$ to $X(e^{j\omega})$ is directly a result of the time normalization in the transformation from $x_s(t)$ to $x[n]$ " [1, p.171]. When creating a digital filter, this normalization becomes crucial because in the continuous-time domain, frequencies are measured in Herz and in the discrete-time domain, frequencies become relative to the sampling rate. The relationship between these two frequency domains is given by $\omega = \Omega T$, where:

- ω is the normalized digital frequency (radians/sample)
- Ω is the analog frequency (radians/second)
- T is the sampling period (seconds)

However, when implementing IIR filters like the Butterworth filter used for our purpose, the bilinear transformation is employed to convert from the continuous-time domain to the discrete-time domain. This transformation introduces frequency warping, where the relationship between the analog and digital frequencies becomes:

$$\omega = 2 \arctan(\Omega T_d / 2)$$

where T_d is the sampling period. This nonlinear relationship compresses the infinite analog frequency range $(-\infty, \infty)$ into the finite digital frequency range $(-\pi, \pi)$. The warping effect is more pronounced at higher frequencies, meaning that in our case it would not be noticeable [1, p.529-530].

A critical property of the bilinear transformation is stability preservation. In the analog domain, a stable system has all poles in the left half of the s-plane reproduced in Figure 4.27. The bilinear transformation maps the entire left half of the s-plane to the interior of the unit circle in the z-plane. This ensures that the 4-pole Butterworth filter, which is stable in the continuous-time domain, remains stable when converted to its discrete-time equivalent.

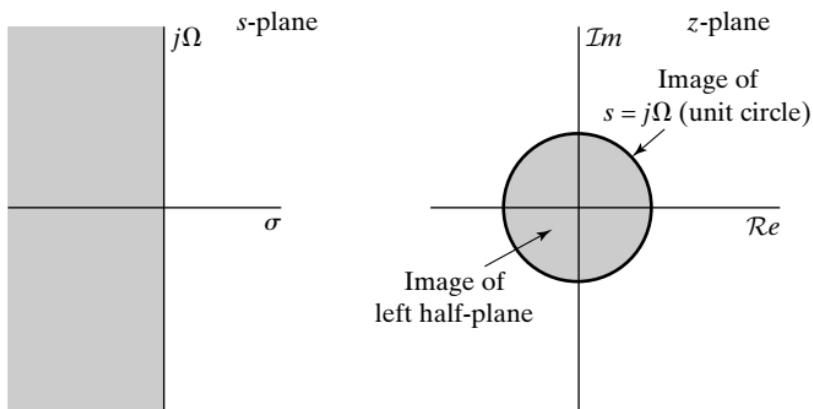


Figure 4.27.: Mapping of the s -plane onto the z -plane using the bilinear transformation [1, p.130]

The filter Frequency Response was reproduced in Figure 4.28 using the `scipy.signal.freqz()` method. The actual implementation uses `filtfilt()` which applies the filter twice, effectively doubling the filter order [40]. This affects the transition steepness and phase response. The Discrete-Time Transfer Function is reproduced in Equation 23.

$$H(e^{j\omega}) = \frac{b[0] + b[1]e^{-j\omega} + b[2]e^{-j2\omega} + b[3]e^{-j3\omega} + b[4]e^{-j4\omega}}{a[0] + a[1]e^{-j\omega} + a[2]e^{-j2\omega} + a[3]e^{-j3\omega} + a[4]e^{-j4\omega}} \quad (23)$$

```

1  from scipy import signal
2  FUNCTION apply_lowpass_filter(data, sampling_rate)
3      // set hardcoded filter values
4      cutoff_freq = 2;
5      filter_order = 4;
6      // calculate Nyquist Frequency and Normalized cut_off

```

```

7     nyquist = 0.5 * sampling_rate;
8     norm_cutoff = cutoff_freq / nyquist;
9     // generate numerator b and denominator a polinomials
10    b, a = signal.butter(filter_order, norm_cutoff, filterType = LOW);
11    // filter the data
12    filtered_data = filter(b,a,data);
13
14    return filtered_data;

```

Listing 4.3: Python apply_lowpass_filter() PseudoCode

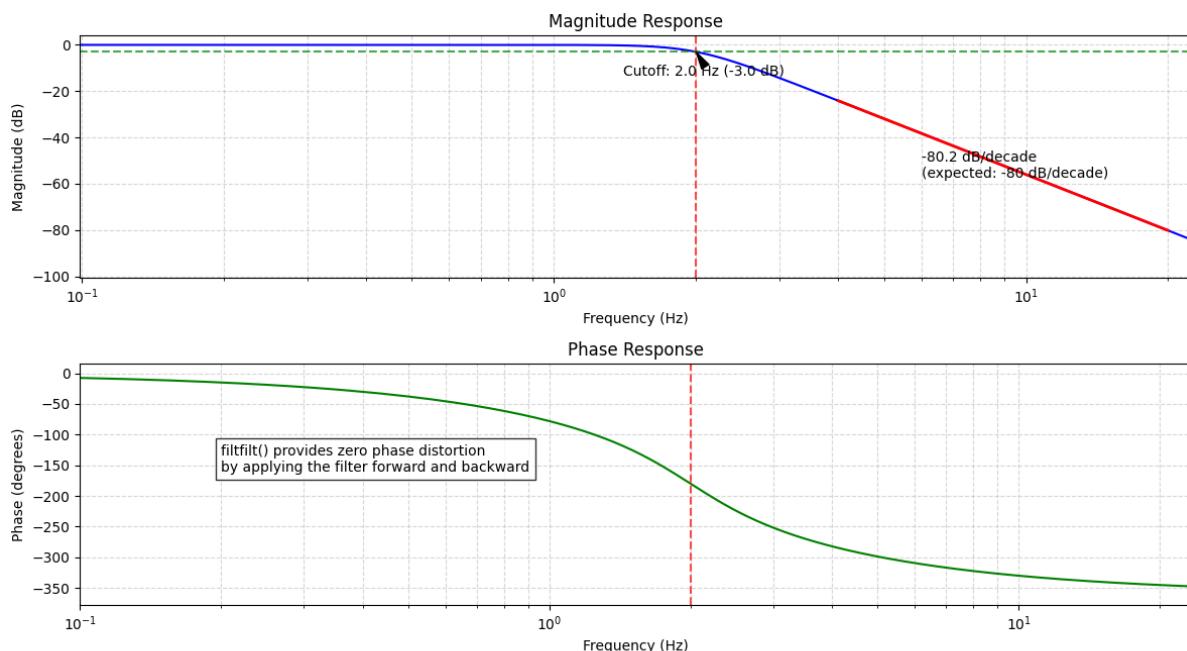


Figure 4.28.: Frequency Response of 4-pole Butterworth Low-Pass Filter (Cutoff: 2.0 Hz, Order: 4, Fs: 500 Hz)

4.2.4. Testing Strategy

The testing was done incrementally with each new version of the program, many tests were carried out as the development was dependent on both C++ code on the Arduino and the Python script on the PC to somewhat work together. At first the C++ program was tested by listening to the output in the Serial Monitor of the Arduino IDE and sending commands the same way. Debug Serial.println() lines were used to ensure the loop on the Arduino was in the correct state. Once the C++ program seemed to somewhat work as intended, the Python script was implemented to send / receive.

4.2.5. Evaluation of design

The design was tested iterartivly while making changes to the code with a signal generator at first to simulate a stead known input signal and compare readings on the Arduino IDE Serial Monitor. Once the Python script was fully working, tests were performed again for the python script's ability to read the values sent correctly. An issue was identified where sometimes the Arduino would send the column headers not at the beggining of the CSV and these lines had to be cleaned manually. This could be due to Serial buffer issues but was not investigated. Later an attempt was made to perform the cleaning programatically by creating a function in the script to do so.

4.3. Renewable Energy Demonstrator Testbench

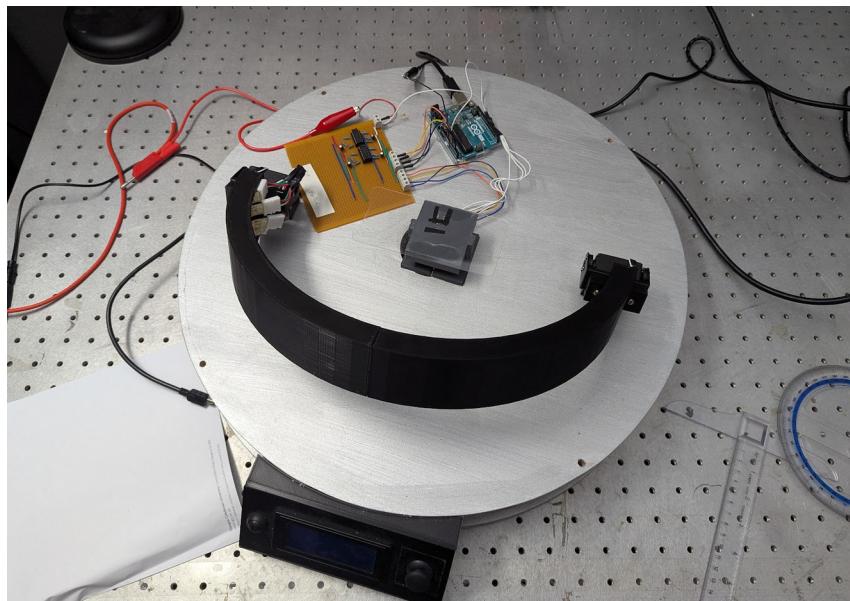


Figure 4.29.: The RED Testbench modified to fit our sensor

For testing the capability of the Sun Sensor to correctly detect the location of the light source, a test bench was required that could reliably place the location of the light at a precise location repeatedly. For this purpose we used a project built by our colleagues in the European Project Semester (EPS) year 2021/22 who createad just such a device intended for demonstrating renewable energy creation live and interacte [21]. Their device was able to demonstrate the energy levels created by a Photovoltaic (PV) cell by light emited at different angles. The light emission would change location based on time of day and the PV cell readings would show the difference in energy. Futher the PV cell was controllable by a joystick to point the PV Cell at the optimum angle for the highest energy capture. For our project, the arch and LED strip were used for outputing light from different angles. The photovoltaic cell was removed together with the servomotor

to make space for our Position Sensitive Detector (PSD) device. The RED device was composed of two separate Arduino microcontrollers:

- Arduino Uno - for controlling the Arch
- Arduino Nano - for controlling the LED Strip

The **Arduino Uno** is the main controller which not only controlled the Arch assembly, but also acted as the primary controller, receiving inputs from a button and joystick, updating an LCD screen and sending an impulse to the secondary Arduino Nano for controlling the LED Strip. A photo of the RED Testbench is available in Figure 4.29.

The **Arduino Nano** is the secondary controller that receives a digital signal from the

4.3.1. LED Controller reprogramming

Once the RED was modified by removing the servo in the middle, both the LED controller and the main controller had to be reprogrammed. This was an easy task as we had access to the original code and permission from the RED [21] team to modify it. For the LED controller, two versions were created:

One version that more closely resembles the original code, listens to the signal on a digital pin from the main arduino, and change to the next position out of a total of five positions.

A second version of the LED controller was created that simply changed which group of 3×3 LEDs were turned on every 5 seconds. This was required to perform a dynamic test where the datastream would be recorded to compare with simulation. The full code is available in Appendix F. This version was a "hack" that allowed us to record a moving light with the arch set to stay at $90^\circ()$ (one of the original 5 positions).

The flowchart shows the basic operation is reproduced in Figure 4.30. The final design has the following angles:

Table 4.5.: Arch and RGB LED Coordinates in Hemispherical System

Preset Position	Altitude (degrees)	Approximate Azimuth (degrees)	RGB LED Position
1	20	20	3
2	50	50	9
3	90	90	13
4	130	130	17
5	160	160	23

The angles chosen to record data were randomly selected, and for simplicity and ease of measurement (manually with protractor) the altitude and azimuth were identical. These were then reproduced in simulation as per Section 4.4. The simplicity was also intended to avoid confusion between team members working on separate parts of the project.

4.3.2. Arch Movement Reprogramming

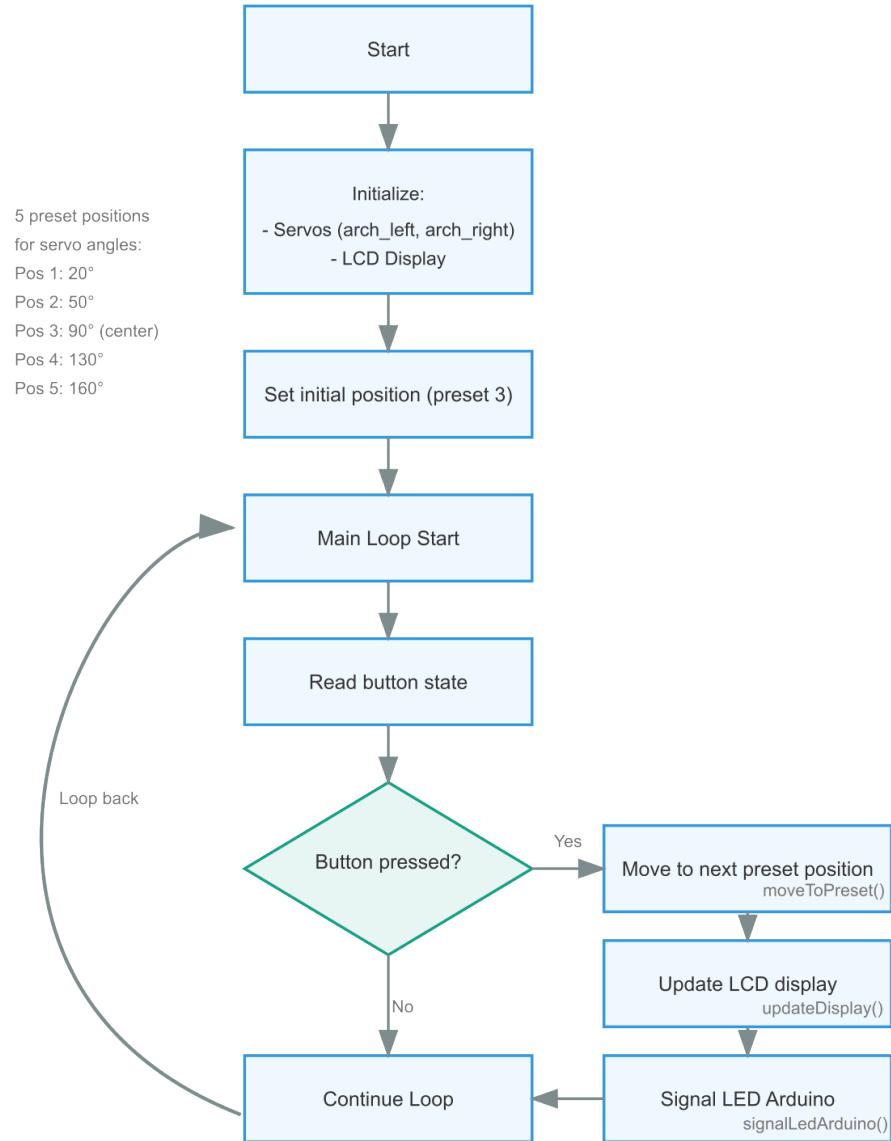


Figure 4.30.: Flowchart of the C++ program on the Arduino Controlling the Arch and other peripherals

The main Arduino microcontroller code that controls most features including the LCD display, buttons, arch servomotors had to be modified as well to fit our testbench purposes. The original code [21] was quite useful as it initialized all components and provided a guide to quickly and easily write a new program that allows for much simpler behaviour, including removal of the PV panel servomotor which had been taken out to provide space for our device. The code is available in Appendix F and a Flowchart is in Figure 4.30. The functions `signalLedArduino()` and `updateDisplay()` had been reused, as well as the

`setup()` - but with many unneeded functions removed such as extra servomotor and PV panel readings. The `loop()` function - which runs automatically in a loop after `setup()` at arduino start, was also completely modified from a much more complex logic to a simple logic that upon button press does the following:

- increase current index by 1
- call the `change_state()` function
- update LCD display with new angles
- signal RGB arduino to change state as well

The index refers to one of the 5 positions as mentioned in Table 4.5.

And the `change_state()` function does the following when called:

- Reset Arch position to zero
- Wait for Arch to reach zero
- Write the angles of the new state to the motors

The Arduino loops in this state and upon button press updates the location of the light. This resulted in a very simple testbench to use allowing us to concentrate on the light sensor device without too much worry about the testbench. Without this testbench available to us, it would have been much harder to get consistent readings of light sources, therefore gratitude is expressed to the RED team.

A look at High Frequency Noise in AC-DC Power Supply

Interference structured at around 170kHz with 400mV peak-to-peak was detected on the signal being received while the RED testbench was on as shown in Figure 4.23. This noise could be generated by several factors in the AC power supply used by the RED testbench:

1. **Switching frequency harmonics** — If it's a switch-mode power supply (SMPS), the fundamental switching frequency or its harmonics might be causing the noise. Many SMPS operate in the 50–200,kHz range.
2. **Poor filtering** — Inadequate output filtering (insufficient capacitance or poor quality capacitors) can allow switching noise to appear on the output.
3. **Improper design of magnetics** — Issues with the transformer or inductor design could cause ringing or oscillations.
4. **Resonance in the circuit** — Parasitic capacitance and inductance forming a resonant circuit at around 170,kHz.

5. Ground loops or poor PCB layout — Improper grounding or PCB layout can create noise paths. [41]

To avoid spending time diagnosing and trying to repair the testbench, an easier solution was reached: performing digital filtering of the acquired signal in post processing. Due to the signal of interest being close to DC - frequencies lower than 1Hz, and the noise being high frequency, around 175kHz, a simple digital Butterworth filter with a cutoff frequency at around 1-2 Hertz was found to be a good solution.

The only remaining issue was that this noise would sometimes trigger the internal components of the testbench, unintentionally triggering the button press from the control interface that was changing the light position, but it happened so rare that it was not a major concern.

4.4. Software Model

4.4.1. Functional Requirements

A Python model was constructed to provide a simulation of the movement and intersection of rays from a movable source to evaluate sensor performance and compare these results with practical experiments. The model allows for a number of configurable parameters:

- Trajectory of the light source in 3D space, which moves in configurable discrete increments.
- Placement, dimensions, and quantity of any number of sensors and apertures.
- Output visualisation, as a static, or animated graphic.

Affording flexibility for the model to simulate any sensor topology under a variety of conditions.

The use of a software model provides several advantages. It enables rapid iteration of design parameters — such as sensor layout, aperture size, or source trajectory — without the need for physical experimentation.

Additionally, it supports direct comparison with experimental data, offering a tool for both predictive analysis and post-experiment interpretation. This is particularly valuable in systems where physical prototyping is expensive or time-consuming.

4.4.2. Theory and Concept

In order to encapsulate the real life behaviours required to simulate this system, the software model is designed to around a structure based on the classes “Planes”, “Areas”, and “Lines”. These classes contain methods related to each of their properties and stores data on their states.

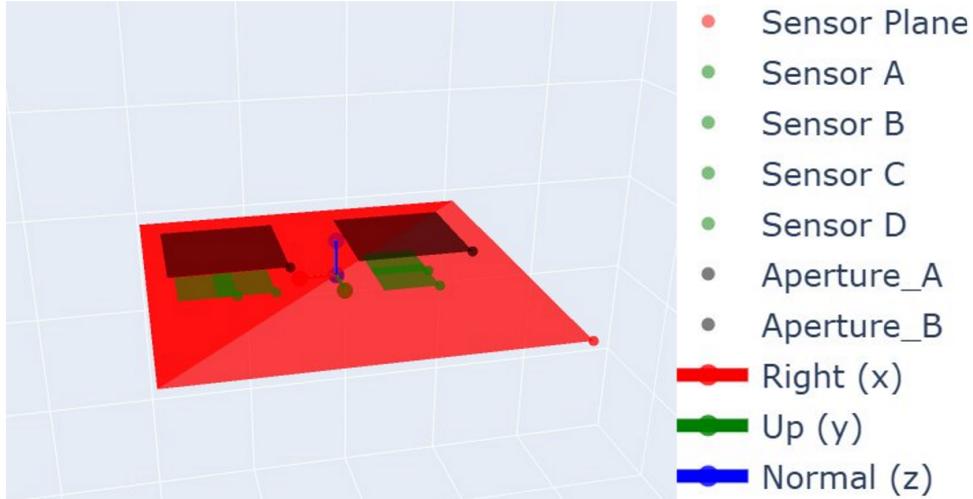


Figure 4.31.: Sensor Topology

The basis for the 3D geometry involves defining 3D vectors:

Table 4.6.: Geometric Vector Definitions

Component	Vectors	Description
Line (Ray)	$\vec{A} = (a, b, c)$	Position vector defining a point on the line
	$\vec{u} = (\alpha, \beta, \gamma)$	Direction vector defining the orientation of the line
Plane	$\vec{P} = (l, m, n)$	Position vector defining a point on the plane
	$\vec{n} = (\lambda, \mu, \nu)$	Normal vector perpendicular to the plane surface

Coordinate System

Each ray is initially defined in the local coordinate system of the source plane. This local frame is established by three orthonormal basis vectors:

- \vec{r} — right vector (local x -axis)
- \vec{u} — up vector (local y -axis)
- \vec{n} — normal vector (local z -axis)

These vectors form the columns of a rotation matrix R which transforms local coordinates to global:

$$R = [\vec{r} \ \vec{u} \ \vec{n}] = \begin{bmatrix} r_x & u_x & n_x \\ r_y & u_y & n_y \\ r_z & u_z & n_z \end{bmatrix} \quad (24)$$

A point $\vec{p}_{\text{local}} = \begin{bmatrix} x_l & y_l & z_l \end{bmatrix}^\top$ defined in local coordinates is converted to global coordinates via:

$$\vec{p}_{\text{global}} = \vec{p}_{\text{plane}} + R \cdot \vec{p}_{\text{local}} \quad (25)$$

Where:

- \vec{p}_{plane} is the global position of the origin of the plane
- R rotates the local point into the global frame

This ensures that all rays originating from the source plane move coherently when the plane is rotated or translated.

Ray generation

Rays are randomly generated within the bounds of the source plane, whose position, size, and direction are defined in the configuration. The position of each ray is calculated using:

$$X_i \sim \mathcal{U}\left(-\frac{w}{2}, \frac{w}{2}\right), \quad Y_i \sim \mathcal{U}\left(-\frac{l}{2}, \frac{l}{2}\right), \quad \text{for } i = 1, \dots, N \quad (26)$$

Where:

- w is the source plane width
- l is the source plane length
- N is the number of positions (Number of Lines)

Line-Plane Intersection

Ray projection, from a source plane to a sensor plane, is modelled using the parametric equation of a 3D line (27). This allows each ray to be described in terms of a parameter t , which enables the calculation of the intersection points between the light rays and the sensor plane.

$$\frac{x - a}{\alpha} = \frac{y - b}{\beta} = \frac{z - c}{\gamma} (= t) \quad (27)$$

Where the intersection coordinates (x, y, z) occur within a target area, a hit occurs, representing illumination.

For any given combination of source plane, and sensor plane, the t parameter is calculated using the Line-Plane Intersection equation.

$$t = \frac{\vec{n} \cdot \vec{P} - \vec{n} \cdot \vec{A}}{\vec{n} \cdot \vec{u}} \quad (28)$$

Arc movement

The simulation converts between spherical (polar) and Cartesian coordinate systems to define the movement of the source plane along an arc trajectory. This transformation enables position and orientation of the plane in 3D.

The spherical coordinates are expressed as (r, θ, φ) and are converted to Cartesian as:

$$x = r \cdot \sin(\varphi) \cdot \cos(\theta) \quad (29)$$

$$y = r \cdot \sin(\varphi) \cdot \sin(\theta) \quad (30)$$

$$z = r \cdot \cos(\varphi) \quad (31)$$

This enables generation of plane positions around a defined arc, supporting vertical, horizontal, and rigid trajectory configurations.

Movement types

- Azimuthal Scanning / Vertical Circles: The azimuthal angle θ remains fixed, while the elevation angle φ is varied.
- Polar Scanning / Horizontal Circles: The elevation angle φ remains fixed, while the azimuthal angle θ is varied.
- Rigid Arc: A semi-circular path in the $xz - plane$ is generated, with the plane rigidly rotated to always face the origin. This mode uses the Rodrigues rotation formula to reorient the source plane toward the target, as it required rotation about an arbitrary axis.

$$R = I + \sin \theta \cdot K + (1 - \cos \theta) \cdot K^2 \quad (32)$$

$$K = \begin{bmatrix} 0 & -k_z & k_y \\ k_z & 0 & -k_x \\ -k_y & k_x & 0 \end{bmatrix} \quad (33)$$

For each position along the trajectory, the plane is rotated such that its normal direction vector \vec{n} points towards to the origin, about which the sensor plane is defined.

Table 4.7.: Definition of vectors and symbols used in ray and plane calculations

Symbol	Description
$\vec{A} = (a, b, c)$	Position vector of the ray origin
$\vec{u} = (\alpha, \beta, \gamma)$	Direction vector of the ray
$\vec{P} = (l, m, n)$	A known point on the target plane
$\vec{n} = (\lambda, \mu, \nu)$	Normal vector of the target plane
t	Ray parameter defining point along the ray path
r	Radius in spherical coordinates
θ	Azimuthal angle in spherical coordinates
φ	Elevation angle in spherical coordinates

4.4.3. Implementation

The software model implements the theoretical foundation to provide a flexible framework for simulating ray projection and intersection calculations. The implementation follows a modular object-oriented approach with components that can be configured to represent different experimental setups.

Architecture Overview

The architecture consists of several key components:

- Core geometric objects (Planes, Areas, Lines) that encapsulate the mathematical properties
- Source Plane movement comprising translations and rotations
- Simulation engine for trajectory generation and intersection testing
- Configuration system for experiment setup
- Visualisation for result analysis
- User interface for interactive control

Where Class objects **Planes**, **Areas** represent:

Arc Rotation

To accurately simulate the interaction between the dynamic source plane and fixed sensor regions, the software model incorporates **three distinct rotation strategies**. These allow for controlled variation in the position and orientation of the source plane and

Table 4.8.: Optical Simulation Components

Component Type	Description
Planes	
Source Plane	Origin of light rays; can move along arc trajectories to simulate different lighting conditions
Sensor Plane	Fixed target plane where sensors are placed to detect light rays
Aperture Plane	Intermediate plane containing apertures that filter the light rays
Areas	
Sensor Areas (A0, A1, A2, A3)	Specific regions on the sensor plane that record light ray hits; used to measure illumination patterns
Aperture Areas (Aperture_A, Aperture_B)	Openings on the aperture plane that allow light to pass through; used to restrict FOV

facilitate flexible experimental scenarios. Each strategy models a different type of scanning behaviour and is implemented modularly within the `arcRotation.py` and `main.py` modules.

Rigid Arc

In this strategy:

- The source plane moves along a semicircular arc in the X-Z plane (constant radius).
- At each step, the plane is rotated to face the global origin (0, 0, 0) on the sensor plane.
- This simulates the rotation relative to unidirectional light emission source, like the Sun.
- All positions are precalculated, then rotated by 'tilt angles' in the X-axis. Visualised in figure 4.32

Horizontal Circles This method generates a set of circular paths in the XY plane, each at a different Z height (φ) angles:

- At each Z-level (elevation), the plane performs a full 360° rotation around the Z-axis, visualised in left figure 4.33
- Simulating scanning the sensor area from different elevation angles, while keeping the plane parallel to the XY plane.

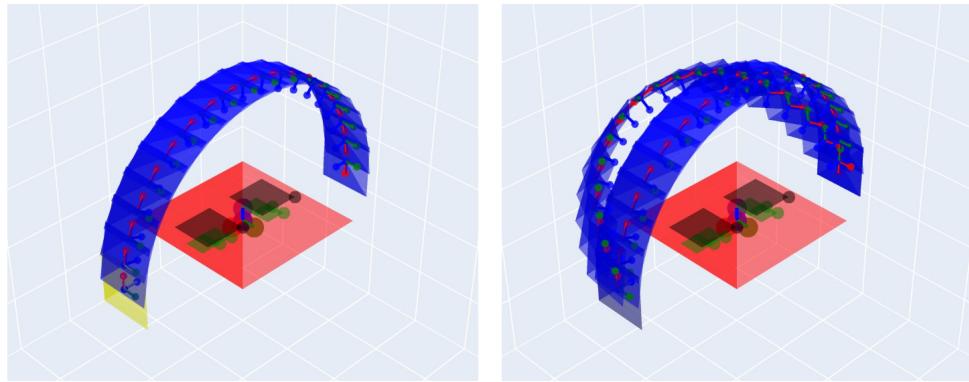


Figure 4.32.: Rigid Arc Scanning Mode - Tilt Angles Left (0°) Right ($0^\circ, 45^\circ$)

Vertical Circles In this mode, the source moves along vertical arcs (meridians), while scanning across azimuthal (θ) angles:

- At each azimuthal angle, the plane performs a 180° rotation around the Z-axis visualised in right figure 4.33.
- This movement style traces vertical circles, like longitude lines on a globe.

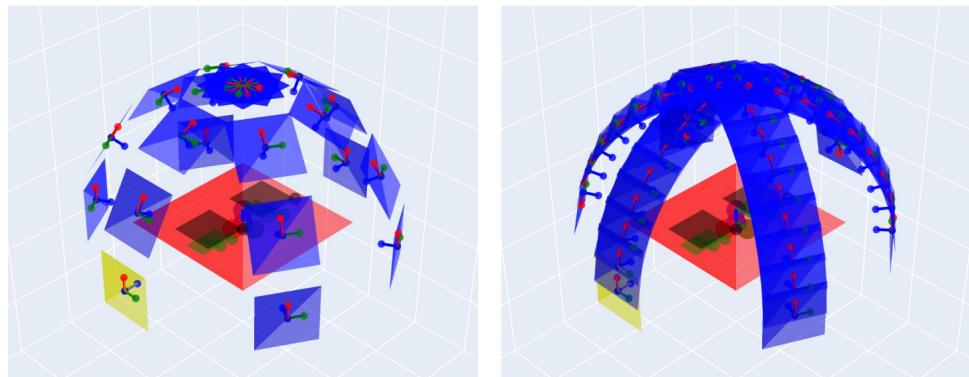


Figure 4.33.: Horizontal and Vertical Scanning Modes

Intersections

The intersection calculations are implemented in the `intersectionCalculations.py` module, with four main functions:

```

1  def direction_vectors(vectorA, vectorB):
2      """
3          Calculates the dot product between two vectors.
4          Used to check if ray direction and plane normal are parallel.
5          Args:
6              vectorA: First vector (Plane normal direction)
7              vectorB: Second vector (Ray normal direction)
8

```

```

9         Returns:
10            Dot product of the two vectors
11        """
12
13        check_direction = np.multiply(vectorA, vectorB)
14        check_direction = np.sum(check_direction)
15
16    return check_direction
17
18
19    def compute_t(nP, nA, nU):
20        """
21
22        Computes the parameter t in the line equation.
23
24        Args:
25            nP: Dot product of plane normal and plane position
26            nA: Dot product of plane normal and line position
27            nU: Dot product of plane normal and line direction
28
29        Returns:
30            Parameter t at which line intersects the plane
31        """
32
33    return (nP - nA) / nU
34
35
36    def calculate_intersection(line, t):
37        """
38
39        Calculates the intersection point using the line equation.
40
41        Args:
42            line: Line object with position and direction
43            t: Parameter value
44
45        Returns:
46            3D coordinates of the intersection point
47        """
48
49        x = line.direction[0] * t + line.position[0]
50        y = line.direction[1] * t + line.position[1]
51        z = line.direction[2] * t + line.position[2]
52
53
54        coordinates = np.array([x,y,z])
55    return coordinates
56
57
58    def intersection_wrapper(sensorPlane, line1):
59        """
60
61        Coordinates the intersection calculation process.
62
63        Args:
64            sensorPlane: Plane object with position and normal
65            line1: Line object with position and direction
66
67        Returns:
68            Intersection coordinates or None if no intersection exists

```

```

56     """
57
58     nU = direction_vectors(sensorPlane.direction, line1.direction)
59
60     if nU != 0:
61         nA = np.dot(sensorPlane.direction, line1.position)
62         nP = np.dot(sensorPlane.direction, sensorPlane.position)
63
64         x = compute_t(nP, nA, nU)
65         IntersectionCoordinates = calculate_intersection(line1, x)
66
67         return IntersectionCoordinates
68     else:
69         print(f" Intersection not possible for nU vector: {nU}")
90         return None

```

Listing 4.4: Ray-Plane Intersection Algorithm

After calculating the intersection point, the system checks if the point falls within the specified areas on the planes. This is handled by the `record_result` method in the `Areas` class:

```

1 def record_result(self, cords):
2     """
3
4         Determines if an intersection point falls within the area
5         boundaries.
6
7         Args:
8             cords: 3D coordinates of the intersection point
9
10
11         Returns:
12             1 if point is within area (hit), 0 otherwise (miss)
13
14         """
15
16         # True if intersection x coordinate is within area boundary (x
17         # min and x max)
18         if (self.position[0] - (self.width / 2) <= cords[0] <= self.
19             position[0] + (self.width / 2)
20                 and # True if intersection y coordinate is within area
21                 boundary (y min and y max)
22                     self.position[1] - (self.length / 2) <= cords[1] <= self
23                     .position[1] + (self.length / 2)):
24                         return 1
25
26         else:
27             return 0

```

Listing 4.5: Area Intersection Testing

This process runs incrementally, checking whether the lines intersect with apertures, and if so, checking for intersection with sensors. If both conditions are met, then 'illumination' occurs.

Assumptions The development of the software model and simulation framework involved several assumptions to enable tractable, consistent, and focused analysis. These assumptions are the basis for the implementation and set the boundary conditions for interpreting results.

Geometric

- Planes are rectangular and rigid: All *Plane* and *Area* objects are assumed to be flat rectangular surfaces.
- Local coordinate frames (*right, up, normal*) are assumed to form an orthonormal basis derived via the *right – hand* rule using a predefined *worldup* vector.
- Plane local coordinates are recalculated from the center position and orientation at each step using fixed width and length.

Line Emission and Direction

- Lines are emitted orthogonally to the source plane's surface (i.e., in the direction of the plane's normal vector).
- All lines are assumed to be infinite in length, with intersection checked against finite planes.
- Initial emission points for all lines are randomly distributed within the local bounds of the source plane.

Rotation and Arc Movement

- In rigid arc mode, the plane always faces the global origin regardless of position.
- For vertical and horizontal circle modes:
- Planes move along spherical arcs with constant radius.
- Primary and secondary rotation matrices are sequentially applied using defined axes.
- Rotations are idealised and simplistic: Instantaneous and discrete.

Intersection Physics and Hit Detection

- Intersection detection assumes ideal mathematical planes: Thickness, reflectivity, or surface texture are not modelled.
- A line is marked as a hit only if the intersection point lies within both the aperture area and a defined sensor area.

- No consideration is given to beam divergence, attenuation, or scattering.
- No physical laws (e.g., Snell's law) are implemented — the model uses only geometric approximation.

Model Configuration

A .JSON file is used to allow for easy setup of different experimental scenarios without modifying code:

- Detailed definition of planes, including position, orientation, and dimensions
- Sensor and aperture areas with specific positions and sizes
- Trajectory specifications for various movement types
- Simulation parameters (number of rays, iterations)
- Visualisation options - Static, or animated plot
- Debugging and performance settings

The `Config` class (Listing 4.6) loads and parses the configuration file, making parameters accessible throughout the application:

```

1
2     class Config:
3         def init(self, file_path=None, data=None):
4             if data is not None:
5                 self.load_from_dict(data)
6             elif file_path:
7                 with open(file_path, "r") as f:
8                     data = json.load(f)
9                     self.load_from_dict(data)
10                else:
11                    raise ValueError("Must provide either file_path or data")
12
13    def load_from_dict(self, data):
14        self.planes = data["planes"]
15        self.sensor_areas = data["sensor_areas"]
16        self.aperture_areas = data["aperture_areas"]
17        self.arc_movement = data["arc_movement"]
18        self.simulation = data["simulation"]
19        self.intersection = data["intersection"]
20        self.visualization = data["visualization"]
21        self.debugging = data["debugging"]
22        self.performance = data["performance"]
23        self.output = data["output"]
```

Listing 4.6: Model configuration - Config Class

Additionally available to the user is an interface, allowing for easy manipulation of the parameters, and visual representation of the sensor positions. Figure 4.34

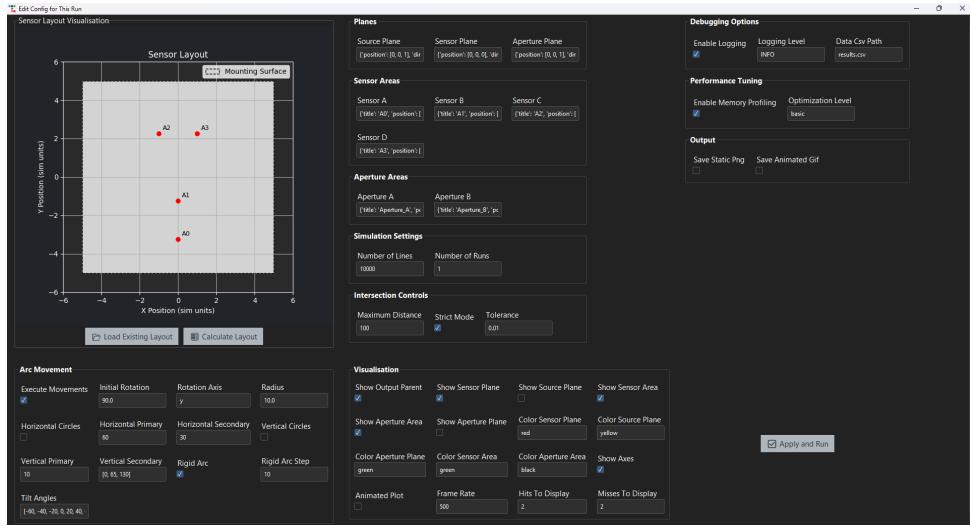


Figure 4.34.: Model Configuration Interface

Model Output

The simulation calculates how many light rays (lines) emitted from a movable source plane are successfully transmitted through the apertures and reach specific sensor areas. This output is stored in CSV files.

results.csv This file records summary statistics for each simulated position during the arc movement, containing:

Providing overall results for any given simulation run.

sensor_results.csv This file records individual sensor readings for each simulated position, containing:

4.4.4. Data Analysis and Evaluation

After the ray-tracing simulation completes, the data generated is analysed to evaluate the performance of the sensor topology.

This component evaluates the performance of the system at each step along the trajectory of the source plane. The analysis uses data stored in the *results.csv* and *sensor_results.csv* output files to calculate and visualise hit efficiency.

Table 4.9.: Simulation Output: Summary of Each Arc Position

Field	Example	Description
sim	0	Simulation run index
idx	4	Arc position index (step in movement sequence)
hits	6321	Number of rays that reached any sensor area
misses	3679	Number of rays that missed all sensors
ray count	10000	Total rays emitted during this simulation run
sim title	Basic	Custom label identifying the simulation configuration
runtime	1.2847	Execution time (seconds) for this movement type

Table 4.10.: Sensor Output: Ray Hits Per Sensor at Each Arc Position

Field	Example	Description
sim	0	Simulation run index
idx	4	Arc position index (step in movement sequence)
A0	2450	Number of rays hitting Sensor A0
A1	1783	Number of rays hitting Sensor A1
A2	1044	Number of rays hitting Sensor A2
A3	1044	Number of rays hitting Sensor A3

Result Analysis Function: `plot_hit_percentage()`

This script evaluates the system's ray interception efficiency across the arc rotation of the source plane, generating two plots:

Overall Hit Percentage per Arc Index This plot displays the proportion of emitted rays that successfully reach any of the defined sensor areas for each simulation index (arc position). The hit percentage is calculated as:

$$\text{Hit (\%)} = \frac{\text{Number of Hits}}{\text{Total Number of Rays}} \times 100 \quad (34)$$

Per-Sensor Hit Percentage Gaining spatial resolution of the intersection behaviours, the second subplot shows the relative 'illumination' of each sensor based on the total hits.

For every arc index, the number of rays detected by each sensor is normalised by the total number of hits. This allows identification of the directional coverage for each sensor.

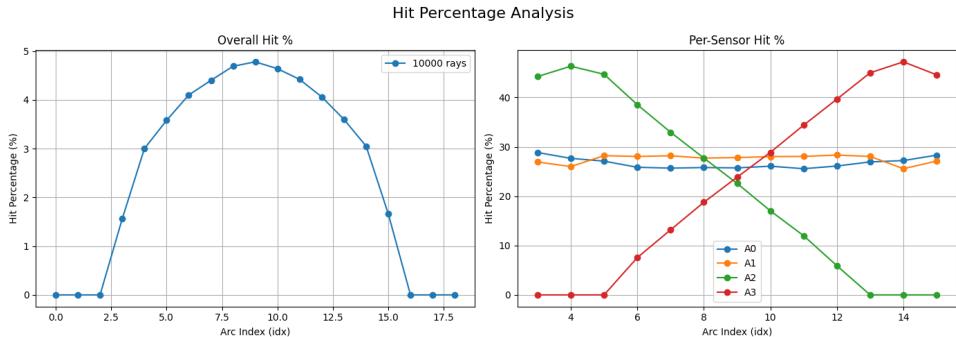


Figure 4.35.: `plot_hit_percentage()` Output Plot

Together these visualisations form an overview of the sensor system's performance and directional sensitivity.

Sensor Spatial Response Function: `sensor_surface_plots()`

While previous analysis plots visualise results per arc index, this function presents a higher-dimensional insight by mapping each sensor's illumination response across both the arc angle and the tilt angle. The underlying data is drawn from `sensor_results.csv` and the angular mapping in `rigid_arc_angles.csv`.

Each output is a 3D surface plot, Figure 4.36, where:

- The *x*-axis represents the **arc angle** (horizontal angular sweep),
- The *y*-axis represents the **tilt angle** (vertical angular sweep),
- The *z*-axis shows the **number of ray hits** for each sensor.

This visualisation allows for an evaluation of the angular coverage of each sensor and the orthogonality of the measurements.

Model Testing

To validate the functionality of the model test configurations were created. Each test scenario targets a specific feature or behaviour of the ray-plane interaction system, ensuring that the model behaves as expected.

The simulation framework was tested using several '.json' configuration files, each defining different spatial setups for the source, aperture, and sensor planes. The expected outcome of each test was determined analytically based on geometric relationships. Below is a summary of each test and its intended validation purpose:

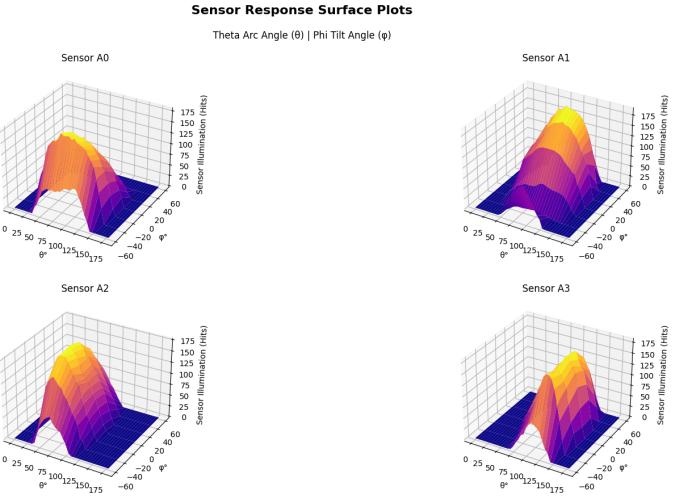


Figure 4.36.: `sensor_surface_plots()` Output Plot

- **`test_arc_rotation.json`**

This test evaluates the full arc movement of the source plane in a rigid semi-circular path, with multiple sensors and two distinct apertures. The simulation checks whether rays, emitted from various arc positions, are correctly filtered through the apertures and accurately detected by the sensors. *Expected result:* a non-uniform distribution of sensor hits across arc positions, demonstrating correct handling of dynamic source movement and occlusion logic.

- **`test_directly_below.json`**

A basic validation scenario where a single sensor is placed directly beneath a uniformly open aperture and source. The system is expected to demonstrate maximal efficiency under ideal alignment conditions. *Expected result:* High % ray intersection success with the sensor area.

- **`test_no_intersection.json`**

This test ensures that rays that are directed orthogonally to the sensor plane (i.e., no downward component). The source emits horizontally, and the sensor is placed far from the expected ray path. *Expected result:* zero valid intersections, confirming non-intersection conditions.

- **`test_off_center.json`**

In this configuration, sensors are slightly offset from the central axis of the source, with a wide aperture. This test is used to assess the spatial precision of the system when determining ray hits that fall near sensor boundaries. *Expected result:* a partial detection pattern, with hits concentrated on central sensors and misses on outer ones depending on angular spread.

- **test_with_aperture.json**

This test includes two apertures spatially separated along the y-axis and a single sensor aligned with one of them. It is intended to verify whether rays are correctly filtered by the aperture before reaching the sensor. *Expected result:* Most rays intersect outside aligned apertures, with sensor registering hits only through valid aperture intersecting paths.

Together, these tests provide functional validation of the core components: ray generation, source arc trajectory, aperture filtering, and intersection logic. The system's ability to handle edge cases (e.g., no intersection, misalignment) supports its robustness and practical viability for analysis and experimentation of potential sensor topologies.

Model Evaluation Function: Run Time and Hit Gain Efficiency

The second aspect focuses on the computational performance of the simulation in relation to the effects of increasing the number of rays. This is implemented in the `plot_runtime_vs_gain()` function, which processes `results.csv` to compute:

Average Runtime per Ray Count - The mean execution time required for the simulation of varying ray counts, providing a baseline for understanding the simulation's computational demands.

Marginal Gain in Hit Percentage - This quantifies how much additional hit accuracy is gained by increasing the number of rays. It is computed by taking the difference in hit percentage between successive ray counts. This helps to identify the point(s) of diminishing returns, where increasing the ray count leads to diminishing returns in terms of accuracy.

Cost per Percentage Gain - Further quantifying efficiency, this analysis calculates the cost per 1% gain in hit percentage. This expresses how many seconds of simulation time are required to improve accuracy by one percentage point, it provides insight into whether it is computationally justified to increase ray count.

$$\text{Cost per Gain} = \frac{\Delta \text{Runtime}}{\Delta \text{Hit Percentage}} \quad (35)$$

These results help in determining the optimal ray count against computational expense, ensuring that the simulation remains accurate and practical.

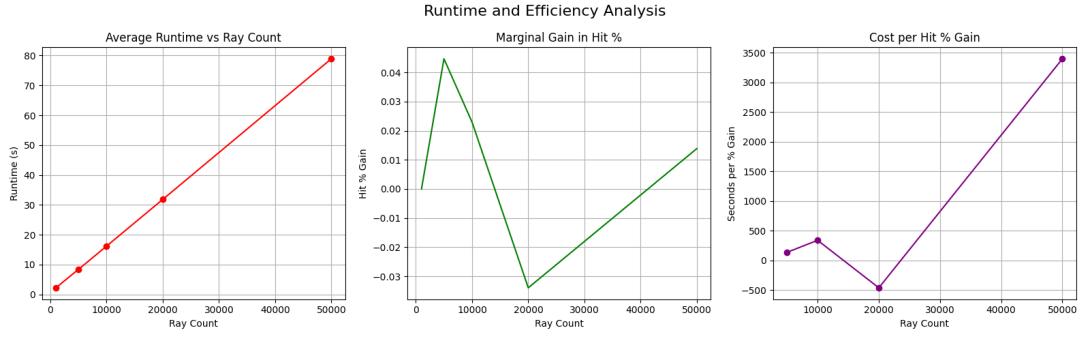


Figure 4.37.: Run Time and Hit Gain Efficiency Output Plot

Table 4.11.: Summary of Analysis Functions

Function	Metrics Generated	Purpose
<code>plot_hit_percentage _combined()</code>	Overall Hit Percentage Per-Sensor Hit Distribution	Assess ray interception performance and directional sensitivity
<code>plot_runtime_vs _gain()</code>	Average Runtime Marginal Gain in Hit % Cost per Gain	Evaluate efficiency vs. accuracy trade-offs in ray count configuration
<code>sensor_surface _plots()</code>	Sensor Response 3D Surfaces	Visualise angular response behaviour of each sensor across the full tilt/arc space

4.4.5. Model Results

Runtime and Efficiency Analysis

To evaluate the computational efficiency of the simulation model, a series of runs were performed using increasing ray counts. The aim was to determine how runtime scales with simulation complexity, and whether additional rays meaningfully improve hit accuracy.

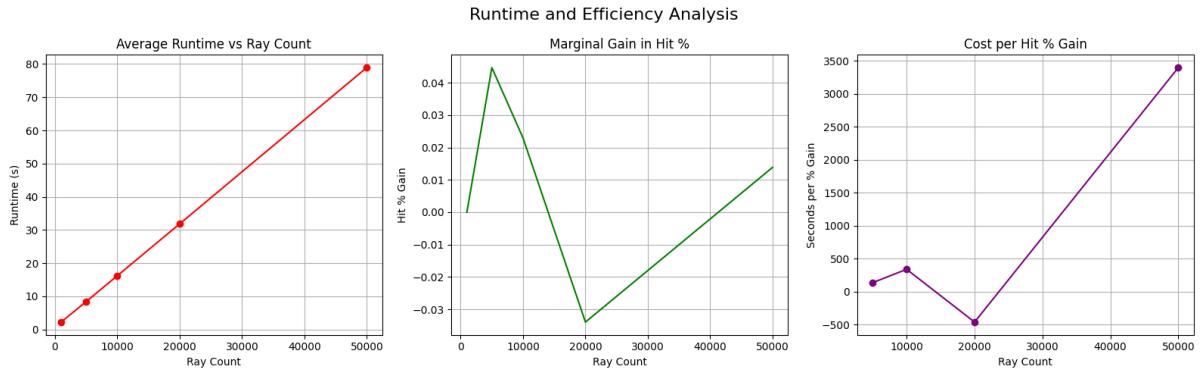


Figure 4.38.: Runtime and efficiency trends with increasing ray count. Left: Runtime increases linearly with ray count. Middle: Marginal gain in hit accuracy. Right: Cost in seconds per 1% hit gain.

Scalability: As shown in Figure 4.38 (left), the simulation runtime increases approx-

imately linearly with the number of rays. This behaviour suggests the implementation scales efficiently.

Accuracy vs. Rays: The centre plot illustrates the marginal gain in hit percentage between successive ray counts. Gains were most meaningfully prominent between 20,000 and 50,000 rays.

Cost per Gain: The rightmost plot quantifies computational cost per 1% gain in accuracy. It reveals that simulations using 50,000 rays are inefficient: they incur high runtimes with negligible improvement. In contrast, ray counts between 10,000 and 20,000 yield the most cost-effective results.

Utilisation: Based on these findings, simulations were conducted using **15,000 rays** to benefit from the best trade-off between runtime and accuracy.

4.5. Material Analysis and Selection

4.5.1. Material Selection and Requirements

The PCB will be operated in harsh environmental obstacles from strict requirements in both space and weight. PCB must withstand the extremities of immense pressures and vacuums, polarising temperatures, vibrations, impacts, space radiation and more. Generally, the materials for PCB made for space conditions are either polyimide or ceramic, as they can withstand extremely harsh conditions [42]. Commonly materials used for space PCB include polyimide, PTFE and alumina [43].

4.5.2. External Factors

The PCB needs to sustain itself when it is launched into and deployed to outer space from the extreme heat it will experience from -200°C in the shadow of a celestial body to over 200°C when sunlight exposes on it

Extreme Temperature Variations

Space exposes PCBs to extreme temperature fluctuations, which can affect their structural integrity and performance. Materials must maintain their thermal stability and dimensional consistency across a wide temperature range.

The temperature in space varies dramatically, ranging from extreme cold in the shadow of celestial bodies to scorching heat under direct solar exposure [44]. Materials expand and contract, with the magnitude of the dimensional change determined by the material's Coefficient of Thermal Expansion (CTE) [45]. When different materials in a PCB assembly have differing CTEs, the repeated cycles of expansion and contraction can result in considerable mechanical stresses at their interconnections [44].

These stresses may result in severe failures such as solder joint cracking, PCB layer delamination, and even circuit malfunction [44]. As a result, materials with naturally low CTE values, such as ceramic PCBs, are widely used in space applications to reduce these concerns [44]. A material's capacity to resist multiple temperature cycles without degradation is also critical for long-term reliability in space missions [43].

Ionizing Radiation

High-energy radiation, including cosmic rays, solar particles, X-rays, and UV radiation, exists throughout the space environment [44].

This ionising radiation poses a significant risk to the performance and durability of electronic components, particularly those found on PCBs. Radiation exposure can impair semiconductor performance, potentially causing data corruption via bit flips and resulting in a steady loss of material properties with time [46].

Celestial sources and the sun emit ionizing radiation that can disrupt the PCB's functionality and degrade the semiconductors' performance, hence, to protect the sensitive electronic components from being damaged by radiation, the PCBs must be made from radiation-hardened materials.

To mitigate these negative effects, radiation-hardened materials are frequently used in space-grade PCBs. Ceramic substrates, for example, are more resistant to radiation than many biological materials. Specialised coatings are also used to give another layer of protection to delicate electrical components [46].

The use of materials with intrinsic radiation resistance is an important design concern for ensuring the ongoing operation of electronic devices in the harsh radiative environment of space [47].

4.5.3. Internal Factors

Mechanical Stresses

The PCB, housing and aperture will experience mechanical stresses during the launch and deployment in the spacecraft, from the vibrations generated from the launch and deployment phase it can cause structural damage. To mitigate this issue there are in place shock-absorbing mechanisms that include the PCB material being flexible and conformal coating to safeguard the electronic components' integrity. Flexible PCBs are more effective at absorbing disturbances and vibrations compared to rigid PCBs.

PCBs are subjected to immense mechanical strains throughout a space mission's launch and deployment phases. The tremendous vibrations and shocks produced during launch, along with the high gravitational forces experienced, can put great stress on every component of the PCB [44].

Furthermore, the deployment of solar arrays and other spacecraft features may cause additional mechanical stress [44]. These forces can cause structural damage to PCBs, such as board cracking, layer delamination, and the breakdown of solder junctions that connect components [44].

To address these issues, designers frequently use shock-absorbing mechanisms, such as flexible PCB materials like polyimide, which are more effective at absorbing vibrations than rigid counterparts. Conformal coatings are also used to give an extra layer of protection against physical damage during launch and deployment [44]. Furthermore, precise PCB layout design is required to guarantee a more even distribution of mechanical loads across the board [44].

Outgassing and Vacuum

Outgassing is another internal factor which is when manufacturing the PCB, outgassing is a soldering wave defect that traps air within a PCB. This is a major issue as it can lead to the PCB impairment from the cavities or blowholes created by the air inside. This is often a result of defective manufacturing and poor material selection choices.

There is a strict size and weight limit for the PCBs because of the tight space in the spacecraft, thus the PCB must be compact and lightweight whilst maintaining equilibrium that it does neither compromise on the functionality and the structure of the PCB.

The near-perfect vacuum of space poses an additional notable challenge for PCB materials. Materials in this environment can undergo outgassing, which is the release of trapped volatile chemicals [43]. These outgassed compounds can have harmful impacts on delicate spacecraft equipment [48]. Optical instruments, such as cameras and telescopes, are especially susceptible to contamination by outgassed materials, which can deposit on their surfaces and degrade performance [44]. Similarly, thermal control surfaces can be influenced, affecting their ability to regulate the spacecraft's temperature [45].

Furthermore, the release of gases near high-voltage components can raise the risk of corona discharge, which could hinder electronic operations and cause damage [45]. To address these difficulties, the materials used in space-grade PCBs must have low outgassing qualities. Polymers such as polyimide and PTFE (Teflon) are widely used because of their extremely low outgassing properties, which help to protect the integrity of sensitive spacecraft components and systems [44].

4.5.4. PCB Material Selection

High-Tg FR-4

While not as extensively used as polyimide or ceramics in the most extreme space environments, high-Tg FR-4 laminates find application in scenarios where thermal conditions are less severe, such as within crewed spacecraft like the International Space Station [43].

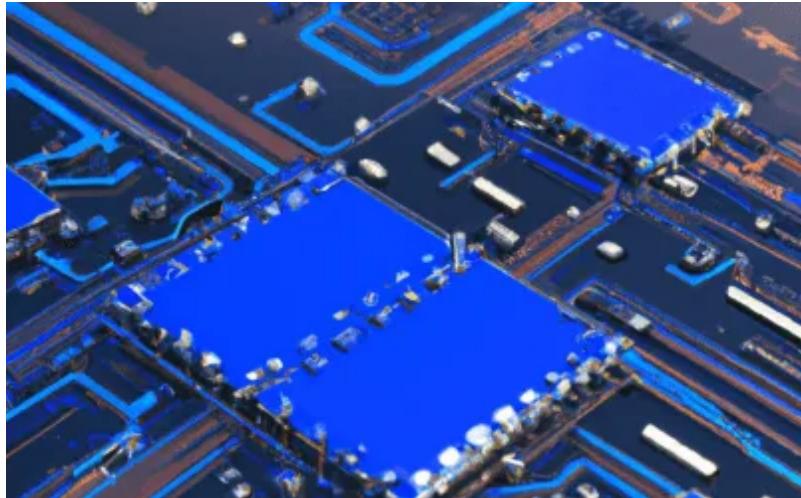


Figure 4.39.: FR-4 PCB [2]

These epoxy-based laminates possess a higher glass transition temperature (T_g) compared to conventional FR-4, providing greater stability at elevated temperatures. Epoxy laminates typically have a glass transition temperature between 150-170°C [43].

Using GRANTA EduPack 2022 R2, we found that FR-4 variations with dissipation factors (DF) of 0.015, 0.02, and more than 0.02 would be excluded from our nanosatellite PCB application. These materials were eliminated because they have larger signal losses at communication frequencies, reducing power efficiency in our power-constrained nanosatellite design. The increased signal attenuation will compromise the reliability of data transmission and could also raise thermal loads on the board. Furthermore, ordinary FR-4 materials often have lower glass transition temperatures, which poses dimensional stability problems during the severe heat cycling encountered in the space environment (-60°C to +100°C). These restrictions would negatively impact both the electrical performance and long-term dependability of the PCB, making these materials unsuitable for the demanding circumstances of the space environment.

Even FR-4.0 with $DF < 0.01$, while superior to other FR-4 varieties, has limits for space applications. Its glass transition temperature range (130-180°C) has a lower limit that may not be adequate for intense orbital thermal cycling. Its thermal expansion coefficient and mild outgassing properties in vacuum may jeopardise long-term reliability and contaminate sensitive optical components. Although more expensive, more specialised space-grade materials would be preferable for maximum performance in nanosatellite applications.

Polyimide

Research is conducted about materials suited for space PCBs, a prominent material used is Polyimide (PI), the common "space age" material with the highest performing class out of plastics.

Polyimide is a flexible polymer that is commonly used as a substrate material in space-grade printed circuit boards due to its inherent flexibility and lightweight nature [43]. This flexibility enables polyimide-based PCBs to effectively absorb mechanical stresses encountered during the dynamic stages of launch and spacecraft deployment. Furthermore, polyimide has strong thermal stability, which allows it to survive temperature fluctuations in space to some extent [43]. It also has low outgassing qualities, which are critical for avoiding contamination of sensitive spacecraft components [44]. However, polyimide is generally regarded as less suited for applications involving high levels of radiation exposure than other materials such as ceramics [43]. For over three decades, DuPont's Pyralux laminates and Kapton polyimide films have been widely used in the aerospace and defence markets, with applications ranging from multi-layer insulation to wire wrapping and flexible circuit interconnects on solar panel backplanes [49]. The Mars Rover Pathfinder also pioneered the use of adhesiveless laminate in space, using Pyralux AP material [49].

A material analysis was conducted using GRANTA Edupack and provides the general information about the material such as strengths and weakness shown in Appendix D. The main benefit of PI is that it has excellent heat resistance [260°C (500°F)] in continuous use, in tandem with having inherent fire-retardance and low smoke emission and excellent heat distortion temperature makes it a prime candidate for PCB. The limitations that Polyimide has is the high cost to produce, with the melt process being very difficult, making the manufacturing process of it be a slower process. It also has a relatively low impact strength, and worse elongation at break which can be a problem during the deployment process.

Alumina

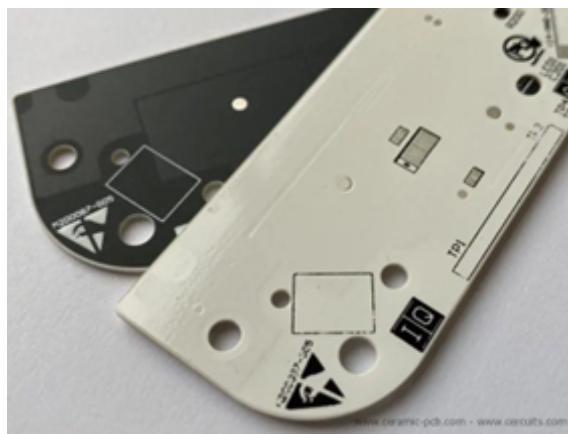


Figure 4.40.: Alumina Oxide PCB [3]

Alumina (Aluminium Oxide or Al_2O_3) is a ceramic that is a chemical compound of aluminium and oxygen, the 96% refers to the purity of the aluminium. A material analysis

was also conducted using GRANTA Edupack and is shown in Appendix D. In abrasive environments they can maintain a long service life due to their high hardness and wear resistance, which would be ideal for the space usage because it has one of the extreme environments imaginable. They also have a high-temperature resistance with a high melting point at 2000-2100°C and outstanding thermal stability. It also has excellent corrosion resistance which is ideal for space usage where it can be a corrosive environment. It also is lightweight with its density ranging from 3.69 to 3.73 g/cm³. The disadvantages of it are the brittleness of the ceramic and can fracture on impact which limits the lifespan in the space usage to mechanical shocks. It is also difficult to machine from the high hardness so initial costs for manufacturing and lead time are higher than other materials [50], [51].

PTFE(Teflon)



Figure 4.41.: PTFE/Teflon sheets [4]

Polytetrafluoroethylene (PTFE) is a synthetic Fluoropolymer they have been known to be chemically inert, low friction properties and has a high heat resistant.

PTFE, also known as Teflon, is renowned for its great electrical characteristics, particularly its low loss tangent and dielectric constant [52]. These characteristics make it an excellent candidate for high-frequency applications, which are common in space communication systems [52]. Furthermore, PTFE is highly resistant to chemicals and outgassing, making it ideal for the hostile space environment [52]. Teflon is also used as a solid lubricant for numerous moving parts in spacecraft because of its resistance to heat and non-stick qualities [53].

From the GRANTA Edupack it has a melting point that ranges from 315-339°C and can operates at its maximum service temperate from 250-270°C which allows the PCB to work in high temperature environments. Because it is chemically inert it is suitable for it to space PCB [54].

Copper Foils



Figure 4.42.: Copper foils [5]

High-performance copper foils are essential in space-grade PCBs to ensure excellent signal integrity and efficient heat dissipation [43]. Copper's excellent electrical and thermal conductivity makes it the ideal material for conductive layers on a PCB. In applications requiring larger current carrying capability, thicker copper traces are frequently used [55]. The quality and thickness of the copper foil are crucial parameters that influence both the electrical performance and thermal management capabilities of the PCB in the demanding space environment.

Kapton

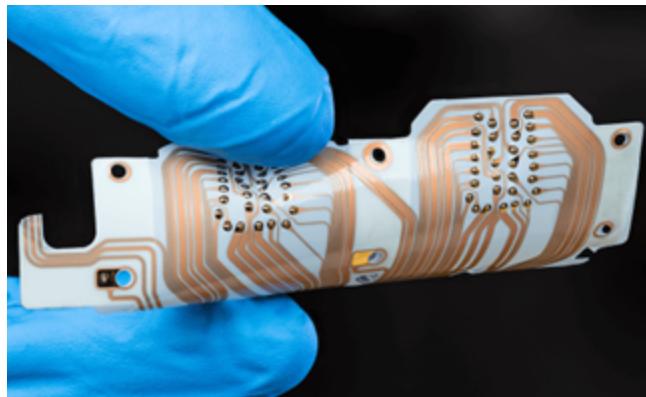


Figure 4.43.: Kapton PCB [6]

Kapton is highly radiation resistant and compatible with harsh conditions. Engineers typically use it to insulate cables and components in high vacuum chambers because it withstands radiation while having minimal influence on instrument base pressure [49]. Kapton, which can withstand temperatures of up to 400°C, can be combined with other materials like gold to create heat-resistant blankets for use in spacecraft. Kapton's flexibility and outstanding performance under harsh temperatures make it an ideal material for a variety of insulation and protection applications in space hardware [49].

Material Selection for PCB

The Comparing the elongation between the polyimide, alumina and PTFE, polyimide has the most ideal value being 75%–80% compared to alumina's 0.07%-0.09% or PTFE's 200%-400%, because having too little elongation can have a major risk of under deformation it will break, and having too much elongation causes the PCB to irreparably warp, having an in-between ensures it can deform when under forces without breaking. Polyimide has the lowest density which is ideal for having weight limit on the shuttle craft. Polyimide has the highest cost per kg which can be a major restraint on the budget whilst alumina and PTFE are far cheaper in comparison. All 3 have the necessary operating temperature criteria of space temperatures from -200°C to +200°C. From this it is decided that the best material for PCB is polyimide, because it has the required strength, operating temperatures and it has the best middle road on its elongation (%) and flexural modulus making it ideal for space usage.

4.5.5. CubeSat Chassis Material

The standard material that is used in the chassis of the CubeSat are aluminium alloys such as 6061 or 7075, which is to remain lightweight but without sacrificing strength. There's also need to thermal management to it to manage the fluctuations in space, commonly using multi-layer insulation to achieve this. The 4-digit designation represents what type of aluminium it is; the first digit can be 1 to 8 which represents which wrought it is. The second digit is often 0 meaning it is in the base form, any number other than this indicates it has been altered. The third and fourth digits identifies what individual alloys they are. A comparison table was created in GRANTA Edupack to compare all the Aluminium 6061 variants would be the best, and 6061-T651 was found to be the best out of all of them, as seen in the appendix.

Aluminium 6061- T651

Aluminium 6061-T651 has a good strength-to-weight ratio which offers a good balance between the strength and weight without compromising one to a detrimental degree. It's also strong with the tensile strength up to 320 MPa. It is used in many structural applications because it is strong by itself and is lighter than steel making it ideal in the automotive and aerospace industries. It is also highly recyclable and non-toxic so after usage it can be reused and does not cause harm to be in proximity with. It is also not suspectable to stress corrosion cracking making it ideal for space use when external and internal forces are applied towards it.

Aluminium 7075

Aluminium 7075 is one of the strongest aluminium available with a tensile strength listed at 580 MPa making it an ideal material for structural and load-bearing applications. Just like 6061 it is also lightweight with a good strength to weight ratio and highly recyclable. However, it is highly susceptible to stress corrosion cracking which is a major weakness especially for space applications from the external and internal forces applied to it.

6061-T651 vs 7075

In terms of strength Aluminium 7075 is considerably stronger than 6061, this makes 7075 a suitable material for applications requiring high strength such as Construction. However, 6061 is much more flexible with having better ductility making it easier to form weld into shape what the design requires. 6061 also has better resistance in corrosion due to things in space such as atomic oxygen oxidising the metal. And 6061 is cheaper and simpler production process because of its machinability compared to 7075. It is decided that the most suitable for the chassis should be 6061-T651 because of the corrosion resistance, and machinability keeping cost down, with the better ductility to avoid the chassis structure to snap under forces, whilst 7075 is stronger, they already are suitable strength wise in space that it is a moot point.

Emerging Materials and Future Trends

Emerging materials and future trends in space PCB technology are being shaped by advances in materials science and manufacturing techniques that try to solve the unique challenges provided by the space environment. Next-generation materials such as 2D materials, organic electronics, and metamaterials are being investigated for their potential to increase the performance and reliability of space electronics by offering severe temperature resistance, radiation shielding, and enhanced thermal conductivity [56].

Furthermore, hybrid and composite materials, including self-healing polymers and multifunctional carbon fibre composites, are being developed to withstand the extreme conditions of space, such as micrometeorite impacts and electromagnetic interference, thus improving the feasibility and safety of space exploration [57]. All these developments point to a future in which space PCB technology is more integrated, efficient, and capable of meeting the expanding demands of space missions.

4.5.6. Quality Assurance and Reliability Standards in Space PCB Manufacturing

NASA Standards

NASA has comprehensive standards to ensure PCB quality and reliability PCBs used in space missions. NASA-STD-8739.1 defines the workmanship standards for polymeric applications on electrical assemblies, such as conformal coatings used to protect PCBs in defence and aerospace applications [58]. The Goddard Space Flight Centre (GSFC) has released GSFC-STD-8001, which specifies quality assurance standards for the design, procurement, fabrication, and use of high-reliability PCBs in GSFC project mission hardware [59]. GSFC-STD-8002 provides the requirements for the validation of manufacturing processes for printed wiring assemblies using water-soluble fluxes [60]. The NASA PCB Working Group (PCB WG) is a helpful resource, offering expertise in PCB technology assessment and recommendations for quality assurance measures [52]. NASA generally uses polyimide-based laminates with glass reinforcements in its PCB constructions [61]. The agency also emphasises the necessity of visual acuity testing in accordance with NASA-STD-8739.6 or IPC-QL-653, and PCB inspectors must hold IPC-A-600 Certified IPC Specialist (CIS) accreditation to ensure complete and accurate quality control [59].

ESA Standards

The European Space Agency (ESA) also enforces strict requirements for materials and procedures used in space applications, including PCBs. These standards are extensively described in the European Cooperation for Space Standardisation (ECSS) series. ECSS-Q-ST-70-02C describes the thermal vacuum outgassing test protocols for screening space materials [42]. ECSS-Q-ST-70-10C specifies the requirements for qualified printed circuit boards [62], whereas ECSS-Q-ST-70-60C Corrigendum 1 specifies the requirements for their procurement [63]. Other relevant ECSS standards address a wide range of materials, mechanical parts, and procedures, including cleaning, heat testing, soldering, and crimping [63]. These standards are intended to ensure the reliability and performance of electronic assemblies in the challenging space environment.

IPC Standards

IPC, a global trade group for the electronics industry, has established a number of standards that are commonly used in the production of space-grade PCBs. IPC-2221 is a generic standard that covers practically every area of PCB design [43]. Specific subsection standards are IPC-2222 for rigid boards, IPC-2223 for flex circuits, and IPC-2226 for HDI structures [43]. IPC-6012 specifies the certification and performance standards for rigid PCBs, whereas IPC-6013 accomplishes the same for flexible printed boards [43].

IPC-A-600 specifies the visual inspection standard for PCB acceptability [43]. For surface mount designs, IPC-7351 provides critical guidelines [43].

For military and aerospace applications, IPC Class 3/A, as defined in IPC-6012 Class 3/A, denotes high reliability requirements [64]. Furthermore, the IPC 6012 Space Addendum specifies standards for class 3 boards used in the space and military avionics industries, including requirements to endure vibration, ground testing, and temperature cycling [65].

US Military Standards(MIL-SPEC)

The United States Military has developed its own performance criteria for PCBs used in defence and aerospace applications. MIL-PRF-31032 specifies the requirements for high-reliability, stiff PCBs, whereas MIL-PRF-50884 addresses flexible PCBs [42]. MIL-PRF-55110 covers stiff PCBs used in military applications [66]. These standards set strict requirements on material selection, design, manufacturing processes, and testing to assure PCB dependability in hostile operating environments [66]. Furthermore, standards such as MIL-STD-810 explain environmental engineering issues for military equipment, while MIL-STD-461 establishes requirements for electromagnetic compatibility [64]. Compliance with MIL-SPEC standards is commonly required for PCBs used in military and aerospace systems.

4.6. Thermal Analysis of the PCB

4.6.1. Hypothesis

The purpose of the thermal analysis is to validate that the Polyimide PCB and the components can operate under the space conditions. The operational temperature of Polyimide minimum is -240°C and the maximum is 260°C . If the components are within the operational temperature range, then it is validated to be operational under dynamic heat conditions, however if it exceeds it, then the PCB material, and the PCB design with the component locations must be altered and reworked. It is expected that the photodiodes area of the PCB to have the most thermal activity, with the lowest thermal activity occurring to the resistors.

4.6.2. Thermal Analysis Parameters

After a PCB CAD model is designed and material selection is completed, the thermal analysis for the PCB can now be conducted. The PCB CAD model was imported into ANSYS workbench using steady-state thermal analysis and a mesh was applied generated to the model to influences the accuracy and convergence of the simulation. Each off the

components have their respective materials applied to the model, such as the photodiode with silicon and aluminium, connectors being made of copper, nickel, zinc, amplifiers and resistors made from aluminium, and the PCB being made of polyimide. Then a convection of 22 °C was applied to the side of the PCB to simulate the transfer of heat of the PCB from the shuttlecraft it is housed in, assumingly room temperature of the spacecraft is the same as on Earth, it would be 22 °C. Each of the components have their respective temperatures when operating. Finally, two tests will be conducted using 200 °C and –200 °C of radiation applied to the entire PCB model to simulate when the PCB is facing to the sun, and when the PCB is in the shadows of celestial bodies respectively.

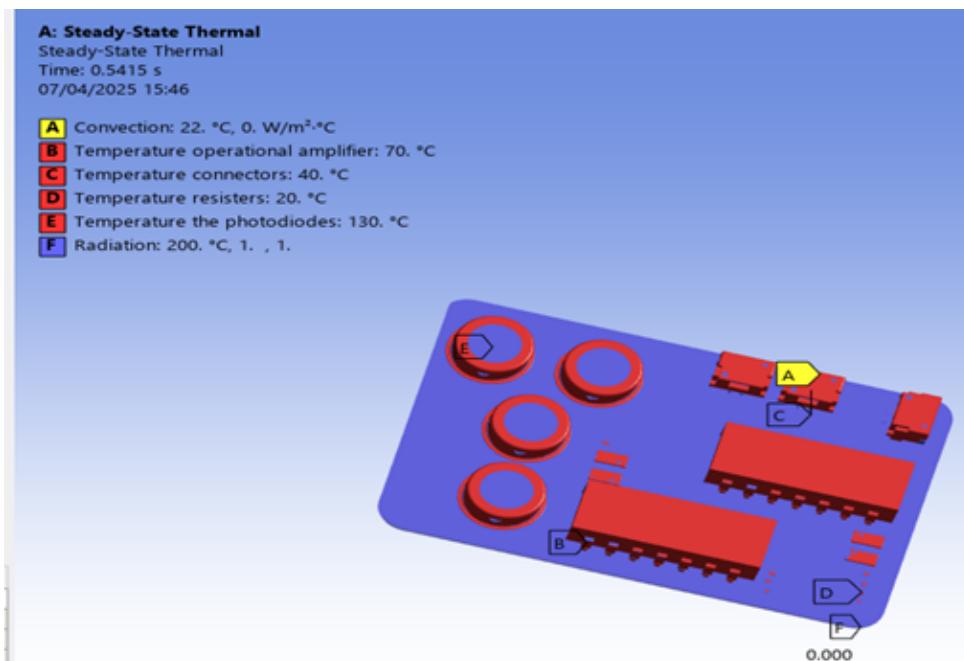


Figure 4.44.: PCB model with Parameters applied

4.6.3. Thermal results

4.6.4. Finite Analysis Evaluation 1 - Under 200 °C in space

The results for the PCB when facing towards the sun depicts the most heat being towards outside of the photodiodes area and the cooler parts are around the other components. As Figure 4.45 shows the maximum temperature experienced is 199.35 °C and the minimum temperature is –31.168 °C, this shows that the PCB can be operable when facing towards the Sun.

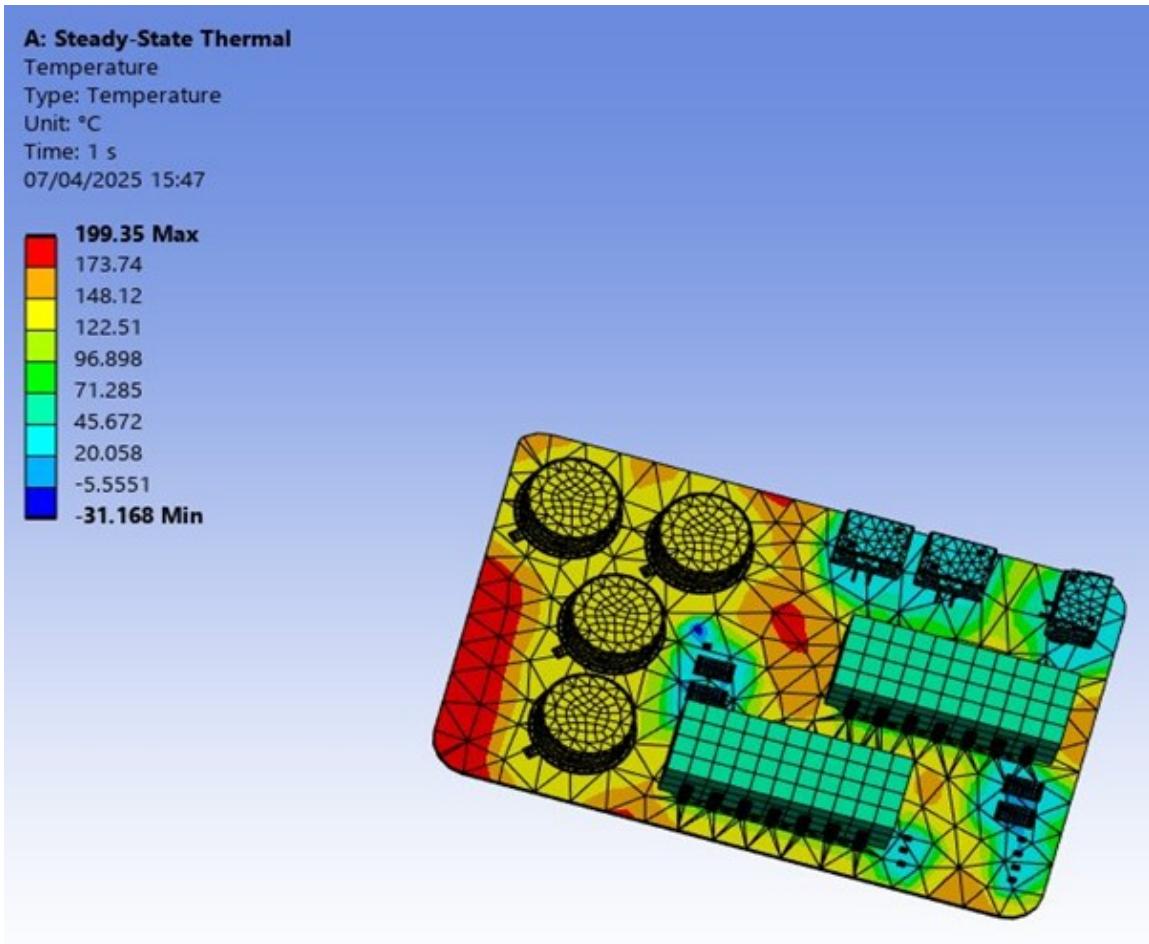


Figure 4.45.: Results under 200 °C radiation

Finite Analysis Evaluation 2 - Under -200°C in space

The results for the PCB, when it is in the shadows of a celestial body depicts the most heat coming from the photodiodes themselves which is to be expected as it creates the most heat in the PCB components followed by the amplifiers then connectors. As Figure 4.46 shows the maximum temperature experienced is 136°C and the minimum temperature is -173.83°C , this shows that the PCB can be operable when in the shadow of a celestial body.

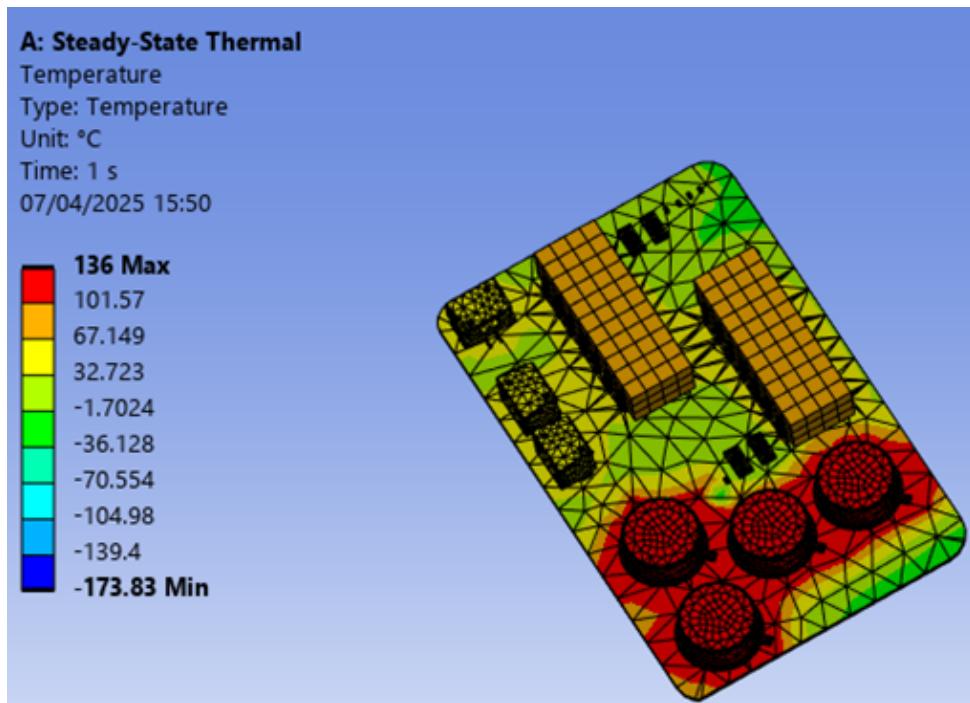


Figure 4.46.: Results under -200°C radiation

4.6.5. Results Evaluation

The results shown for both under 200°C and -200°C radiation shows that the PCB is operable under the harsh thermal conditions in space. That validates the design of the PCB and their component placements, and the material selection. The polyimide having about a clearance when facing the sun at 60°C and when in the shadows of a celestial object at -66°C shows that it would not be on the brink of melting or freezing during operation. The next step would be analysis the stresses and forces at work in space, unfortunately, a mechanical analysis of the stress cannot be conducted because it requires further work with the complete work of the entirety of the CubeSat chassis and all the power source such as the solar panels.

4.7. CubeSat Chassis Design

For the purposes of visualization, there were 2 potential designs for the CubeSat chassis. The CubeSat chassis were design dependant on where the PCB would be mounted. The first CubeSat chassis design was taken from GrabCAD.com and an aperture is mounted to this design to show where they would be mounted. The second CubeSat chassis was based on a design found online and recreated as much as possible, however this did not have any measurements given so dimensions were assumed.

4.7.1. CubeSat Chassis Design 1

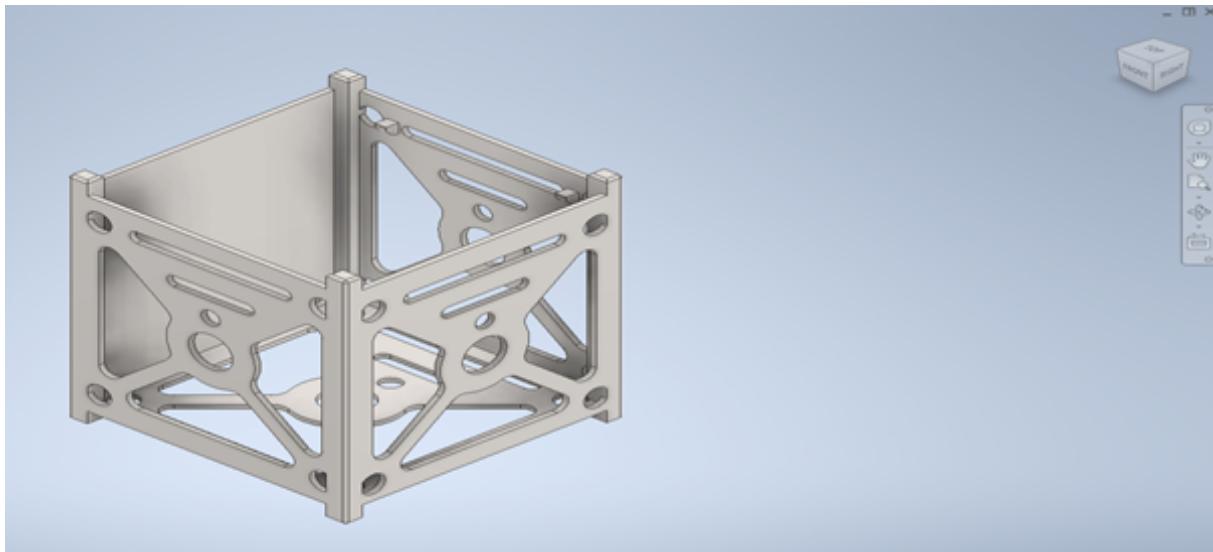


Figure 4.47.: Front view of the first CubeSat chassis design

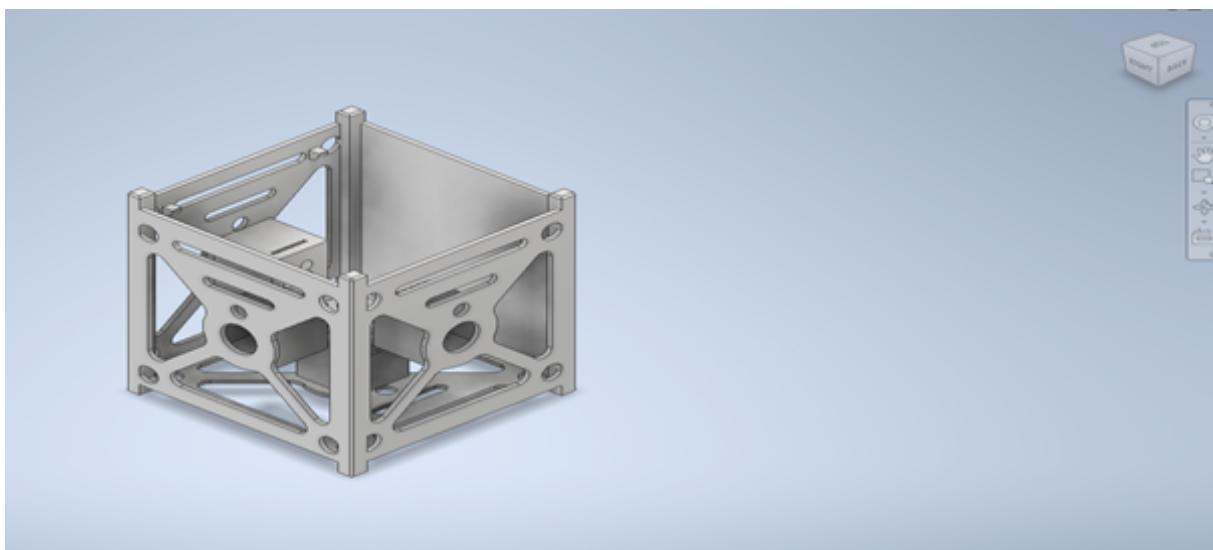


Figure 4.48.: Angular view of the first CubeSat chassis design

The first design taken from <https://grabcad.com/library/cubesat-1u-5> as seen in Figure 4.47 has exposed holes made to allow for easy removal, the circular holes at the centre are to let the light to expose to the aperture where the PCB photodiodes are located. The software used to show these are AutoCAD Inventor, with a different angle and the aperture shown in Figure 4.48.

4.7.2. CubeSat Chassis Design 2

The second hypothetical design is a much more basic skeleton frame made to house the aperture as seen below in Figure 4.49 and Figure 4.50.



Figure 4.49.: Front view of 1U Cubesat Skeleton Chassis [7]



Figure 4.50.: Angular view of 1U Cubesat Skeleton Chassis [7]

This was recreated in inventor with rough estimation on the thickness of the PCBs and skeletons itself and the apertures. The PCBs are mounted on top of each other compared to being the aperture housing the PCB mounted on each side of the CubeSat, the PCB cannot be more than 96×96 mm. As seen below in Figure 4.51 and Figure 4.52, are the CubeSat Chassis with and without the PCBs, additional angles and drawings are found in the appendix.

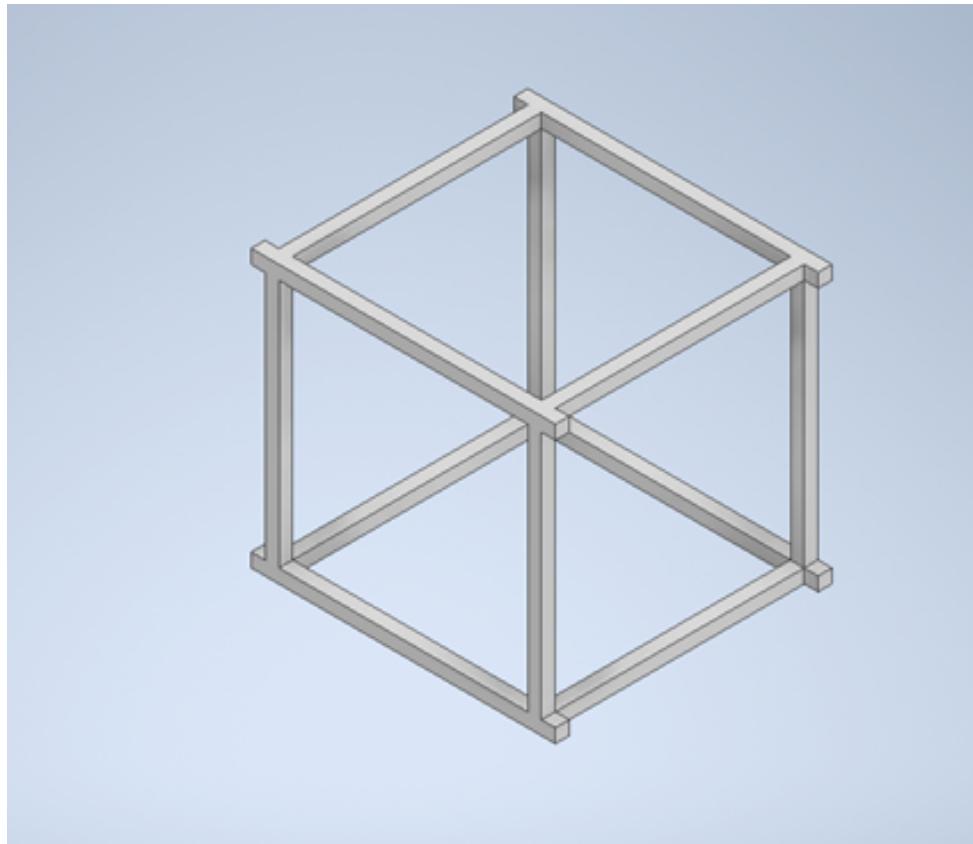


Figure 4.51.: Second chassis design without PCBs

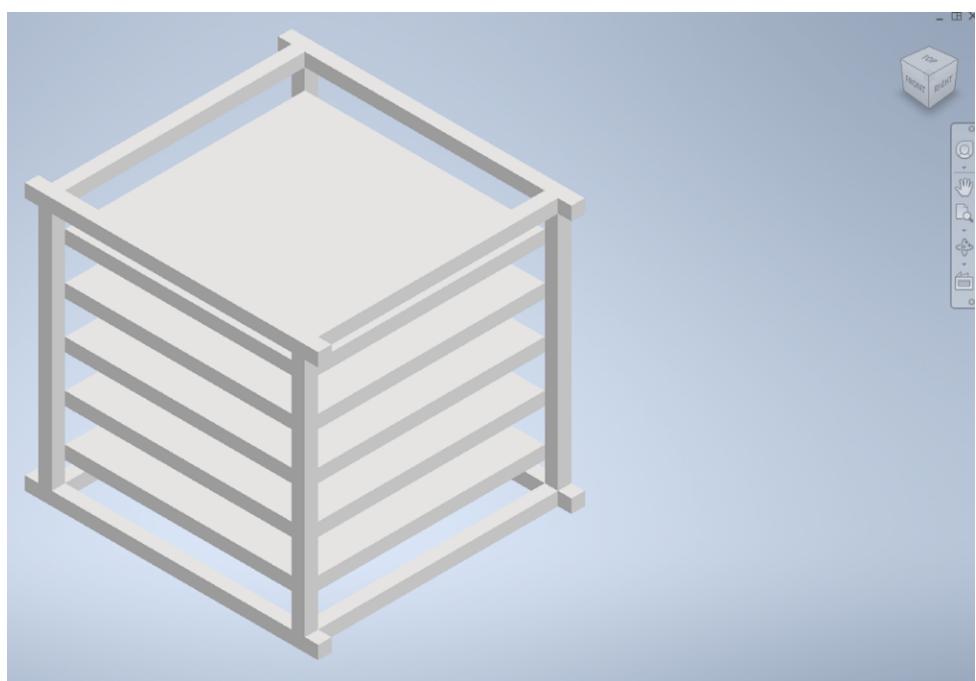


Figure 4.52.: Chassis with the aperture

5. Results

5.1. Sensor Characterization

TO FINISH

5.2. Amplification Performance

This section provides results of the amplifier performance.

5.3. Photodiode Angular Response

This section discusses the results of the response of the solar sensor to angular changes of the light source.

5.4. Enclosure Effectiveness

The final design of the enclosure is shown in Figure 4.10. The enclosure was designed and modified throughout its various iterations to be lightweight and portable, while still providing adequate protection for the internal components. The design also allowed for easy access to the internal components for maintenance and troubleshooting. The enclosure was deemed a successful design as it met the requirements of being a stable and portable platform for the internal components, as in the Photodiode array. Problems with previous designs were addressed in the final design iteration, such as the need for easier access to the anode and cathode connections of the photodiode array, which was achieved by using a removable cover, the rail-lock system, for the enclosure. This system allowed for the photodiode array to be easily mounted and secured into position. This system also allowed for the height of the enclosure to be adjusted so that the surface that mounted the glass and the aperture was at a suitable height for when it was in use with the RED testbench.

5.5. Data Acquisition System (DAQ) System Evaluation

The DAQ was considered a success, it was able to take recordings in realtime of all four photodiodes and send them to a computer for post-processing. In its final design the overall system that included the Python script as well as the Arduino, was able to perform its task of generating both data in csv format that allowed comparison with simulated input, as well as the addition of noise filtering.

5.6. Comparative Analysis

To assess the accuracy of the developed simulation model, results from a full simulation run were compared against experimental data collected using the same sensor topology and an equivalent arc motion sequence.

Figure 5.1 presents a side-by-side comparison:

- **Left:** Simulated illumination per sensor, expressed as the percentage of total emitted rays that intersected with each sensor area, plotted against the arc rotation angle.
- **Right:** Experimentally measured voltages from each physical sensor during the emitter's arc sweep. Signals have been filtered to reduce noise and highlight the response envelope.

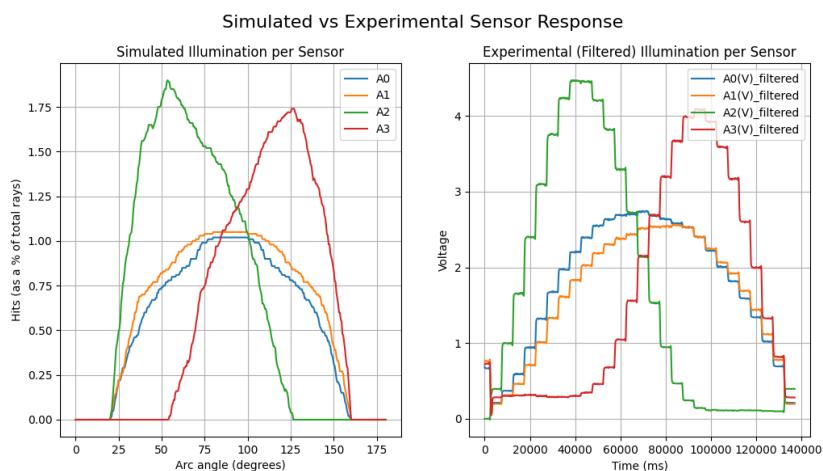


Figure 5.1.: Model and Physical Results Comparison

Key Observations

1. **Overall Response Pattern:** Both plots show a clear progression of peak response across the sensor array. Sensor A2 activates first, followed by A1, A0, and then A3

— consistent with the expected spatial sequence during the rotation.

2. **Peak Alignment:** The positions of the peak responses in the simulated and experimental plots align closely, suggesting that the source plane's movement and orientation in the simulation accurately reflect real-world conditions.
3. **Response Smoothness:** The transition between sensor activations is smooth in both data sets. This indicates realistic spatial spreading of rays in the simulation, and good agreement with physical measurements.

Interpretation

This result provides strong validation for:

- The geometric modelling of planes, areas, and ray emission in the simulation framework.
- The arc rotation logic, particularly the “rigid arc” strategy where the plane consistently faces the origin.
- The accuracy of the designed sensor layout and aperture configuration.

5.6.1. Breadboard vs. Stripboard Results

5.6.2. Iteration Improvements Analysis

5.6.3. Performance Against Design Requirements

The performance ...

5.6.4. Design Evolution Assessment

The what now?

5.7. System Limitations And Considerations

This section discusses the limitations and future work.

5.7.1. Renewable Energy Demonstrator (RED) testbench limitations and impact

The Renewable Energy Demonstrator (RED) device borrowed from the EPS team [21] proved to be valuable as a testbench for our project, but several limitations affected our ability to gather comprehensive data for accurate sun vector determination.

Servo Motor Limitations

There were at least two issues with the Servo Motors which reduced our ability to gather data:

- Inability to reach the target angles requested in code without physical intervention, presumably due to insufficient torque or gearing issues.
- Control deadband did not allow us to gather a large enough sample (small-angle adjustment) to create a LUT that would allow the prediction of light position by interpolation.

When attempting to move from 0° to 90° , for example, the servomotors would often stall, requiring manual assistance to “help” the arch reach the desired position. This introduced inconsistency in our testing methodology and required manual adjustment using protractors.

Signal Interference Issues

The RED’s Power Supply interference made the signal very noisy. However due to the nature of the high frequency noise and low frequency signal, it meant that it was relatively easy to filter using the `scipy.signal` library [39], detailed in Section 4.2.3. Further, arch position changing without pressing the button likely due to noise in the RED from the Power Supply made testing slightly more annoying, but it was not happening often enough to disrupt testing too much, it was considered an acceptable quirk.

Impact on Look-Up Table development

The most significant impact to the project caused by these issues with the RED are that we were unable to generate a LUT or a "Sensor Response Surface" plot using the prototype, as the angle precision from the arch was not good enough. Originally, our plan included:

- Taking measurements at fine angular increments (every 5°) across the hemisphere
- Mapping between sensor readings and light source position
- Using said mapping to develop interpolation algorithms for positions between measured points
- Validating position determination accuracy across the sensor’s field of view

The combination of these limitations of the RED testbench made it impractical to gather the comprehensive dataset required to fulfill these goals, as without a reliable dataset across the entire FOV it is not possible to predict the location of light. This

limitations ultimately directly affected the ability to detect the light position, and we could only take some readings and compare them to the simulated environment.

5.7.2. Aperture placement accuracy

Another big limitation on accuracy of the readings will be the manually placed apertures. As shown earlier in Figure 4.5, the apertures are far from perfectly in the middle. This will have a huge impact on the accuracy of the readings especially when compared to a simulated environment which have perfectly placed apertures. However, if we had been able to create a LUT using the real measurements with aperture as-is, the results could have theoretically been used to read the location of light. Unfortunately, as stated above, that was also impossible due to the limitations of the testbench. These two main factors were the main cause of not being able to perform a calibration and perform actual readings to test the ability to read light locations.

5.7.3. Model Limitations and Future Work

- The vertical scale differs between plots figure 5.1: simulated hits are relative percentages, while the experimental data reflects voltage. A calibration factor could be introduced in future to improve this correspondence.
- Minor asymmetries in the experimental data may arise from mechanical inaccuracies or misalignment, which are not currently modelled.
- Future improvements may incorporate ray divergence, non-ideal optics, or sensor sensitivity profiles to further refine realism.

6. Conclusions

This project has successfully met its objective of developing a cost-effective and reliable analogue sun sensor system suitable for attitude determination in Low Earth Orbit (LEO) nanosatellite missions. Through the integration of hardware prototyping and software simulation, a robust methodology was established to evaluate the viability of photodiode-based sun sensing solutions.

A functional photodiode array was constructed, accompanied by a custom-designed signal conditioning circuit incorporating a transimpedance amplifier and post-amplification low-pass filtering. The system output was digitised using a Arduino-based data acquisition system, with post-processing conducted via Python to apply digital filtering and interpret results.

The software model, implemented in Python, facilitated accurate simulation of ray-plane interactions using geometric approximation. It enabled flexible configuration of sensor and aperture topologies, and supported visual and quantitative analysis of illumination results across a range of incident angles and positions. Comparative analysis between experimental and simulated data demonstrated strong correlation, validating the fidelity of the model and the underlying assumptions.

While the Renewable Energy Demonstrator (RED) testbench introduced some mechanical limitations, such as inconsistent servo performance and control noise, these were effectively mitigated through calibration, manual adjustments, and signal filtering techniques.

Ultimately, this project delivers a validated hybrid sun sensing system — combining practical prototyping with a software model.

7. Future Work

Enhanced Aperture Design and Manufacturing The current prototype utilized manually placed apertures, resulting in alignment inconsistencies that affected measurement accuracy. Future iterations should explore precision manufacturing techniques such as photolithography or laser etching to create apertures with consistent placement and sharper edges. Additionally, testing alternative aperture geometries could optimize the sensor's field of view and accuracy across different incident angles.

Advanced Calibration Techniques Development of a comprehensive Look-Up Table (LUT) and interpolation algorithm would enable more precise attitude determination. Future work should include deploying the sensor on a high-precision platform to map responses across the entire field of view, generating a detailed calibration map.

Environmental Qualification and Testing While thermal simulations suggested the viability of the design in space conditions, physical testing in thermal vacuum chambers would validate the sensor's performance under actual space-like conditions. Radiation testing would also be necessary to assess component degradation over an extended mission lifetime.

Discrimination capabilities Methods to avoid detecting sun reflections from the Earth, Moon, or spacecraft components should be investigated. Possibilities include implementing intensity thresholds, spectral filtering, or temporal signature analysis to distinguish direct solar illumination from reflected sources.

Bibliography

- [1] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signals and Systems*. India: Pearson, 2013.
- [2] 86PCB, “Fr-4 pcb: Material properties, thickness selection, and application guide - 86pcb,” Jul 19 2024. [Online]. Available: <https://86pcb.com/blog/fr-4-pcb-material-properties-thickness-selection-and-application-guide/>
- [3] CERCuits, “Aluminum oxide pcb (alumina),” 2025. [Online]. Available: <https://ceramic-pcb.com/aluminum-oxide-pcb-al2o3/>
- [4] Imimg, “Teflon/ptfe sheets,” 2025. [Online]. Available: <https://3.imimg.com/data3/JP/YY/MY-1996077/teflon-ptfe-sheets.jpg>
- [5] V. Solutions, “Copper foil,” 2022. [Online]. Available: <https://viin-solution.com/images/newimage/Copper-Foil2.jpg>
- [6] Goodfellow, “Kapton,” 2025. [Online]. Available: https://www.goodfellow.com/media/wysiwyg/kapton-image-2_1_.png
- [7] A. O. Inc., “1u cubesat skeleton chassis,” 09-08 2021. [Online]. Available: <https://satsearch.co/products/aphelion-1u-cubesat-skeleton-chassis>
- [8] J. Puig-Suari and C. Turner, “Development of the standard cubesat deployer and a cubesat class picosatellite,” 2001.
- [9] K. S. Balaji, B. S. Anand, P. M. Reddy, V. C. B, M. D. P. Lingam, and V. K, “Studies on attitude determination and control system for 1u nanosatellite,” *ICCPCT*, pp. 616–625, 2023.
- [10] I. Lopez-Calle and A. I. Franco, “Comparison of cubesat and microsat catastrophic failures in function of radiation and debris impact risk,” *Scientific Reports*, vol. 13, no. 1, -01-07 2023.
- [11] W. Guanghui, P. Shum, X. Guoliang, and Z. Xuping, “Position detection improvement of position sensitive detector (psd) by using analog and digital signal processing,” in *2007 6th International Conference on Information, Communications and Signal Processing*, 2007, pp. 1–4, iD: 1.

- [12] P. Ortega, G. López-Rodríguez, J. Ricart, M. Domínguez, L. M. Castañer, J. M. Quero, C. L. Tarrida, J. García, M. Reina, A. Gras, and M. Angulo, “A miniaturized two axis sun sensor for attitude control of nano-satellites,” p. 1623, -10 2010.
- [13] S. Dwik* and N. Somasundaram, “Modeling and simulation of two-dimensional position sensitive detector (psd) sensor,” p. 744, -11-30 2019.
- [14] F. J. Delgado, J. M. Quero, J. García, C. L. Tarrida, J. M. Moreno, A. G. Sáez, and P. Ortega, “Sensosol: Multifov 4-quadrant high precision sun sensor for satellite attitude control,” in *2013 Spanish Conference on Electron Devices*, 2013, pp. 123–126, iD: 1.
- [15] A. Ali, K. Ullah, H. U. Rehman, I. Bari, and L. M. Reyneri, “Thermal characterisation analysis and modelling techniques for cubesat-sized spacecrafts,” *The Aeronautical Journal*, vol. 121, no. 1246, p. 1858, -10-17 2017.
- [16] A. Raslan, G. Michna, and M. Ciarcia, “Thermal simulation of a cubesat,” Research paper, 2019. [Online]. Available: https://www.researchgate.net/publication/335123882_Thermal_Simulation_of_a_CubeSat
- [17] A. Dhariwal, N. Singh, and A. K. Kushwaha, “Structural analysis of 1u cubesat designed for low earth orbit missions,” in *2023 International Conference on Innovations in Communication, Automation and Technology (ICICAT)*. IEEE, 06 2023, p. 1.
- [18] F. et. al., “Analysis of received power characteristics of commercial photodiodes in indoor los channel visible light communication,” *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 7, pp. 164–172, Nov 7 2017. [Online]. Available: <https://doi.org/10.14569/IJACSA.2017.080722>
- [19] N. A. Fortune, “A short guide to using python for data analysis in experimental physics,” Authorea Preprints, July 2021. [Online]. Available: <https://www.authorea.com/doi/full/10.22541/au.152961025.52816543?commit=338281975198a5f5599acd5c0cb0535a1155bea5>
- [20] G. Koukis and V. Tsoussidis, “Satellite-assisted disrupted communications: IoT case study,” *Electronics*, vol. 13, no. 1, -12-20 2023.
- [21] N. I. Shopov, S. Hannisse, and S. Gupta, “Renewable energy demonstrator (red),” Glasgow Caledonian University, Tech. Rep., 2022.
- [22] G. Keiser, *Fiber Optic Communications*, 1st ed. Singapore: Springer, 2021.
- [23] P. Horowitz and W. Hill, *The Art of Electronics*, 3rd ed. 32 Avenue of the Americas, New York, NY 10013-2473, USA: Cambridge University Press, 2015.

- [24] K. Rajan, M. Samykano, K. Kadirgama, W. S. W. Harun, and M. M. Rahman, “Fused deposition modeling: process, materials, parameters, properties, and applications,” *The International Journal of Advanced Manufacturing Technology*, vol. 120, no. 3-4, p. 1531, -02-26 2022.
- [25] J.-Y. Lee, J. An, and C. K. Chua, “Fundamentals and applications of 3d printing for novel materials,” *Applied Materials Today*, vol. 7, p. 120, -06 2017.
- [26] E. Yankov and M. P. Nikolova, “Comparison of the accuracy of 3d printed prototypes using the stereolithography (sla) method with the digital cad models,” *MATEC Web of Conferences*, vol. 137, 2017.
- [27] S. K. Tiwari, S. Pande, S. Agrawal, and S. M. Bobade, “Selection of selective laser sintering materials for different applications,” *Rapid Prototyping Journal*, vol. 21, no. 6, pp. 630–648, 2015.
- [28] L. C. Geng, X. L. Ruan, W. W. Wu, R. Xia, and D. N. Fang, “Mechanical properties of selective laser sintering (sls) additive manufactured chiral auxetic cylindrical stent,” *Experimental Mechanics*, vol. 59, no. 6, p. 913, -03-06 2019.
- [29] X. Niu, S. Singh, A. Garg, H. Singh, B. Panda, X. Peng, and Q. Zhang, “Review of materials used in laser-aided additive manufacturing processes to produce metallic products,” p. 282, -10-01 2018.
- [30] A. C. D. Leon, Q. Chen, N. B. Palaganas, J. O. Palaganas, J. Manapat, and R. C. Advincula, “High performance polymer nanocomposites for additive manufacturing applications,” *Reactive and Functional Polymers*, vol. 103, p. 141, -04-19 2016.
- [31] W. Ameen, “Exploring the manufacturability and quality of small-scale hole via fused deposition modeling technology,” -02-12 2024.
- [32] M. Altuğ, “A comparison study in terms of dimensional accuracy and precision of 3d modeling,” *International Journal of Innovative Engineering Applications*, vol. 6, no. 1, p. 24, -06-28 2022.
- [33] E. Yankov and M. P. Nikolova, “Comparison of the accuracy of 3d printed prototypes using the stereolithography (sla) method with the digital cad models,” *MATEC Web of Conferences*, vol. 137, 2017.
- [34] N. M. Pu’ad, R. H. A. Haq, H. M. Noh, H. Z. Abdullah, M. I. Idris, and T. C. Lee, “Review on the fabrication of fused deposition modelling (fdm) composite filament for biomedical applications,” *Materials Today: Proceedings*, vol. 29, -06-18 2020.

- [35] R. Srinivasan, K. N. Kumar, A. J. Ibrahim, K. V. Anandu, and R. Gurudhevan, “Impact of fused deposition process parameter (infill pattern) on the strength of petg part,” *Materials Today: Proceedings*, vol. 27, p. 1801, -04-19 2020.
- [36] T. S. Srivatsan, “Materials processing handbook, Joanna R. Groza, James F. Shackelford, Enrique J. Lavernia, and Michael T. Powers, Editors: CRC Press, Boca Raton, FL, 2007, 670 pages, ISBN: 978-0-8493-3216-6,” *Materials and Manufacturing Processes*, vol. 27, no. 10, pp. 1146–1147, 2012.
- [37] Unionfab, “The ultimate guide to pla bed temperature,” Jan 20 2025. [Online]. Available: <https://www.unionfab.com/blog/2024/06/pla-bed-temperature>
- [38] Atmel, “Atmega328p 8-bit avr microcontroller with 32k bytes in-system programmable flash datasheet,” 2015. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
- [39] T. S. community, “butter scipy v1.15.2 manual.” [Online]. Available: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html#scipy.signal.butter>
- [40] ——, “filtfilt — scipy v1.15.2 manual,” 2025. [Online]. Available: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.filtfilt.html>
- [41] G. L. G. Burbui, U. Reggiani, and L. Sandrolini, “Prediction of low-frequency electromagnetic interferences from smps,” *IEMC*, vol. 2, pp. 472–477, 2006.
- [42] W. Leverett, “Satellite technology: Pcb in space exploration and communications | abl circuits,” Feb 5 2024. [Online]. Available: <https://www.ablcircuits.co.uk/blog/pcb-satellite-exploration-communication/>
- [43] C. D. Systems, “Applying ipc pcb design standards | cadence,” 2025. [Online]. Available: <https://resourcespcb.cadence.com/blog/applying-ipc-pcb-design-standards-cadence>
- [44] Proto-Electronics, “Space-grade pcbs: challenges in designing electronics for extreme environments,” Jan 11 2024. [Online]. Available: <https://youtu.be/ZgLfodLIgcI>
- [45] 911EDA, “Space pcb design challenges: Engineering for the cosmos,” 2024. [Online]. Available: <https://www.911eda.com/articles/space-pcb-design-challenges/>
- [46] Gatema, “Challenges in pcb design for space applications | gatema,” Jul 17 2024. [Online]. Available: <https://www.gatemapcb.com/challenges-in-pcb-design-for-space-applications/>

- [47] P. Directory, "Understanding pcb design techniques for space-qualified applications. - pcb directory," Apr 9 2025. [Online]. Available: <https://www.pcbdirectory.com/community/understanding-pcb-design-techniques-for-space-qualified-applications>
- [48] F. Circuits, "Pcbs for space: Ensuring reliability in space applications," Feb 21 2025. [Online]. Available: <https://www.fscircuits.com/pcbs-for-space/>
- [49] M. E. Tech, "Making a material difference in aerospace and defense electronics," December 1 2024. [Online]. Available: <https://www.mobilityengineeringtech.com/component/content/article/52137-making-a-material-difference-in-aerospace-defense-electronics>
- [50] Mascera-Tec, "Advantages and disadvantages of alumina ceramic," Website, n.d., accessed: 15 April 2025. [Online]. Available: <https://www.mascera-tec.com/news/advantages-and-disadvantages-of-alumina-ceramic>
- [51] V. Ceramics, "Advantages and disadvantages of alumina ceramics," Website, n.d., accessed: 15 April 2025. [Online]. Available: <https://vhandy.com/pros-and-cons-of-alumina-ceramics/>
- [52] C. D. Systems, "Designing reliable pcbs for space applications," 2025. [Online]. Available: <https://resourcespcb.cadence.com/blog/designing-reliable-pcbs-for-space-applications>
- [53] F. Lou, "How to select satellite components that can withstand space," Feb 5 2023. [Online]. Available: <https://revolutionized.com/satellite-components/>
- [54] R. PCB, "Introduction to ptfe teflon," LinkedIn post, September 14 2023. [Online]. Available: <https://www.linkedin.com/pulse/what-ptfe-teflon-material-pcb-rayming-techonloy>
- [55] S. international, "Design guidelines for military and aerospace pcbs," May 1 2022. [Online]. Available: <https://www.mobilityengineeringtech.com/component/content/article/45799-design-guidelines-for-military-and-aerospace-pcbs>
- [56] A. E. Ona-Olapo, T. D. Iluyomade, T. M. Olatunde, and O. P. Igbinenikaro, "Next-generation materials for space electronics: A conceptual review," *Open Access Research Journal of Engineering and Technology*, vol. 6, no. 2, pp. 51–62, Apr 30 2024. [Online]. Available: <https://oarjpublication.com/journals/oarjet/sites/default/files/OARJET-2024-0020.pdf>
- [57] J. C. Ince, M. Peerzada, L. D. Mathews, A. R. Pai, A. Al-Qatatsheh, S. Abbasi, Y. Yin, N. Hameed, A. R. Duffy, A. K. Lau, and N. V. Salim, "Overview of emerging hybrid and composite materials for space applications," *Advanced Composites and Hybrid Materials*, vol. 6, no. 4, -06-26 2023.

- [58] S. C. Systems, “Nasa inspection criteria for conformal coating,” Mar 10 2021. [Online]. Available: <https://scscoatings.com/newsroom/blog/nasa-inspection-criteria-for-conformal-coating/>
- [59] G. S. F. Centre, “Standard quality assurance requirements for printed circuit boards (gsfc-std-8001),” Nov 21 2019. [Online]. Available: <https://standards.nasa.gov/standard/GSFC/GSFC-STD-8001>
- [60] ——, “Standard quality assurance for use of water soluble flux(gsfc-std-8002),” Mar 09 2015. [Online]. Available: <https://standards.nasa.gov/sites/default/files/standards/GSFC/Baseline/0/GSFC-STD-8002.pdf>
- [61] C. D. Systems, “Designing reliable pcbs for space applications,” 2025. [Online]. Available: <https://resourcespcb.cadence.com/blog/designing-reliable-pcbss-for-space-applications>
- [62] ECSS, “Ecss-q-st-70-02c,” Nov 15 2008. [Online]. Available: <https://ecss.nl/standard/ecss-q-st-70-02c-thermal-vacuum-outgassing-test-for-the-screening-of-space-materials/>
- [63] ——, “List of published ecss standards (long) | european cooperation for space standardization,” 2019. [Online]. Available: <https://ecss.nl/list-of-published-ecss-standards-long/>
- [64] S. international, “Design guidelines for military and aerospace pcbs,” May 1 2022. [Online]. Available: <https://www.mobilityengineeringtech.com/component/content/article/45799-design-guidelines-for-military-and-aerospace-pcbs>
- [65] R. Shashikanth, “5 military grade pcb design rules | sierra circuits,” Jan 5 2021. [Online]. Available: <https://www.protoexpress.com/blog/military-grade-pcb-design-rules-considerations/>
- [66] C. D. Systems, “Pcb design for military and aerospace applications,” Oct 9 2024. [Online]. Available: <https://resourcespcb.cadence.com/blog/2024-pcb-design-for-military-aerospace-applications>

A. Appendix - DAQ

A.1. Arduino DAQ full code

A.1.1. Arduino C++ Code

```
1  /*
2   * Reads 4 analog inputs (0-5V) for recording_dur seconds
3   * Streams data to PC while recording
4   */
5   // change to true to print debug messages on Serial Monitor
6   // printing debug lines impacts transmission speed and messes csv
7   // do not leave on during measurements!
8   bool debug = false;
9
10  const int analogInputs[] = {A0, A1, A2, A3};
11
12
13  // set up the global variables
14  unsigned long start_time;
15  const unsigned long recording_dur = 5000; // 25 seconds in
milliseconds (ENSURE python code is greater than this)
16  unsigned long last_sample_time = 0;
17  const unsigned long min_samp_interval = 2; // Sample every 2ms (
adjust for stability)
18  bool recording = false;
19  int sample_count = 0;
20
21 void setup() {
22     // serial communication at 115200 bps
23     Serial.begin(115200);
24
25     // Set pins as input
26     for (int i = 0; i < 4; i++) {
27         pinMode(analogInputs[i], INPUT);
28     }
29
30     // Optimize ADC for faster sampling
31     // Set ADC prescaler to 16 (default is 128)
32     //
```

```

33     // Bit: 7:enable; 6: initiate a conversion 5:
34     //
35     ADCSRA = (ADCSRA & 0xF8) | 0x04;
36
37     // Wait for serial connection to establish
38     delay(1000);
39
40     // Send ready message
41     Serial.println("ARDUINO_DAQ_READY");
42 }
43
44 void loop() {
45     // Check if we received a command
46     if (Serial.available() > 0) {
47         String command = Serial.readStringUntil('\n');
48         command.trim();
49
50         if (command == "START") {
51             if(debug) Serial.println("received START command");
52
53             // Clear any remaining data in serial buffer
54             while (Serial.available()) {
55                 Serial.read();
56             }
57
58             // Reset sample counter
59             sample_count = 0;
60
61             // Send header once
62             Serial.println("Sample,Time(ms),A0(V),A1(V),A2(V),A3(V)");
63
64             // Start recording
65             recording = true;
66             start_time = millis();
67             last_sample_time = start_time;
68
69             // Send confirmation
70             Serial.println("RECORDING_STARTED");
71         }
72     }
73
74     // If we're recording, collect and send data immediately
75     if (recording) {
76         if(debug) Serial.println("Recording!");
77         unsigned long currentTime = millis();
78         unsigned long elapsed_time = currentTime - start_time;
79

```

```

80     // Check if we're still within the recording period
81     if (elapsed_time <= recording_dur) {
82         if(debug) Serial.println("elapsed time << duration");
83         // Only sample at the specified interval
84         if (currentTime - last_sample_time >= min_samp_interval) {
85             last_sample_time = currentTime;
86
87             // Increment sample counter
88             sample_count++;
89
90             // Start building the output string
91             String data_string = String(sample_count) + "," + String(
92             elapsed_time);
93
94             // Multiplex through the four inputs sequentially
95             for (int i = 0; i < 4; i++) {
96                 if(debug) Serial.println("reading input: " + String(i));
97                 int raw_value = analogRead(analogInputs[i]);
98                 float voltage = raw_value * (5.0 / 1023.0);
99                 data_string += "," + String(voltage, 3);
100            }
101
102            // Send the complete data string at once
103            Serial.println(data_string);
104        }
105    } else {
106        // End of recording
107        recording = false;
108
109        // Send notification that recording is complete
110        Serial.println("RECORDING_COMPLETE");
111        Serial.print("SAMPLES_COLLECTED:");
112        Serial.println(sample_count);
113        Serial.println("END_OF_DATA");
114    }
115}
116

```

Listing A.1: C++ Code on Arduino

A.1.2. PC-side Python Serial Receive Script

```

1 import serial
2 import time
3 import matplotlib.pyplot as plt
4 import pandas as pd

```

```

5      import os
6      import numpy as np
7      from scipy import signal
8
9      def apply_lowpass_filter(data, fs):
10         """Apply a 4-pole low-pass Butterworth filter with 5Hz cutoff"""
11         cutoff_freq = 2.0
12         filter_order = 4
13
14         nyquist = 0.5 * fs
15         normal_cutoff = cutoff_freq / nyquist
16         # analog=False implies bilinear Transformation
17         b, a = signal.butter(filter_order, normal_cutoff, btype='low',
18         analog=False)
19         filtered_data = signal.filtfilt(b, a, data)
20
21     return filtered_data
22
23     # Load data from CSV, apply a 4-pole low-pass filter, and save the
24     # filtered data
25     def filter_and_save_data(filename):
26
27         # Read the CSV data to pandas DataFrame
28         # It knows what are the column names
29         df = pd.read_csv(filename)
30
31         # Clean the dataframe - convert all columns to numeric
32         for col in df.columns:                      # v - write NaN where it
33             can't convert to number (in teh data, not column names)
34             df[col] = pd.to_numeric(df[col], errors='coerce')
35
36         # remove rows with NaN (not a number)
37         df = df.dropna()
38
39         # Samples not at exact distance from each other
40         # Calculate the sampling frequency (median of differences)
41         # numpy.diff to get the difference between samples
42         time_diffs = np.diff(df['Time(ms)'])
43         # numpy.median to get the median
44         median_time_diff = np.median(time_diffs) # in milliseconds
45         fs = 1000.0 / median_time_diff # Convert to Hz
46
47         # Filter each analog channel:
48         # ID column head for each channel
49         analog_channels = ['A0(V)', 'A1(V)', 'A2(V)', 'A3(V)']
50         # take each channel one at a time
51         for channel in analog_channels:

```

```

49         # if name matches a column name
50         if channel in df.columns:
51             # add a new column _filtered , and send the array
52             # containing all the raw values to
53             # have them filtered , and save them in the new _filtered
54             # column
55             df[f"{channel}_filtered"] = apply_lowpass_filter(df[
56             channel].values , fs)
57
58             # Save the pandas Dataframe with filtered columns to a new CSV
59             file
60             filtered_filename = f"{os.path.splitext(filename)[0]}_filtered.
61             csv"
62             df.to_csv(filtered_filename , index=False)
63
64             return filtered_filename
65
66             #
67             # Plot the DAQ data with original and filtered signals overlapped
68             #
69             def plot_data(filename):
70
71                 # Read the CSV data with pandas
72                 df = pd.read_csv(filename)
73
74                 # Initialize an empty list to store our analog channel names
75                 analog_channels = []
76
77                 # Look through all column names in the DataFrame
78                 for col in df.columns:
79                     # the column name starts with 'A'
80                     if col.startswith('A'):
81                         # the column name ends with '(V)'
82                         if col.endswith('(V)'):
83                             # the column name does NOT contain '_filtered'
84                             if '_filtered' not in col:
85                                 # add this column name to our list
86                                 analog_channels.append(col)
87
88                 # Create color cycle for different channels
89                 colors = ['blue' , 'green' , 'red' , 'purple']
90
91                 # Create a single plot with all channels overlapping
92                 plt.figure(figsize=(14, 8))
93
94                 # Plot original data (semi-transparent)
95                 for i, channel in enumerate(analog_channels):

```

```

91         color = colors[i % len(colors)]
92         plt.plot(df['Time(ms)'], df[channel], label=f'{channel}
93 Original',
94             linewidth=1.5, alpha=0.4, color=color, linestyle='--'
95 )
96
97     # Plot filtered data (solid lines)
98     for i, channel in enumerate(analog_channels):
99         filtered_channel = f"{channel}_filtered"
100        if filtered_channel in df.columns:
101            color = colors[i % len(colors)]
102            plt.plot(df['Time(ms)'], df[filtered_channel], label=f'{channel} Filtered',
103                 linewidth=2.5, color=color, linestyle='--')
104
105    # Set the y-axis range from 0 to 5V
106    plt.ylim(0, 5)
107
108    # Add labels and title
109    plt.xlabel('Time (ms)')
110    plt.ylabel('Voltage (V)')
111    plt.title('Arduino DAQ - 4-Channel Readings with 4-Pole 5Hz Low-
112 Pass Filter')
113    plt.legend()
114    plt.grid(True)
115
116    # Add data summary
117    duration = df['Time(ms)'].max() - df['Time(ms)'].min()
118    sample_count = len(df)
119    sample_rate = sample_count/(duration/1000) if duration > 0 else
120    0
121
122    info_text = f"Data summary:\n" \
123        f"Duration: {duration:.1f} ms\n" \
124        f"Samples: {sample_count}\n" \
125        f"Sample rate: {sample_rate:.1f} Hz\n" \
126        f"Filter: 4-pole Butterworth, 5Hz cutoff"
127
128    plt.figtext(0.02, 0.02, info_text, fontsize=10,
129                bbox=dict(facecolor='white', alpha=0.8))
130
131    # Save the plot
132    plot_filename = f"{os.path.splitext(filename)[0]}_plot.png"
133    plt.savefig(plot_filename, dpi=300, bbox_inches='tight')
134
135    # Show the plot
136    plt.tight_layout()

```

```

133     plt.show()
134
135     def main():
136
137         # ASk if to plot or measure
138         what_to_do = int(input("Record new measurement (1) or plot
existing (2): "))
139
140         # User selected to plot old csv
141         if(what_to_do == 2):
142             plot_data(input("Insert the name of the csv: "))
143
144
145         # User selected to record new measurement
146         elif(what_to_do == 1):
147             # Use a default port (COM3 for Windows, modify as needed)
148             default_port = "COM3" # Change to match your system
149
150             print(f"Using port: {default_port}")
151
152             # Configure serial port
153             try:
154                 ser = serial.Serial(default_port, 115200, timeout=2)
155                 print("Connected to Arduino!")
156             except serial.SerialException:
157                 print(f"Error: Could not open port {default_port}")
158                 print("Please modify the default_port variable in the
script.")
159             return
160
161             time.sleep(2) # Wait for Arduino to reset
162
163             # Flush buffers
164             ser.reset_input_buffer()
165             ser.reset_output_buffer()
166
167             # Wait for Arduino ready
168             print("Waiting for Arduino to be ready...")
169             ready = False
170             timeout = time.time() + 10 # don't wait too long
171
172             while not ready and time.time() < timeout:
173                 line = ser.readline().decode('utf-8', errors='ignore').
strip()
174                 if line == "ARDUINO_DAQ_READY":
175                     ready = True
176                     print("Arduino is ready!")

```

```

177
178     if not ready:
179         print("Timed out waiting for Arduino. Make sure it's
properly connected.")
180         ser.close()
181         return
182
183     # Create a filename for this recording session
184     filename = f"arduino_daq_data_{time.strftime('%Y%m%d_%H%M%S
')}.csv"
185
186     print(f"Starting data recording to {filename}...")
187     print("Press Ctrl+C to stop if needed.")
188
189     with open(filename, 'w', newline='') as file:
190         # Send start command
191         ser.write(b"START\n")
192
193         recording = True
194         data_lines = 0
195
196         # Start time for timeout
197         start_time = time.time()
198         timeout_duration = 15 # seconds
199
200         while recording and (time.time() - start_time) <
timeout_duration:
201             if ser.in_waiting:
202                 line = ser.readline().decode('utf-8', errors='
ignore').strip()
203
204                 if "RECORDING_COMPLETE" in line:
205                     recording = False
206                     print("Recording complete!")
207                 elif "END_OF_DATA" in line:
208                     pass
209                 elif line:
210                     # Write the line to the file
211                     file.write(line + '\n')
212                     data_lines += 1
213
214                     # Show progress occasionally
215                     if data_lines % 100 == 0:
216                         print(f"Recorded {data_lines} data
points...")
217
218             # Close the serial port

```

```

219         if ser.is_open:
220             ser.close()
221             print("Serial port closed.")
222
223             print(f"Recorded {data_lines} lines of data.")
224
225             # Process the data
226             print("Applying filters to data...")
227             filtered_filename = filter_and_save_data(filename)
228             print(f"Filtered data saved to {filtered_filename}")
229
230             print("Generating plot...")
231             plot_data(filtered_filename)
232             print("Done!")
233
234     else:
235         print("Wrong Choice. Goodbye!")
236         exit()
237
238 if __name__ == "__main__":
239     try:
240         main()
241     except KeyboardInterrupt:
242         print("\nProgram terminated by user.")

```

Listing A.2: Python Serial Receive Script

A.2. Derivation of TIA Voltage Out

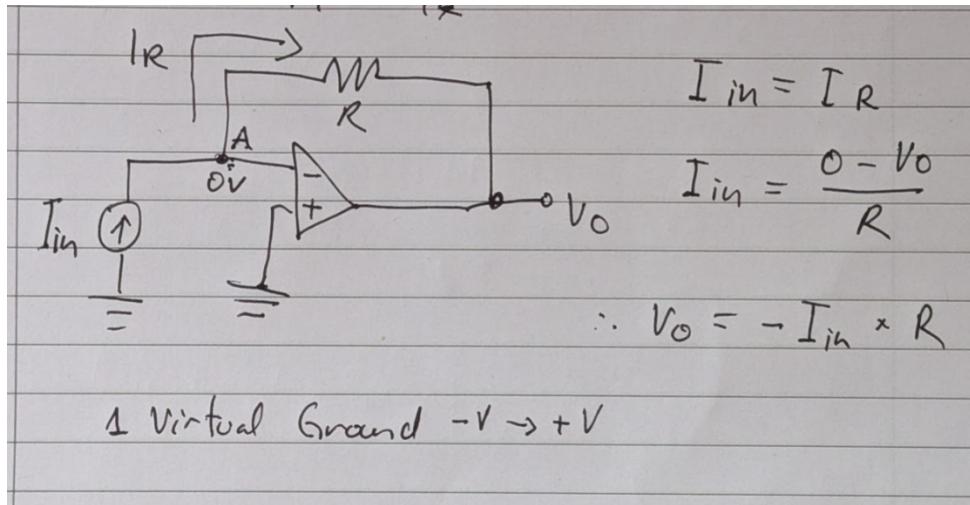


Figure A.1.: Solution Deriving TIA Vout

B. Appendix - Software Model Code

B.1. Section 1 Title

B.1.1. subsectiontitle

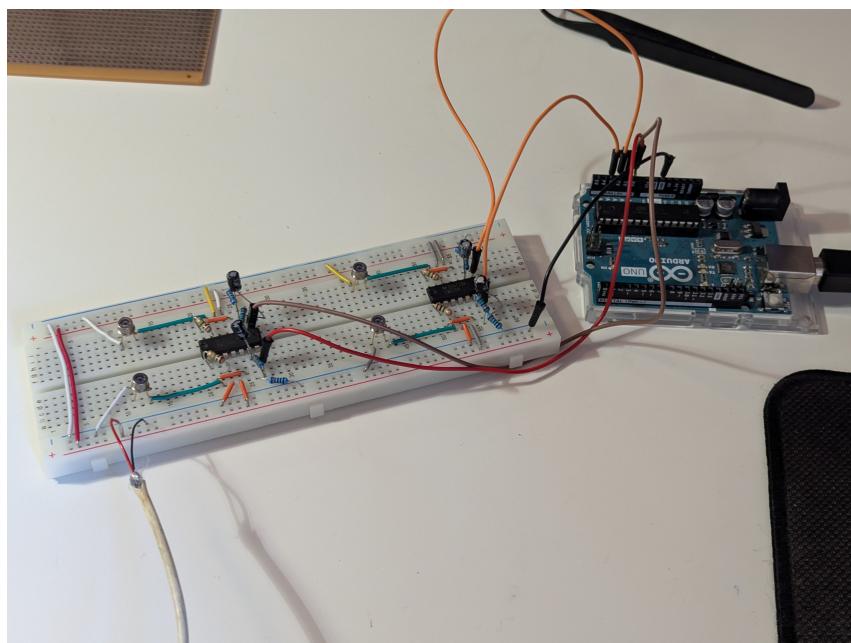
B.2. Section 2 Title

B.2.1. subsectiontitle

C. Appendix - Photos of Lab Work

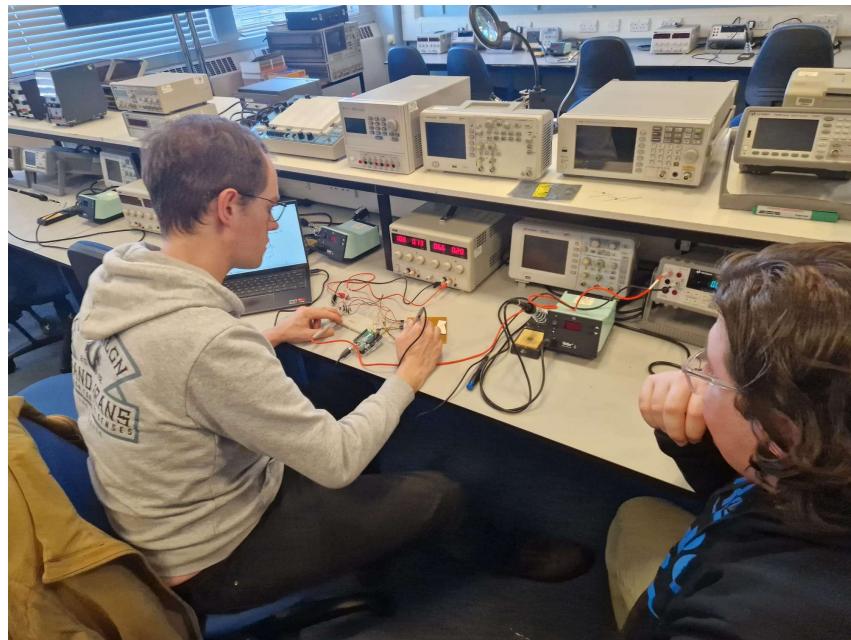
C.1. Prototype Images

C.1.1. BreadBoard Prototype

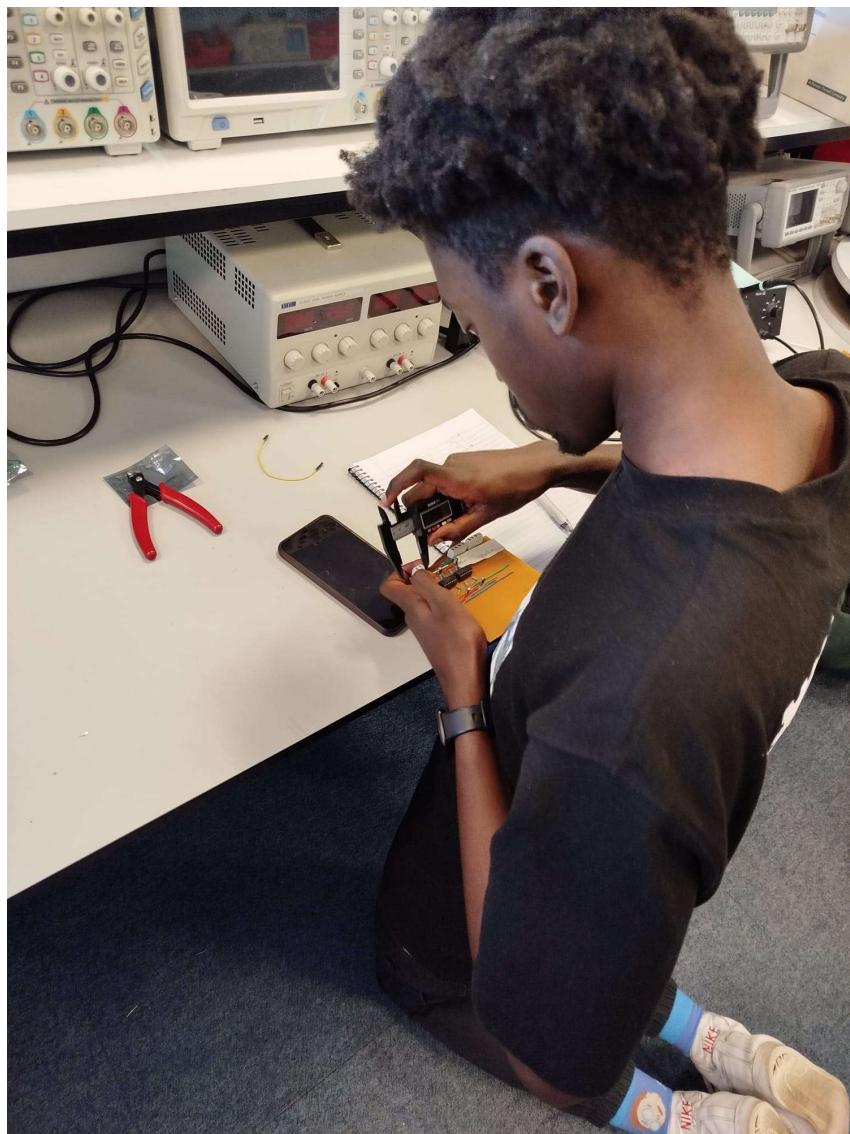


Initial BreadBoard Prototype of the Photodiode Circuit

C.1.2. Building the Prototype

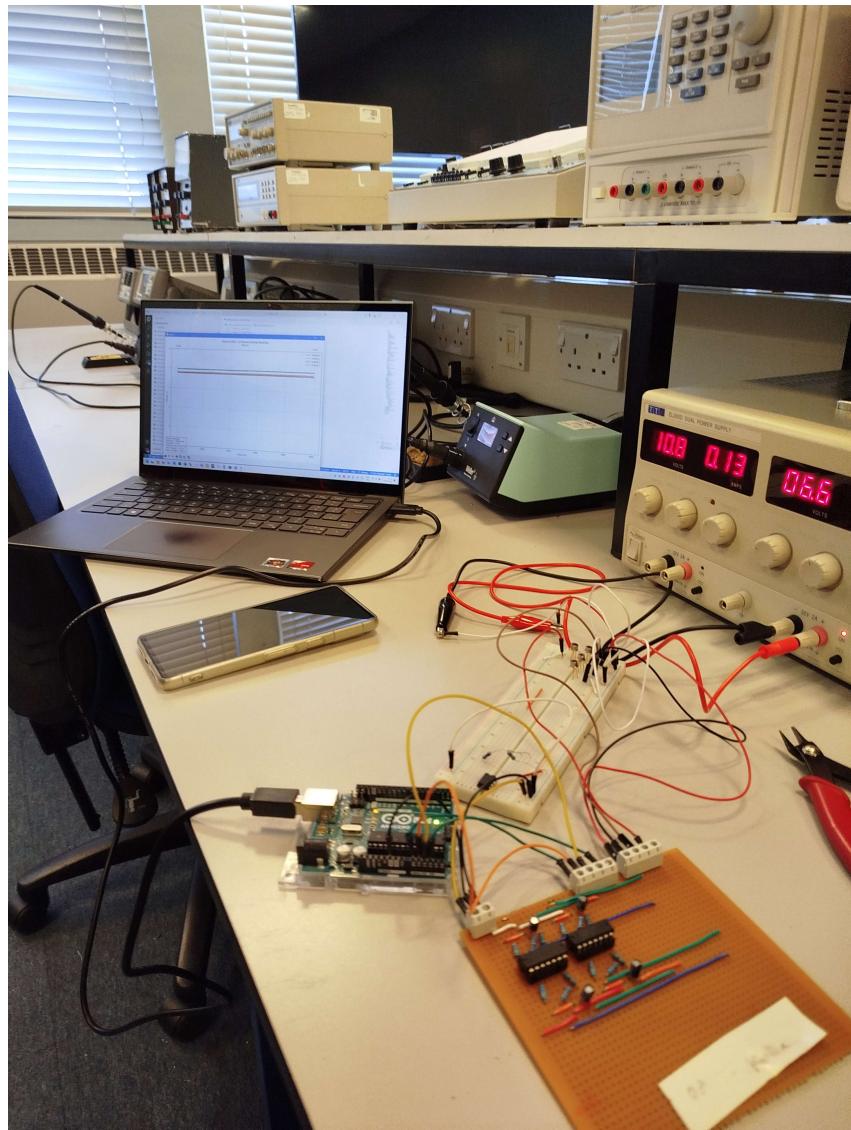


Lab Work for the Prototype (session 1)



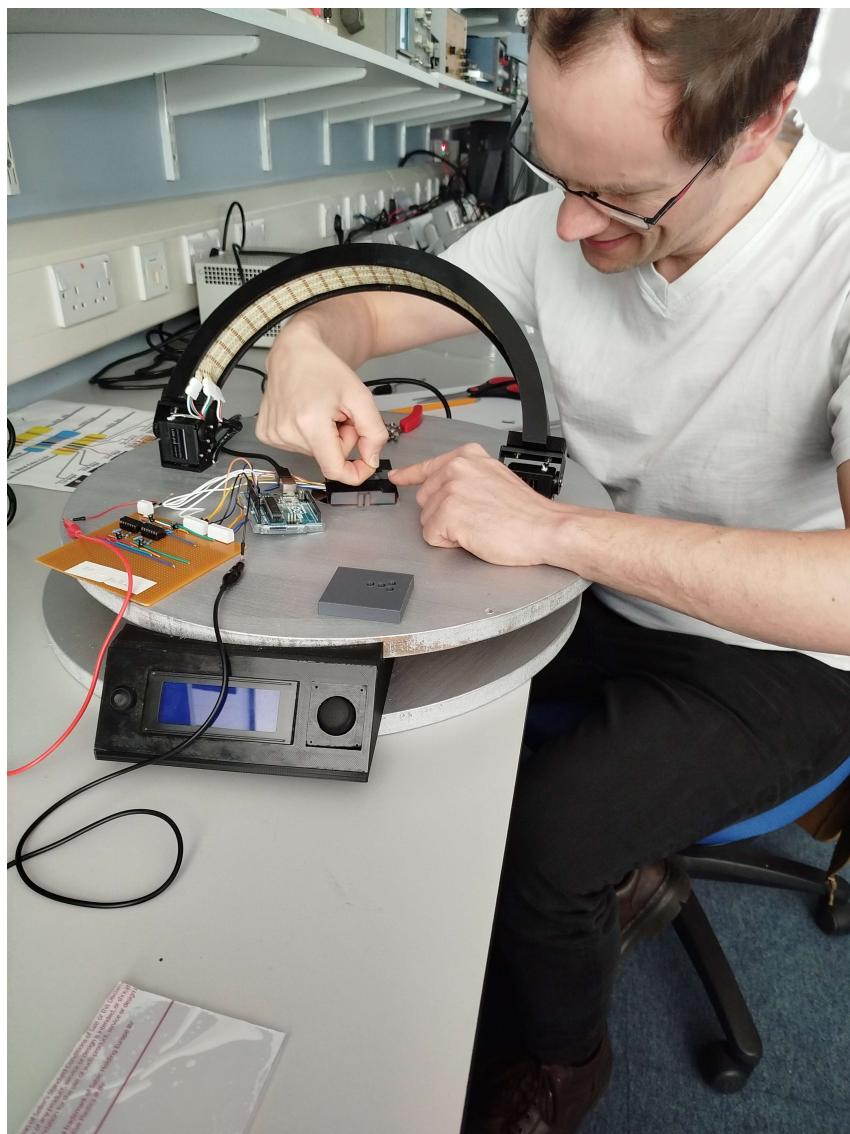
lab Work for the Prototype (session 2)

C.1.3. Prototype Testing



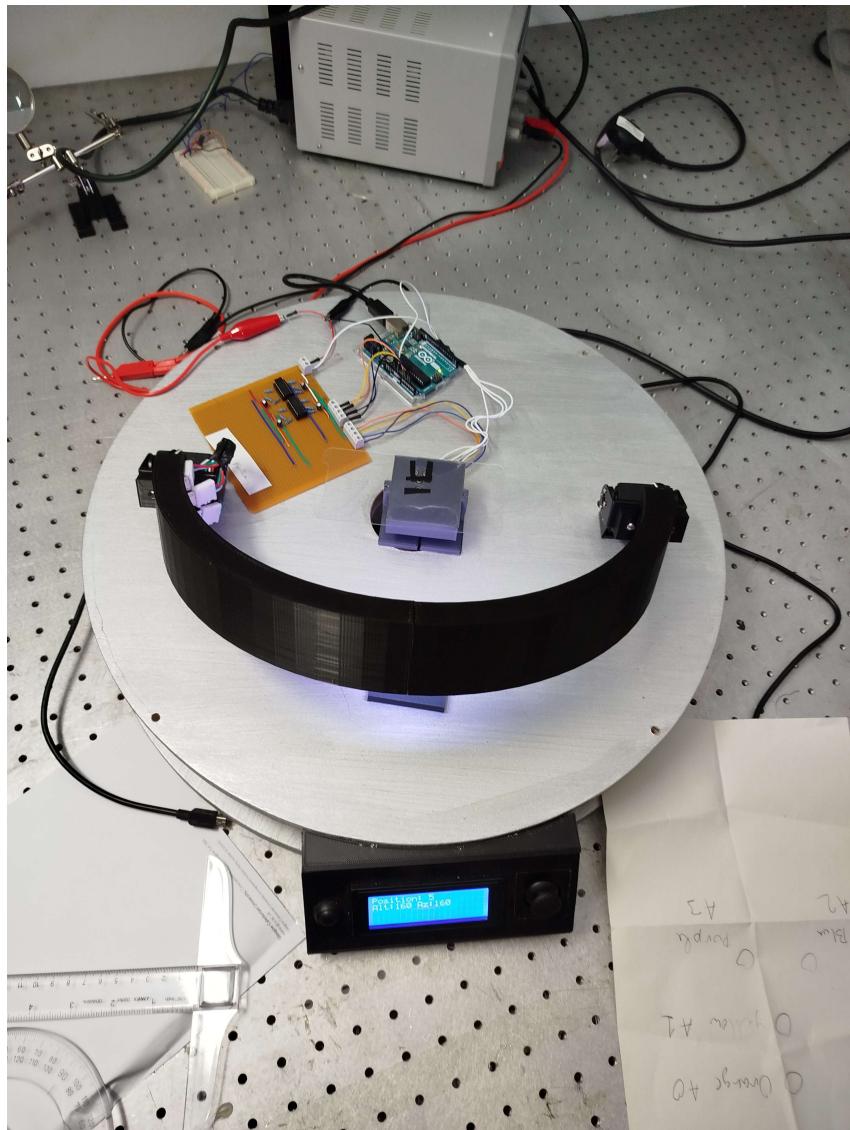
First Lab Test of the Prototype

C.1.4. RED testbench

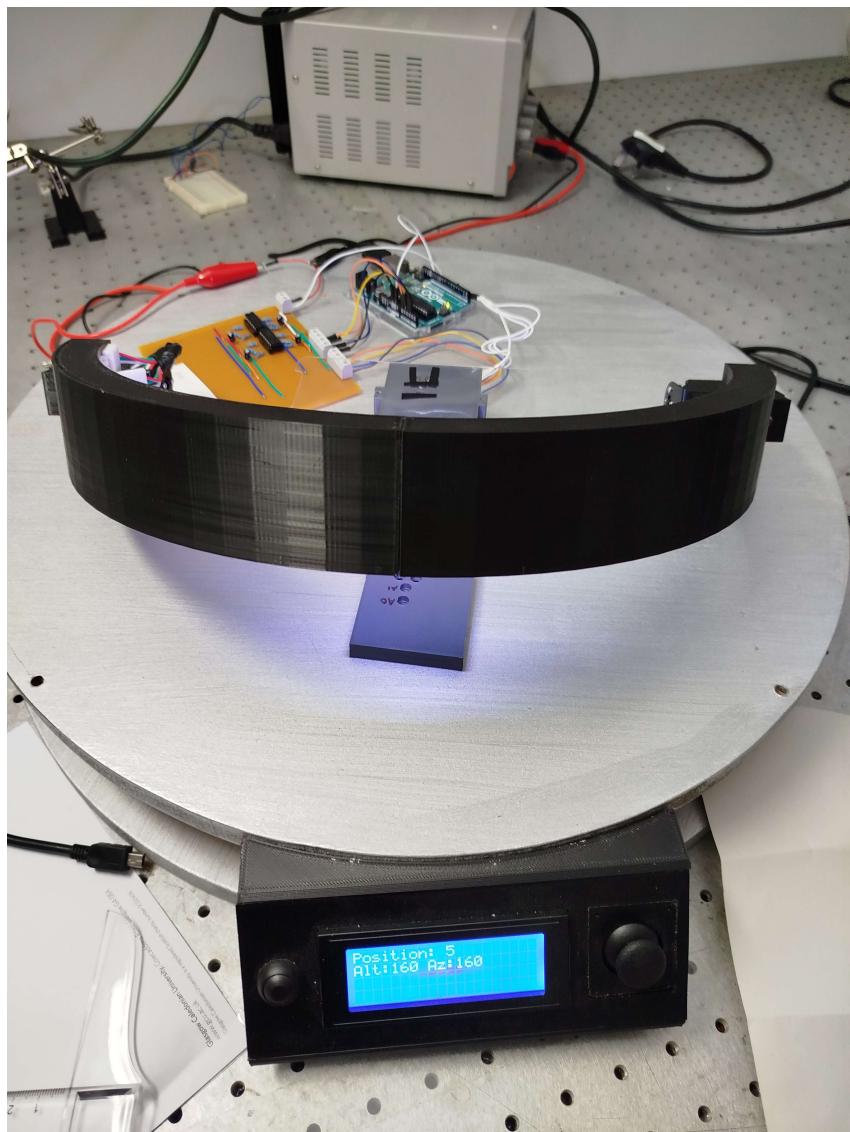


RED Testbench 1

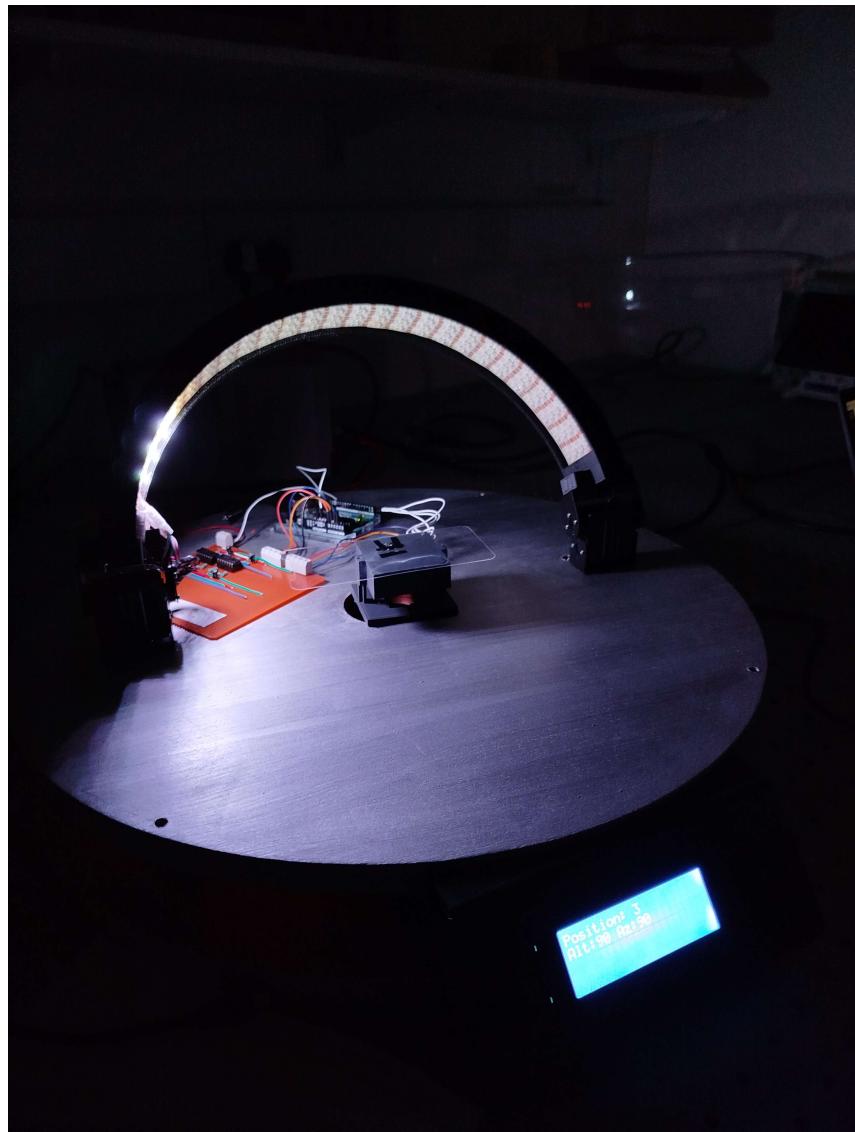
C.1.5. Solar Lab Prototype Testing



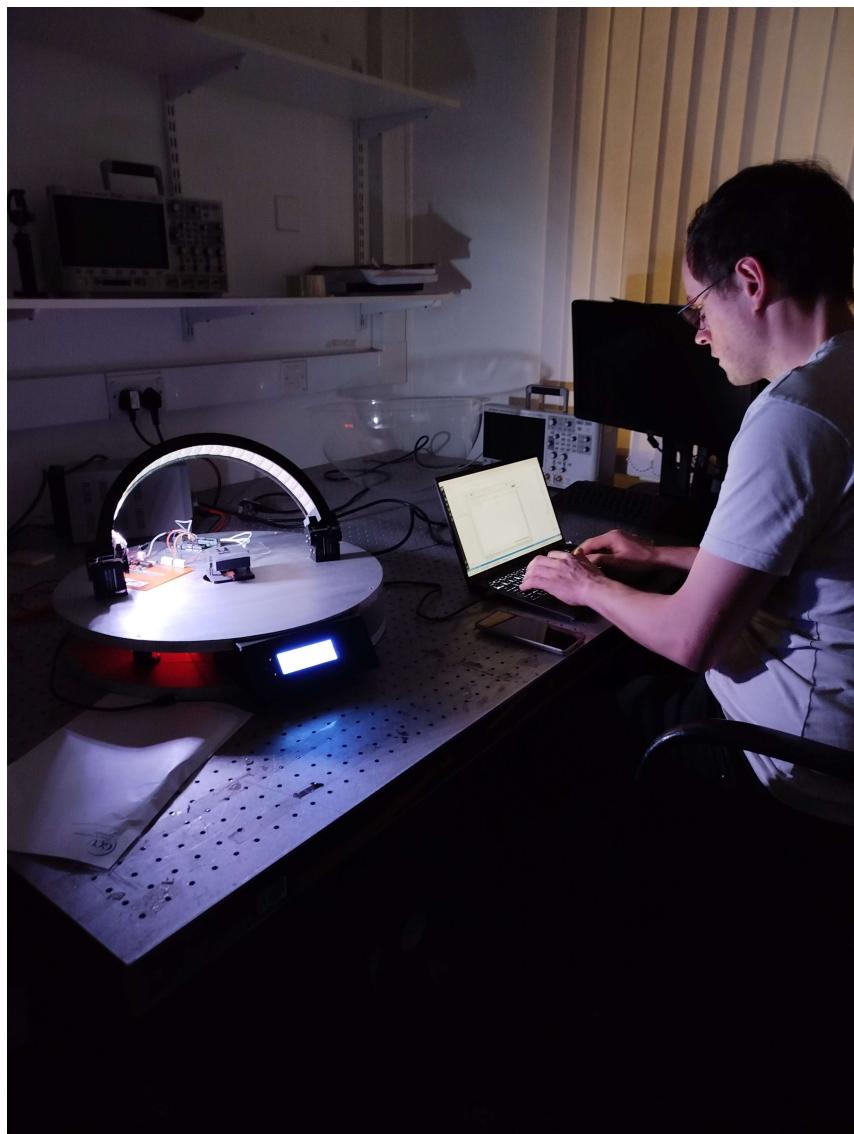
Solar Lab Test of the Prototype 1



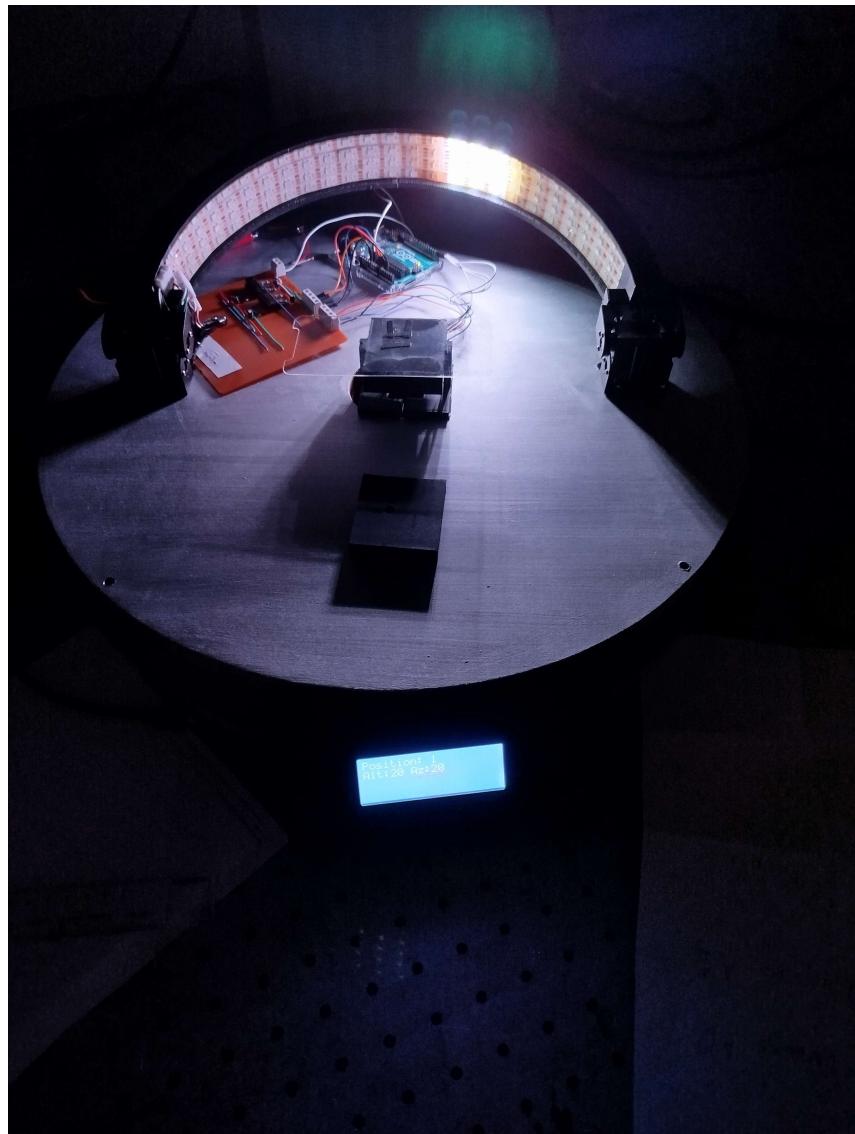
Solar Lab Test of the Prototype 2



Solar Lab Test of the Prototype 3



Solar Lab Test of the Prototype 4



Solar Lab Test of the Prototype 5

D. Appendix - Material Analysis

D.1. Material Analysis - ANSYS

D.1.1. PCB Comparison

Comparison - MaterialUniverse			
All Data	Project Data	Ranges	Averages
# Values	% Change	Highlight % Change > 10	Apply
Alumina (96%)	Polyimide (PI), PCB laminate	PTFE (unfilled)	
General information			
Included in Materials Data for Simulation		✓	✓
Materials Data for Simulation name	PCB laminate, Polyamide (PI)	Plastic, PTFE	
Composition overview			
Form	Bulk material	Other	Bulk material
Material family	Ceramic (technical)	Composite (polymer matrix)	Plastic (thermoplastic, semi-crystalline)
Base material	Oxide	PI (Polyimide, aromatic)	PTFE (Polytetrafluoroethylene)
Polymer code	PI	PTFE	
Composition detail (metals, ceramics and glasses)			
Al2O3 (alumina) (%)	96	0	0
Other (%)	4	0	0
Composition detail (polymers and natural materials)			
Polymer (%)	0	100	100
Price			
Price (GBP/kg)	11.2 - 16.8	60.8 - 72.1	9.16 - 10.3
Price per unit volume (GBP/m^3)	41400 - 62700	103000 - 123000	19600 - 22600
Physical properties			
Density (kg/m^3)	3690 - 3730	1700	2140 - 2200
Porosity (closed) (%)	0		
Porosity (open) (%)	0		
Mechanical properties			
Young's modulus (GPa)	268 - 282	24.9	0.4 - 0.552
Young's modulus with temperature (GPa) #			0.42
Specific stiffness (MN.m/kg)	72.3 - 76	14.6	0.184 - 0.255
Yield strength (elastic limit) (MPa)	210 - 231	249	19.7 - 21.7
Tensile strength (MPa)	210 - 231	249	20.7 - 34.5
Specific strength (kN.m/kg)	56.6 - 62.3	146	9.06 - 10
Elongation (% strain)	0.07 - 0.09	75 - 85	200 - 400
Compressive modulus (GPa)		24.3 - 25.5	0.402 - 0.423
Compressive strength (MPa)	2100 - 2310	231 - 269	11.2 - 12.3
Flexural modulus (GPa)	268 - 282	24.3 - 25.5	0.537 - 0.564

Edupack(PCB) part 1

D.1.2. Aluminium Comparison

Comparison - MaterialUniverse				
	All Data	Project Data	Ranges	Averages
	# Values	% Change	Highlight % Change >	10 Apply
Flexural modulus (GPa)	268 - 282	24.3 - 25.5	0.537 - 0.564	
Flexural strength (modulus of rupture) (MPa)	345 - 370	334 - 390	29 - 48.3	
Shear modulus (GPa)	108 - 114	10.4 - 10.9	0.138 - 0.19	
Bulk modulus (GPa)	172 - 181	12.3 - 12.9	1.53 - 1.6	
Poisson's ratio	0.22 - 0.26	0.17	0.44 - 0.46	
Shape factor	15	8	4	
Hardness - Vickers (HV)	1750 - 1950	26 - 27	6 - 7	
Hardness - Shore D			55 - 60	
Elastic stored energy (springs) (kJ/m^3)	80.1 - 97.1	1250	379 - 546	
Fatigue strength at 10^7 cycles (MPa)	178 - 208	92.2 - 108	5.75 - 7	
Impact & fracture properties				
Fracture toughness (MPa.m^0.5)	3 - 3.6	5.6 - 6.78	1.32 - 1.8	
Toughness (G) (kJ/m^2)	0.0329 - 0.0468	1.27 - 1.83	3.65 - 6.97	
Impact strength, notched 23 °C (kJ/m^2)		8	15 - 17	
Thermal properties				
Melting point (°C)	2000 - 2100		315 - 339	
Glass temperature (°C)		360 - 410	117 - 130	
Heat deflection temperature 0.45MPa (°C)			71 - 121	
Heat deflection temperature 1.8MPa (°C)		360	31 - 62	
Maximum service temperature (°C)	1580 - 1660	250 - 320	250 - 271	
Minimum service temperature (°C)	-273	-270	-268 - -200	
Thermal conductivity (W/m.°C)	25 - 25.5	0.3	0.242 - 0.261	
Thermal conductivity with temperature (W/m.°C) #			0.273	
Specific heat capacity (J/kg.°C)	687 - 715	1060	970 - 1090	
Specific heat capacity with temperature (J/kg.°C) #			967	
Thermal expansion coefficient (μstrain/°C)	6.4 - 7.8	14	120 - 170	
Thermal shock resistance (°C)	101 - 127	714	244 - 389	
Thermal distortion resistance (MW/m)	3.24 - 3.95	0.0214	0.00147 - 0.0021	
Latent heat of fusion (kJ/kg)	920 - 1040		600	
Electrical properties				
Electrical resistivity (μohm.cm)	1e20 - 1e21	8.2e19	3.3e23 - 3e24	
Electrical conductivity (%IACS)	1.72e-19 - 1.72e-18	2.1e-18	5.75e-23 - 5.22e-22	
Dielectric constant (relative permittivity)	9 - 9.3	4.25	2 - 2.2	
Dissipation factor (dielectric loss tangent)	0.00025 - 0.00035	0.014	0.00019 - 0.00021	
Dielectric strength (dielectric breakdown) (MV/m)	10 - 26	22	18.2 - 19.7	
Comparative tracking index (V)				

Edupack(PCB) part 2

Comparison - MaterialUniverse			
<input type="checkbox"/> All Data <input type="checkbox"/> Project Data <input type="checkbox"/> Ranges <input type="checkbox"/> Averages <input type="checkbox"/> # Values <input type="checkbox"/> % Change <input type="checkbox"/> Highlight % Change > 10 <input type="checkbox"/> Apply			
<small>Comparative tracking index (v)</small>			
<input type="checkbox"/> Magnetic properties	Non-magnetic	Non-magnetic	Non-magnetic
<input type="checkbox"/> Optical, aesthetic and acoustic properties			
Refractive index	1.75 - 1.77	1.31 - 1.36	
Transparency	Opaque	Translucent	Translucent
Acoustic velocity (m/s)	8500 - 8720	3830	428 - 506
Mechanical loss coefficient (tan delta)	0.000015 - 0.00005	0.00461 - 0.0061	0.0725 - 0.1
<input type="checkbox"/> Healthcare & food			
Food contact	Yes	No	Yes
<input type="checkbox"/> Restricted substances risk indicators			
RoHS 2 (EU) compliant grades?	✓	✓	
REACH Candidate List indicator (0-1, 1 = high risk)	0	0	0.01
SIN List indicator (0-1, 1 = high risk)	0	0.01	0.01
<input type="checkbox"/> Critical materials risk			
Contains > 5wt% critical elements?	Yes	No	
<input type="checkbox"/> Usage in power systems			
Nuclear power?		✓	
<input type="checkbox"/> Nuclear reactor systems			
Thermonuclear (fusion) reactor (ITER)		✓	
<input type="checkbox"/> Absorption & permeability			
Water absorption @ 24 hrs (%)	0.2 - 2.9	0.005 - 0.01	
Water absorption @ sat (%)	0.808 - 11.7		
Humidity absorption @ sat (%)	2.8 - 3		
Water vapor transmission (g.mm/m ² .day)		0.166 - 0.184	
Permeability (O ₂) (cm ³ .mm/m ² .day.atm)		217 - 363	
Permeability (CO ₂) (cm ³ .mm/m ² .day.atm)		453 - 842	
Permeability (N ₂) (cm ³ .mm/m ² .day.atm)		81.3 - 169	
<input type="checkbox"/> Processing properties			
Polymer injection molding	Limited use	Unsuitable	
Polymer extrusion	Limited use	Unsuitable	
Polymer thermoforming	Unsuitable	Unsuitable	
Linear mold shrinkage (%)		3 - 6	
Molding pressure range (MPa)		13.8 - 34.4	

Edupack(PCB) part 3

Comparison - MaterialUniverse			
<input type="checkbox"/> All Data <input type="checkbox"/> Project Data <input type="checkbox"/> Ranges <input type="checkbox"/> Averages <input type="checkbox"/> # Values <input type="checkbox"/> % Change <input type="checkbox"/> Highlight % Change > 10 <input type="checkbox"/> Apply			
Polymer molding CO2 (kg/kg)			1.66 - 1.83
Polymer molding water (l/kg)			13.4 - 20.1
Coarse machining energy (per unit wt removed) (MJ/kg)	1.87 - 2.06	0.526 - 0.582	
Coarse machining CO2 (per unit wt removed) (kg/kg)	0.14 - 0.155	0.0395 - 0.0436	
Fine machining energy (per unit wt removed) (MJ/kg)	14.4 - 15.9	0.989 - 1.09	
Fine machining CO2 (per unit wt removed) (kg/kg)	1.08 - 1.19	0.0742 - 0.082	
Grinding energy (per unit wt removed) (MJ/kg)	113 - 125	3.26 - 3.6	1.5 - 1.66
Grinding CO2 (per unit wt removed) (kg/kg)	8.5 - 9.39	0.244 - 0.27	0.113 - 0.125
<input type="checkbox"/> Recycling and end of life			
Recycle	✗	✗	✓
Embodied energy, recycling (MJ/kg)			96 - 106
CO2 footprint, recycling (kg/kg)			5.24 - 5.78
Recycle fraction in current supply (%)			0.672 - 0.742
Downcycle	✓	✓	✓
Combust for energy recovery	✗	✗	✓
Heat of combustion (net) (MJ/kg)			4.69 - 4.92
Combustion CO2 (kg/kg)			0.859 - 0.903
Landfill	✓	✓	✓
Biodegrade	✗	✗	✗
<input type="checkbox"/> Geo-economic data for principal component			
Principal component	Alumina		
Annual world production, principal component (tonne/yr)	1.15e8		
<input type="checkbox"/> Links			
Legislation and Regulations	0	0	1
Low Carbon Energy Systems	0	0	1
Nuclear power systems	0	0	1
ProcessUniverse	36	39	55
Producers	13	0	10
Reference	28	0	15
Shape	20	0	24
Full Datasheet	view	view	view
<input type="checkbox"/> #Functional Parameters			
Temperature (°C)	23		

Edupack(PCB) part 4

Comparison - MaterialUniverse			
<input type="checkbox"/> All Data <input type="checkbox"/> Project Data <input type="checkbox"/> Ranges <input type="checkbox"/> Averages <input type="checkbox"/> # Values <input type="checkbox"/> % Change <input type="checkbox"/> Highlight % Change > 10 <input type="checkbox"/> Apply			
Molding pressure range (MPa)			13.8 - 34.4
Durability			
Water (fresh)	Excellent	Acceptable	Excellent
Water (salt)	Excellent	Unacceptable	Excellent
Weak acids	Excellent	Limited use	Excellent
Strong acids	Excellent	Unacceptable	Excellent
Weak alkalies	Excellent	Unacceptable	Excellent
Strong alkalies	Excellent	Unacceptable	Excellent
Organic solvents	Excellent	Acceptable	Excellent
Oils and fuels		Acceptable	
Oxidation at 500C	Excellent	Unacceptable	Unacceptable
UV radiation (sunlight)	Excellent	Poor	Good
Halogens	Acceptable		
Metals	Acceptable		
Flammability	Non-flammable	Non-flammable	Non-flammable
Oxygen index (%)	100	53	94 - 96
Primary production energy, CO2 and water			
Embodied energy, primary production (virgin grade) (MJ/kg)	49.5 - 54.7	351 - 387	283 - 312
Embodied energy, primary production (typical grade) (MJ/kg)	49.5 - 54.7	351 - 387	281 - 310
CO2 footprint, primary production (virgin grade) (kg/kg)	2.67 - 2.95	14.5 - 16	15.4 - 17
CO2 footprint, primary production (typical grade) (kg/kg)	2.67 - 2.95	14.5 - 16	15.3 - 16.9
Water usage (l/kg)	53.4 - 59.1		434 - 480
Processing energy, CO2 footprint & water			
Polymer extrusion energy (MJ/kg)		8.03 - 8.85	
Polymer extrusion CO2 (kg/kg)		0.642 - 0.708	
Polymer extrusion water (l/kg)		5.74 - 8.61	
Polymer molding energy (MJ/kg)		20.7 - 22.8	
Polymer molding CO2 (kg/kg)		1.66 - 1.83	
Polymer molding water (l/kg)		13.4 - 20.1	
Coarse machining energy (per unit wt removed) (MJ/kg)	1.87 - 2.06	0.526 - 0.582	
Coarse machining CO2 (per unit wt removed) (kg/kg)	0.14 - 0.155	0.0395 - 0.0436	
Fine machining energy (per unit wt removed) (MJ/kg)	14.4 - 15.9	0.989 - 1.09	

Edupack(PCB) part 5

Comparison - MaterialUniverse		
	All Data	Project Data
	Ranges	Averages
	# Values	% Change
	Highlight % Change > 10	Apply
	Aluminum, 6061, T651	Aluminum, 7075, T6
General information		
Condition	T651 (Solution heat-treated and artificially aged)	T6 (Solution heat-treated and artificially aged)
UNS number	A96061	A97075
EN name	EN AW-6061 (EN AW-Al Mg1SiCu)	EN AW-7075 (EN AW-Al Zn5,5MgCu)
EN number	3.3211	3.4365
Included in Materials Data for Simulation	✓	✓
Materials Data for Simulation name	Aluminum alloy, wrought, 6061, T651	Aluminum alloy, wrought, 7075, T6
Composition overview		
Material family	Metal (non-ferrous)	Metal (non-ferrous)
Base material	Al (Aluminum)	Al (Aluminum)
Composition detail (metals, ceramics and glasses)		
Al (aluminum) (%)	95.8 - 98.6	87.2 - 91.4
Cr (chromium) (%)	0.04 - 0.35	0.18 - 0.28
Cu (copper) (%)	0.15 - 0.4	1.2 - 2
Fe (iron) (%)	0 - 0.7	0 - 0.5
Mg (magnesium) (%)	0.8 - 1.2	2.1 - 2.9
Mn (manganese) (%)	0 - 0.15	0 - 0.3
Si (silicon) (%)	0.4 - 0.8	0 - 0.4
Ti (titanium) (%)	0 - 0.15	0 - 0.2
Zn (zinc) (%)	0 - 0.25	5.1 - 6.1
Other (%)	0 - 0.15	0 - 0.15
Price		
Price (GBP/kg)	1.52 - 1.75	3.05 - 3.5
Price per unit volume (GBP/m^3)	4110 - 4780	8410 - 9910
Physical properties		
Density (kg/m^3)	2690 - 2730	2770 - 2830
Mechanical properties		
Young's modulus (GPa)	66.6 - 70	69 - 76
Young's modulus with temperature (GPa) #	69	69.6 - 76.6
Specific stiffness (MN.m/kg)	24.5 - 25.8	24.6 - 27.2
Yield strength (elastic limit) (MPa)	241 - 281	460 - 530
Yield strength with temperature (MPa) #	279	499

Edupack(Aluminium) part 1

Comparison - MaterialUniverse			
	All Data	Project Data	Ranges Averages # Values % Change Highlight % Change > 10
Yield strength with temperature (MPa) #	279	499	
Tensile strength (MPa)	274 - 320	530 - 580	
Tensile strength with temperature (MPa) #		557	
Specific strength (kN.m/kg)	88.8 - 104	164 - 189	
Elongation (% strain)	10 - 14.4	2 - 10	
Tangent modulus (MPa)	563	1190	
Compressive modulus (GPa)	67.9 - 71.3		
Compressive strength (MPa)	241 - 281	460 - 530	
Flexural modulus (GPa)	66.6 - 70	69 - 76	
Flexural strength (modulus of rupture) (MPa)	241 - 281	460 - 530	
Shear modulus (GPa)	25.6 - 26.9	26 - 28	
Shear strength (MPa)	166 - 193		
Bulk modulus (GPa)	66.6 - 70	67 - 74	
Poisson's ratio	0.325 - 0.335	0.325 - 0.335	
Shape factor	25	15	
Hardness - Vickers (HV)	100 - 107	152 - 168	
Hardness - Brinell (HB)		145 - 165	
Elastic stored energy (springs) (kJ/m^3)	427 - 576	1460 - 1940	
Fatigue strength at 10^7 cycles (MPa)	106 - 124	152 - 168	
Fatigue strength model (stress amplitude) (MPa) #	96.9 - 136	143 - 179	
Impact & fracture properties			
Fracture toughness (MPa.m^0.5)	30 - 36	26.6 - 26.8	
Toughness (G) (kJ/m^2)	13.3 - 18.9	9.38 - 10.3	
Thermal properties			
Melting point (°C)	582 - 652	475 - 635	
Maximum service temperature (°C)	130 - 150	80 - 100	
Minimum service temperature (°C)	-273	-273	
Thermal conductivity (W/m.°C)	161 - 174	131 - 137	
Thermal conductivity with temperature (W/m.°C) #		146	
Specific heat capacity (J/kg.°C)	934 - 972	913 - 979	
Specific heat capacity with temperature (J/kg.°C) #	1020		
Thermal expansion coefficient (μ strain/°C)	23.4 - 24.6	22.9 - 24.1	
Thermal expansion coefficient with temperature (μ strain/°C) #	22.8		
Thermal expansion coefficient with temperature_Reference temp (°C)	20		
Thermal shock resistance (°C)	146 - 173	266 - 317	

Edupack(Aluminium) part 2

Comparison - MaterialUniverse			
	All Data	Project Data	Ranges
	Averages	# Values	% Change
Thermal shock resistance (°C)	146 - 173	266 - 317	
Thermal distortion resistance (MW/m)	6.66 - 7.32	5.51 - 5.9	
Latent heat of fusion (kJ/kg)	384 - 393	384 - 393	
Electrical properties			
Electrical resistivity (μohm.cm)	3.8 - 4.2	5.1 - 5.3	
Electrical conductivity (%IACS)	41.1 - 45.4	32.5 - 33.8	
Galvanic potential (V)	-0.79 - -0.71	-0.78 - -0.7	
Magnetic properties			
Magnetic type	Non-magnetic	Non-magnetic	
Optical, aesthetic and acoustic properties			
Transparency	Opaque	Opaque	
Acoustic velocity (m/s)	4950 - 5080	4960 - 5210	
Mechanical loss coefficient (tan delta)	0.0001 - 0.002	0.0001 - 0.002	
Healthcare & food			
Food contact	Yes	Yes	
Restricted substances risk indicators			
RoHS 2 (EU) compliant grades?	✓	✓	
REACH Candidate List indicator (0-1, 1 = high risk)	0	0	
SIN List indicator (0-1, 1 = high risk)	0	0	
Critical materials risk			
Contains >5wt% critical elements?	Yes	Yes	
Abundance risk level	Medium	Medium	
Sourcing and geopolitical risk level	High	High	
Environmental country risk level	Very high	Very high	
Price volatility risk level	Medium	Low	
Conflict material risk level	Caution	Caution	
Processing properties			
Metal casting	Unsuitable	Unsuitable	
Metal cold forming	Excellent	Acceptable	
Metal hot forming	Excellent	Excellent	
Metal press forming	Acceptable	Acceptable	
Metal deep drawing	Acceptable	Acceptable	
Machining speed (m/min)	88.4	76.2	
Weldability	Good	Unsuitable	

Edupack(Aluminium) part 3

Comparison - MaterialUniverse		
<input type="checkbox"/> All Data <input type="checkbox"/> Project Data <input type="button" value="Ranges"/> <input type="button" value="Averages"/> # Values % Change Highlight % Change > <input type="text"/>		
Weldability	Good	Unsuitable
Weldability_Notes	Preheating is not required, post weld heat treatment is required	
Durability		
Water (fresh)	Excellent	Excellent
Water (salt)	Acceptable	Acceptable
Weak acids	Excellent	Excellent
Strong acids	Excellent	Excellent
Weak alkalis	Acceptable	Acceptable
Strong alkalis	Unacceptable	Unacceptable
Organic solvents	Excellent	Excellent
Oxidation at 500C	Unacceptable	Unacceptable
UV radiation (sunlight)	Excellent	Excellent
Galling resistance (adhesive wear)	Limited use	Limited use
Flammability	Non-flammable	Non-flammable
Corrosion resistance of metals		
Stress corrosion cracking	Not susceptible	Highly susceptible
Stress corrosion cracking_Notes	Rated in chloride; Other susceptible environments: Halide, water	Rated in chloride; Other susceptible environments: Halide, water
Primary production energy, CO2 and water		
Embodied energy, primary production (virgin grade) (MJ/kg)	186 - 205	180 - 198
Embodied energy, primary production (typical grade) (MJ/kg)	113 - 133	110 - 128
CO2 footprint, primary production (virgin grade) (kg/kg)	13.2 - 14.6	12.7 - 14
CO2 footprint, primary production (typical grade) (kg/kg)	8.16 - 9.53	7.89 - 9.21
Water usage (l/kg)	1130 - 1250	1080 - 1190
Processing energy, CO2 footprint & water		
Roll forming, forging energy (MJ/kg)	6.06 - 6.7	10.6 - 11.7
Roll forming, forging CO2 (kg/kg)	0.455 - 0.502	0.796 - 0.879
Roll forming, forging water (l/kg)	4.14 - 6.21	6.09 - 9.13
Extrusion, foil rolling energy (MJ/kg)	11.8 - 13.1	20.9 - 23.1
Extrusion, foil rolling CO2 (kg/kg)	0.888 - 0.981	1.57 - 1.74
Extrusion, foil rolling water (l/kg)	6.61 - 9.92	10.5 - 15.8

Edupack(Aluminium) part 4

Comparison - MaterialUniverse			
	All Data	Project Data	Ranges
	Averages	# Values	% Change
Extrusion, foil rolling water (l/kg)	6.61 - 9.92	10.5 - 15.8	
Wire drawing energy (MJ/kg)	43.6 - 48.2	77.7 - 85.9	
Wire drawing CO2 (kg/kg)	3.27 - 3.61	5.83 - 6.44	
Wire drawing water (l/kg)	16.4 - 24.6	29.3 - 43.9	
Metal powder forming energy (MJ/kg)	23.2 - 25.6	20.7 - 22.9	
Metal powder forming CO2 (kg/kg)	1.85 - 2.05	1.66 - 1.83	
Metal powder forming water (l/kg)	25.3 - 37.9	22.6 - 33.8	
Vaporization energy (MJ/kg)	15500 - 17100	15500 - 17100	
Vaporization CO2 (kg/kg)	1160 - 1280	1160 - 1280	
Vaporization water (l/kg)	6460 - 9690	6460 - 9690	
Coarse machining energy (per unit wt removed) (MJ/kg)	1.34 - 1.48	2.02 - 2.24	
Coarse machining CO2 (per unit wt removed) (kg/kg)	0.101 - 0.111	0.152 - 0.168	
Fine machining energy (per unit wt removed) (MJ/kg)	9.14 - 10.1	16 - 17.6	
Fine machining CO2 (per unit wt removed) (kg/kg)	0.685 - 0.758	1.2 - 1.32	
Grinding energy (per unit wt removed) (MJ/kg)	17.8 - 19.7	31.4 - 34.8	
Grinding CO2 (per unit wt removed) (kg/kg)	1.34 - 1.48	2.36 - 2.61	
Non-conventional machining energy (per unit wt removed) (MJ/kg)	155 - 171	155 - 171	
Non-conventional machining CO2 (per unit wt removed) (kg/kg)	11.6 - 12.8	11.6 - 12.8	
Recycling and end of life			
Recycle	✓	✓	
Embodied energy, recycling (MJ/kg)	31.8 - 35.1	31 - 34.3	
CO2 footprint, recycling (kg/kg)	2.5 - 2.76	2.44 - 2.69	
Recycle fraction in current supply (%)	42.8 - 47.2	42.8 - 47.2	
Downcycle	✓	✓	
Combust for energy recovery	✗	✗	
Landfill	✓	✓	
Biodegrade	✗	✗	
Geo-economic data for principal component			
Principal component	Aluminum	Aluminum	
Typical exploited ore grade (%)	30.4 - 33.6	30.4 - 33.6	
Minimum economic ore grade (%)	25 - 39	25 - 39	
Abundance in Earth's crust (ppm)	82300 - 84100	82300 - 84100	
Abundance in seawater (ppm)	0.0005 - 0.005	0.0005 - 0.005	

Edupack(Aluminium) part 5

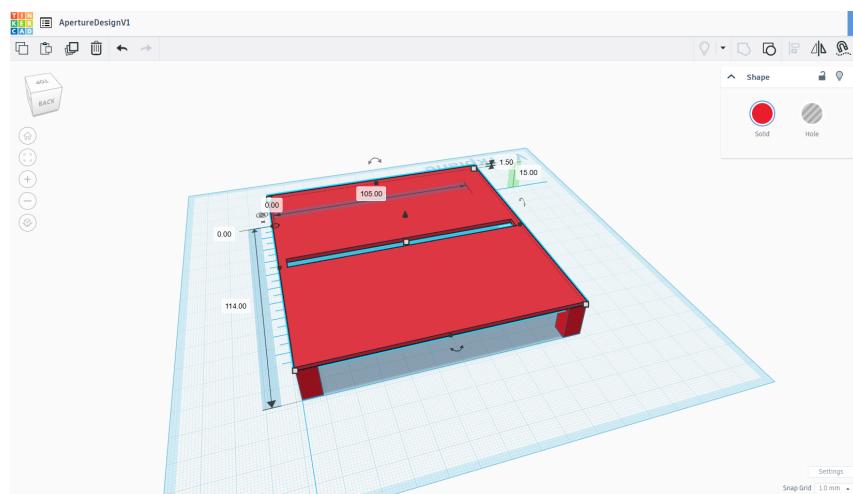
Comparison - MaterialUniverse			
	All Data	Project Data	Ranges
	% Averages	# Values	% Change
Fine machining CO2 (per unit wt removed) (kg/kg)	0.685 - 0.758	1.2 - 1.32	
Grinding energy (per unit wt removed) (MJ/kg)	17.8 - 19.7	31.4 - 34.8	
Grinding CO2 (per unit wt removed) (kg/kg)	1.34 - 1.48	2.36 - 2.61	
Non-conventional machining energy (per unit wt removed) (MJ/kg)	155 - 171	155 - 171	
Non-conventional machining CO2 (per unit wt removed) (kg/kg)	11.6 - 12.8	11.6 - 12.8	
Recycling and end of life			
Recycle		✓	✓
Embodied energy, recycling (MJ/kg)	31.8 - 35.1	31 - 34.3	
CO2 footprint, recycling (kg/kg)	2.5 - 2.76	2.44 - 2.69	
Recycle fraction in current supply (%)	42.8 - 47.2	42.8 - 47.2	
Downcycle		✓	✓
Combust for energy recovery		✗	✗
Landfill		✓	✓
Biodegrade		✗	✗
Geo-economic data for principal component			
Principal component	Aluminum	Aluminum	
Typical exploited ore grade (%)	30.4 - 33.6	30.4 - 33.6	
Minimum economic ore grade (%)	25 - 39	25 - 39	
Abundance in Earth's crust (ppm)	82300 - 84100	82300 - 84100	
Abundance in seawater (ppm)	0.0005 - 0.005	0.0005 - 0.005	
Annual world production, principal component (tonne/yr)	5.73e7	5.73e7	
Reserves, principal component (tonne)	2.69e10	2.69e10	
Links			
Elements in this material	6	6	
Nations of the World	24	28	
ProcessUniverse	107	92	
Producers	14	13	
Reference	26	23	
Shape	24	24	
Full Datasheet	View	View	
#Functional Parameters			
Temperature (°C)	23		
Stress Ratio	-1		
Number of Cycles (cycles)	1e7		

Edupack(Aluminium) part 6

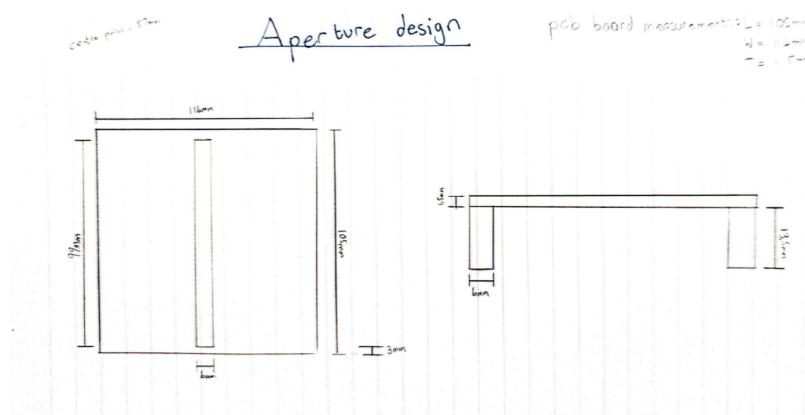
E. Appendix - CAD and 3D Printing

E.1. CAD Modelling and 3D printing

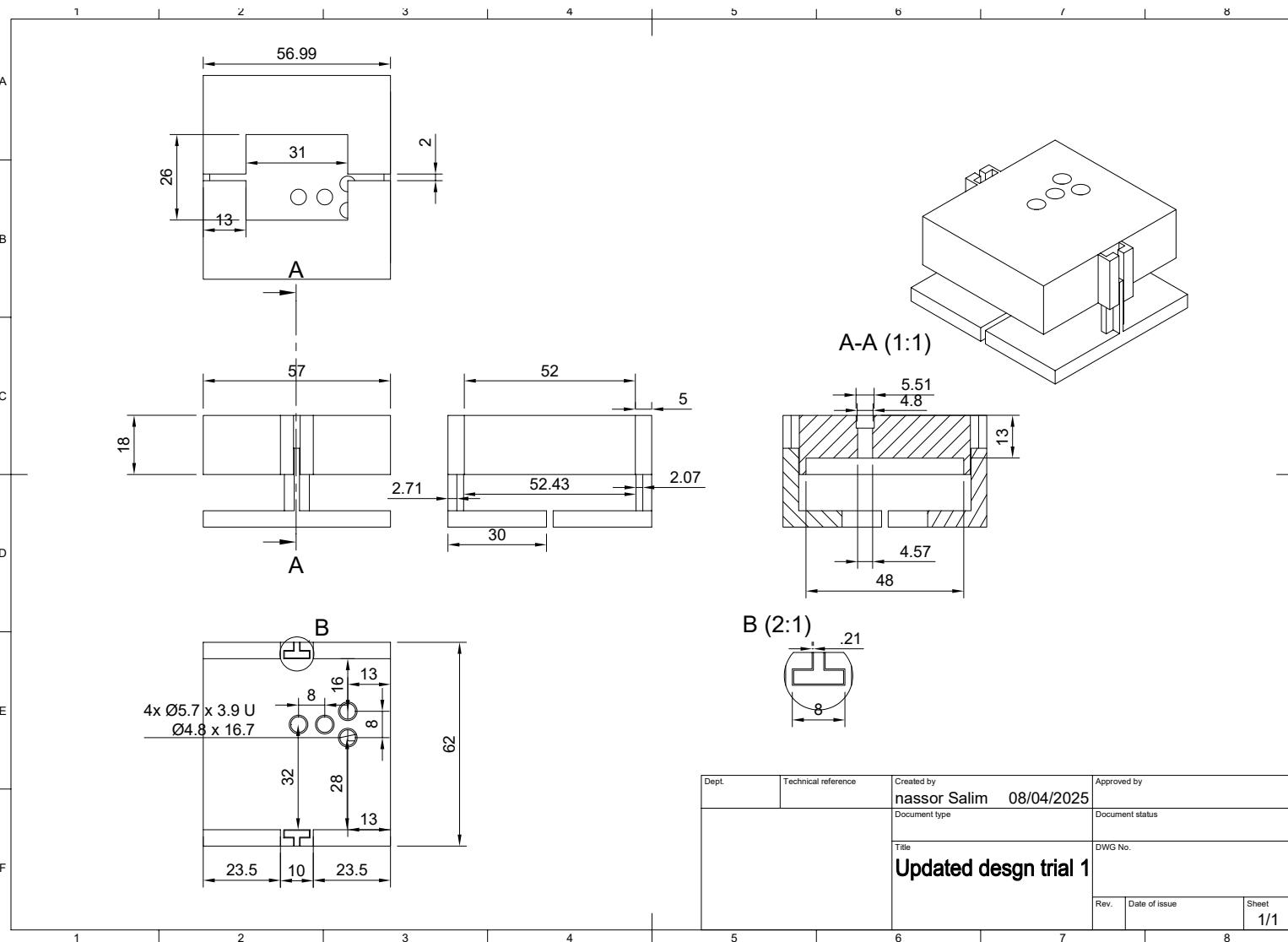
E.1.1. Design Iterations

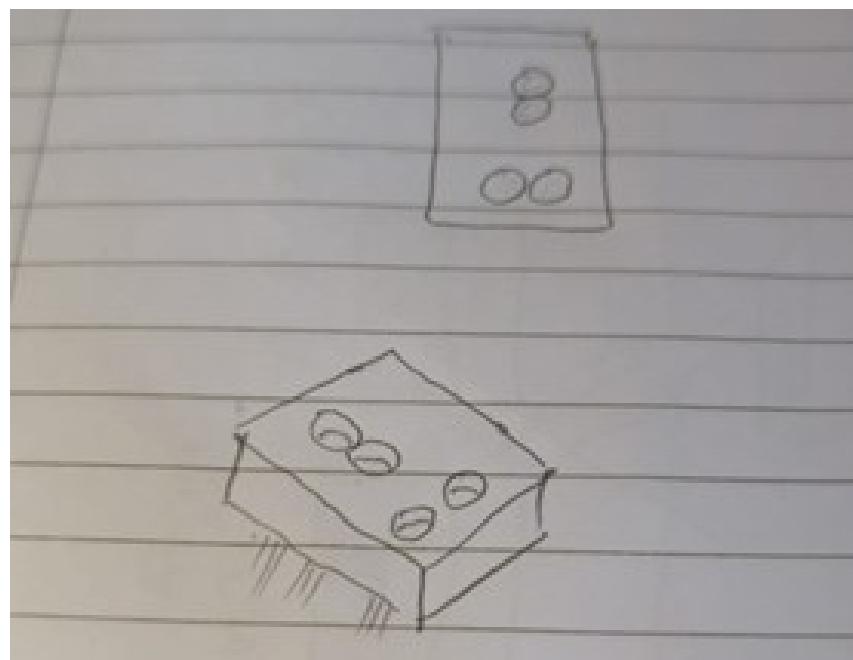


First Design Iteration in Tinkercad

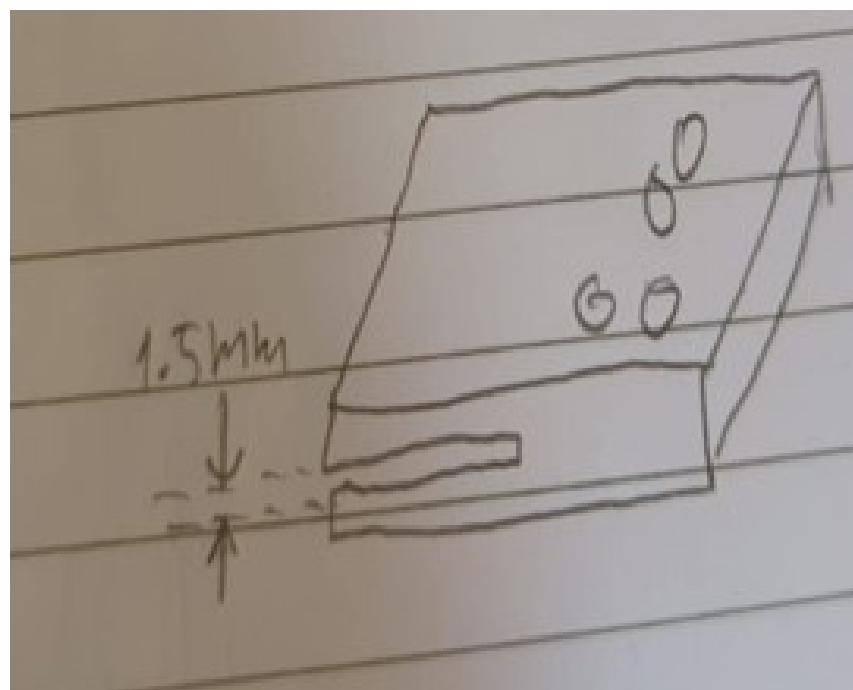


First Design Iteration Sketch

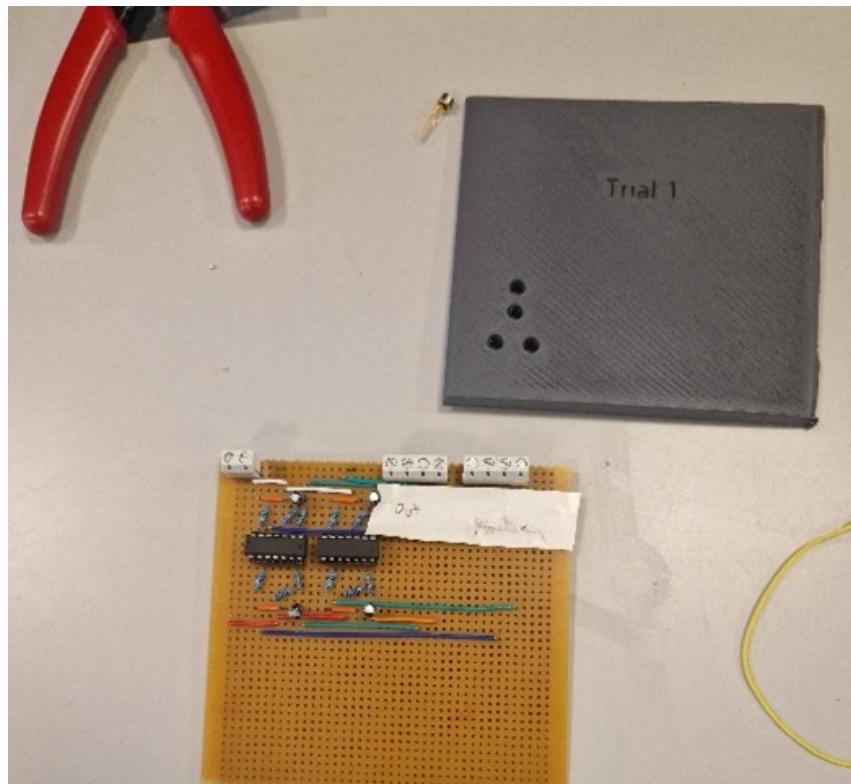




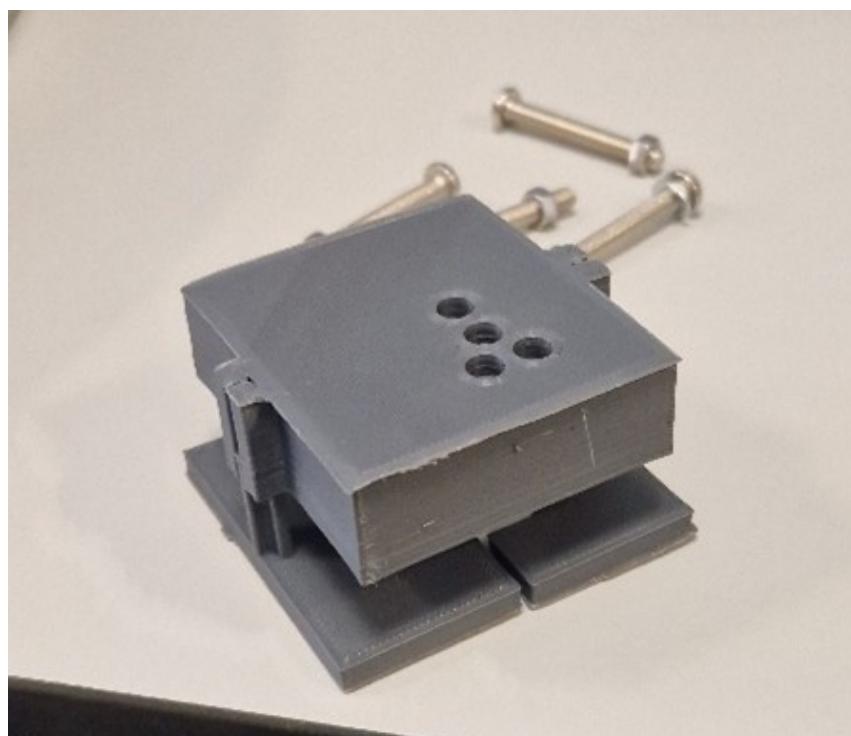
Intial Sketch 1



Intial Sketch 2

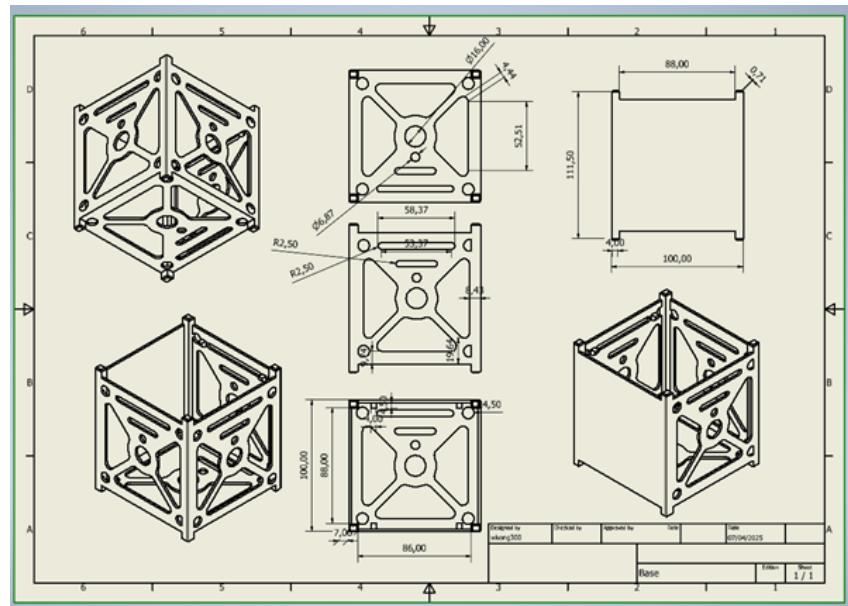


Trial Printing

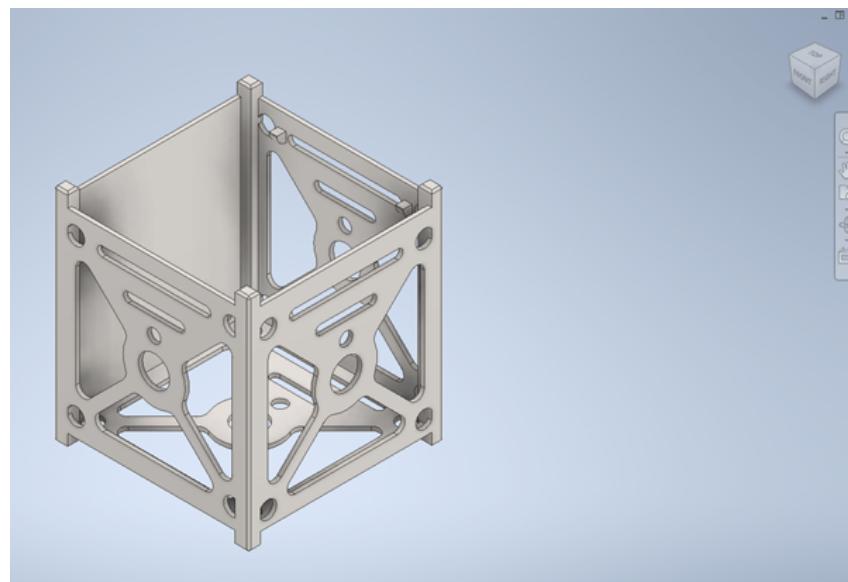


First Print Attempt at Final Design

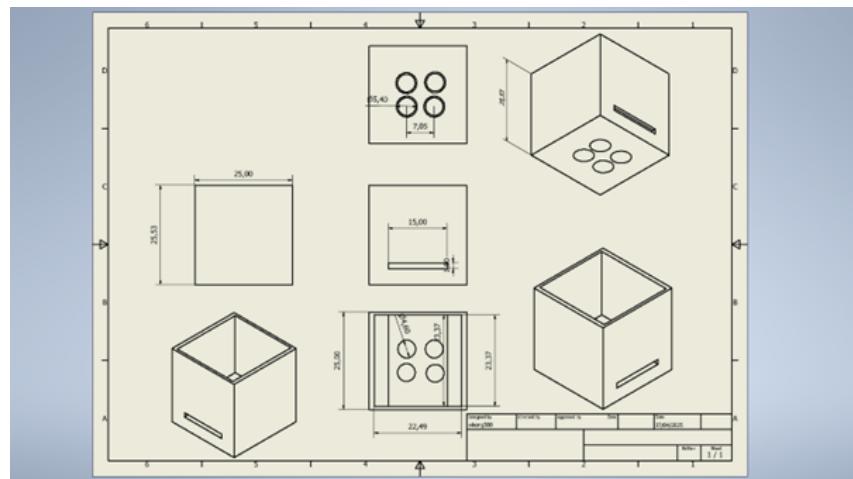
E.1.2. CubeSat Chassis CAD Modelling



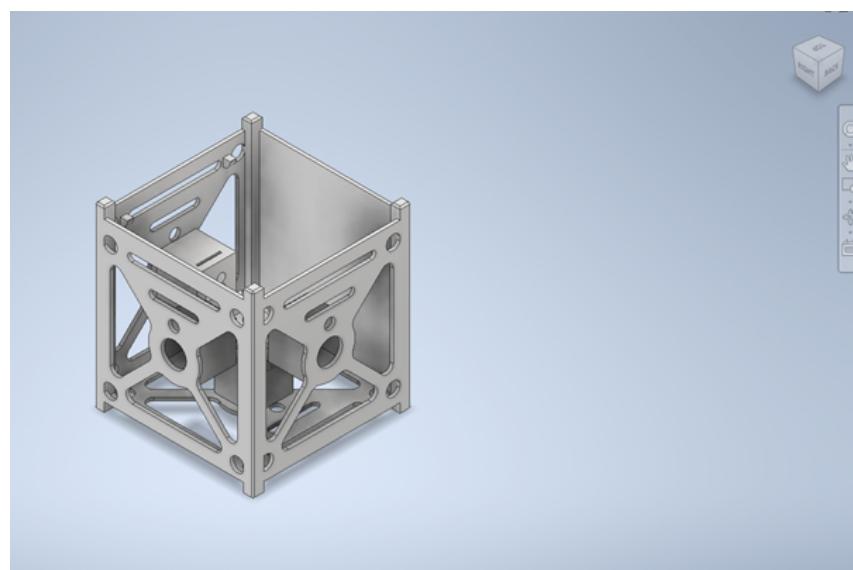
First CubeSat Chassis Design



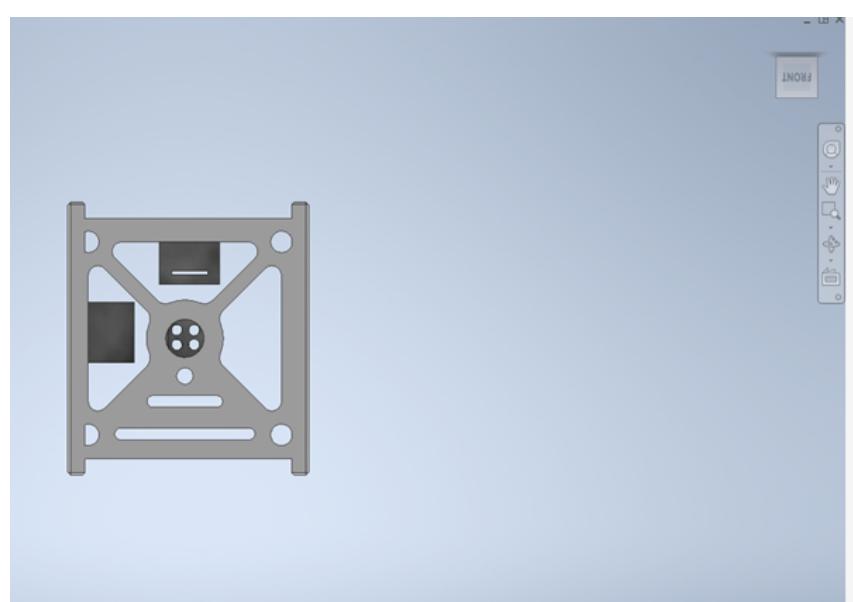
CAD Model of CubeSat Chassis



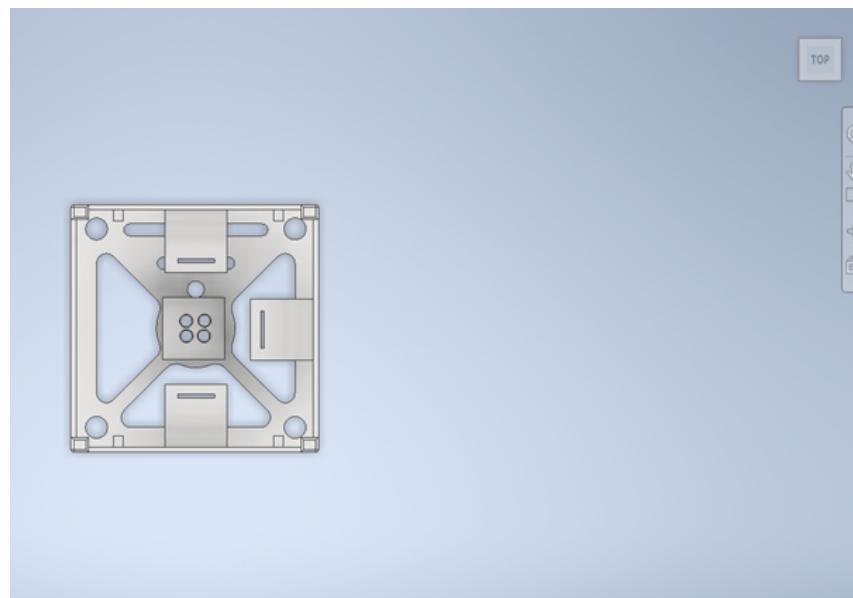
Technical Drawing of Diode Frame



CAD Model of CubeSat Chassis and Diode Frame



Side View of Combined CAD Model



Top View of Combined CAD Model

F. Appendix - Renewable Energy Demonstrator (RED) testbench code

F.1. Appendix - RED LED Strip Code

F.1.1. LED Strip Code for automatic transition

This code is modified to change the LED position every 5 seconds automatically.

```
1 #include <FastLED.h>
2
3 // LED strip configuration
4 #define NUM_LEDS 27
5 #define LED_PIN 5
6 CRGB leds[NUM_LEDS]; // define FastLED datatype
7
8 // automatic transition all leds:
9 const int timeInterval = 5000; // miliseconds
10
11
12 // Communication pin from main Arduino
13 // const int led_status_pin = 12; // Input pin to receive signals
14
15 // LED Positions
16 // MODIFY THESE VALUES TO CHANGE PRESET LED POSITIONS
17
18 //all positions version
19 const int NUM_PRESETS = 25;
20 // modified from 5 positions to all positions automatically every
21 // timeInterval seconds
21 int presetLedPositions[NUM_PRESETS] = {
22     1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25
23 };
24
25 // State variables
26 int currentPresetIndex = 2; // start close to the edge
27 bool lastSignalState = LOW;
```

```

28
29 void setup() {
30   Serial.begin(9600);
31   Serial.println("LED Controller Starting..."); 
32
33   // LED strip
34   FastLED.addLeds<WS2812B, LED_PIN, GRB>(leds, NUM_LEDS);
35   FastLED.setBrightness(255); // brightness (0-255)
36
37   // communication pin
38   // pinMode(led_status_pin, INPUT); not used
39
40   // Clear LED
41   clearAllLeds();
42
43   // starting position
44   currentPresetIndex = 0; // begining position
45
46   // small delay
47   delay(50);
48
49   // Show initial LED position on screen
50   updateLedDisplay(presetLedPositions[currentPresetIndex]);
51
52   // Debug output
53   Serial.print("Starting with preset index: ");
54   Serial.println(currentPresetIndex);
55 }
56
57 void loop() {
58
59
60   // do not wait for input anymore, automatic
61   if (1) {
62
63     // Clear previous LEDs
64     clearAllLeds();
65
66     // Move to next preset
67     currentPresetIndex = (currentPresetIndex + 1) % NUM_PRESETS;
68
69     // Update LED display
70     updateLedDisplay(presetLedPositions[currentPresetIndex]);
71
72     // wait preset speed
73     delay(timeInterval);
74   }

```

```

75
76
77 }
78
79 void clearAllLeds() {
80     for (int i = 0; i < NUM_LEDS; i++) {
81         leds[i] = CRGB::Black;
82     }
83     FastLED.show();
84 }
85
86 // update the location of the LEDS,
87 // based on centerindex
88 void updateLedDisplay(int centerIndex) {
89     if (centerIndex > 0) {
90         leds[centerIndex - 1] = CRGB::White;
91     }
92
93     leds[centerIndex] = CRGB::White;
94
95     if (centerIndex < NUM_LEDS - 1) {
96         leds[centerIndex + 1] = CRGB::White;
97     }
98
99     FastLED.show();
100    Serial.print("LED updated at preset index: ");
101    Serial.print(currentPresetIndex + 1); // Display 1-5 instead of 0-4
102    Serial.print(" (LED position: ");
103    Serial.print(centerIndex);
104    Serial.println(")");
105 }

```

Listing F.1: Cpp Code of the RGB strip with automatic changes

F.1.2. LED Strip Code for Manual 5 location transition

```

1 #include <FastLED.h>
2
3 // LED strip configuration
4 #define NUM_LEDS 27
5 #define LED_PIN 5
6 CRGB leds[NUM_LEDS];
7
8 // Communication pin from main Arduino
9 const int led_status_pin = 12; // Input pin to receive signals
10
11 // Preset LED Positions Configuration

```

```

12 // MODIFY THESE VALUES TO CHANGE PRESET LED POSITIONS
13 const int NUM_PRESETS = 5;
14 int presetLedPositions[NUM_PRESETS] = {
15     3,      // Position 1 - center LED index
16     9,      // Position 2 - center LED index
17     13,     // Position 3 - center LED index (middle)
18     17,     // Position 4 - center LED index
19     23      // Position 5 - center LED index
20 };
21
22 // State variables
23 int currentPresetIndex = 2; // Start at center position (index 2, which
24 // is the 3rd preset)
24 bool lastSignalState = LOW;
25
26 void setup() {
27     Serial.begin(9600);
28     Serial.println("LED Controller Starting...");
29
30     // Initialize LED strip
31     FastLED.addLeds<WS2812B, LED_PIN, GRB>(leds, NUM_LEDS);
32
33     // Initialize communication pin
34     pinMode(led_status_pin, INPUT);
35
36     // Clear all LEDs
37     clearAllLeds();
38
39     // Show initial LED position
40     updateLEDstrip(presetLedPositions[currentPresetIndex]);
41 }
42
43 void loop() {
44     // Read signal from main Arduino
45     bool signalState = digitalRead(led_status_pin);
46
47     // Apply debounce delay
48     delay(5);
49
50     // Read signal again after debounce
51     bool debouncedSignalState = digitalRead(led_status_pin);
52
53     // Only proceed if the signal is stable
54     if (debouncedSignalState == signalState) {
55         // If stable signal state changed from previous loop iteration
56         if (debouncedSignalState != lastSignalState) {
57             // Debug print for any signal change

```

```

58         Serial.print("Signal changed from ");
59         Serial.print(lastSignalState ? "HIGH" : "LOW");
60         Serial.print(" to ");
61         Serial.println(debouncedSignalState ? "HIGH" : "LOW");
62
63         // Detect rising edge (LOW to HIGH transition)
64         if (debouncedSignalState == HIGH && lastSignalState == LOW)
65     {
66
67             Serial.println("Signal received from main Arduino");
68
69             // Clear previous LEDs
70             clearAllLeds();
71
72             // Move to next preset
73             currentPresetIndex = (currentPresetIndex + 1) % NUM_PRESETS;
74
75             // Update LED strip
76             updateLEDstrip(presetLedPositions[currentPresetIndex]);
77         }
78
79         // Update the last signal state
80         lastSignalState = debouncedSignalState;
81     }
82
83     // If signal is not stable after debounce, don't update
84     lastSignalState
85 }
86
87 void clearAllLeds() {
88     for (int i = 0; i < NUM_LEDS; i++) {
89         leds[i] = CRGB::Black;
90     }
91     FastLED.show();
92 }
93
94 void updateLEDstrip(int centerIndex) {
95
96     if (centerIndex > 0) {
97         leds[centerIndex - 1] = CRGB::White;
98     }
99
100    leds[centerIndex] = CRGB::White;
101
102    if (centerIndex < NUM_LEDS - 1) {
103        leds[centerIndex + 1] = CRGB::White;
104    }

```

```

103 FastLED.show();
104 Serial.print("LED updated at preset index: ");
105 Serial.print(currentPresetIndex + 1);
106 Serial.print(" (LED position: ");
107 Serial.print(centerIndex);
108 Serial.println(")");
109 }

```

Listing F.2: Cpp Code of the RGB strip with Manual 5 locations

F.1.3. ARCH controller C++ code

```

1 #include <Servo.h>
2 #include <LiquidCrystal_I2C.h>
3
4 // Servo
5 Servo arch_left;
6 Servo arch_right;
7
8 // LCD screen
9 LiquidCrystal_I2C lcd(0x27, 16, 2);
10
11 // Pin Definitions
12 const int pushButton = 13;
13 const int led_status_pin = 12; // Signal to secondary Arduino for LED
14 control
15
16 // Presets
17
18 const int NUM_PRESETS = 5;
19 int presetAngles[NUM_PRESETS] = {
20     20,    // Position 1 - angle in degrees
21     50,    // Position 2 - angle in degrees
22     90,    // Position 3 - angle in degrees (center) - may not reach, due
23     // to servos?
24     130,   // Position 4 - angle in degrees
25     160    // Position 5 - angle in degrees
26 };
27
28 // State Variables
29 int current_index = 2; // Start at center position (index 2, which is
30 // the 3rd preset)
31 bool lastButtonState = HIGH; // Using INPUT_PULLUP, so HIGH is not
32 // pressed
33
34 void setup() {
35     Serial.begin(9600);

```

```

32
33 // Initialize servos
34 arch_left.attach(5);
35 arch_right.attach(6);
36
37 // Set initial position
38 change_state(current_index);
39
40 // Initialize LED communication pin
41 pinMode(led_status_pin, OUTPUT);
42 digitalWrite(led_status_pin, LOW);
43
44 // Initialize LCD
45 lcd.init();
46 lcd.backlight();
47 updateDisplay();
48
49 // Initialize button
50 pinMode(pushButton, INPUT_PULLUP);
51 }
52
53 void loop() {
54 // Read button state
55 bool buttonState = digitalRead(pushButton);
56
57 // Check for button press (transition from HIGH to LOW)
58 if (buttonState == LOW && lastButtonState == HIGH) {
59 // Move to next preset
60 current_index = (current_index + 1) % NUM_PRESETS;
61
62 // Update servo position
63 change_state(current_index);
64
65 // Update display
66 updateDisplay();
67
68 // Signal the LED Arduino to change LEDs
69 signalLedArduino();
70
71 // Debounce delay
72 delay(200);
73 }
74
75 // Save button state for next iteration
76 lastButtonState = buttonState;
77 }
78

```

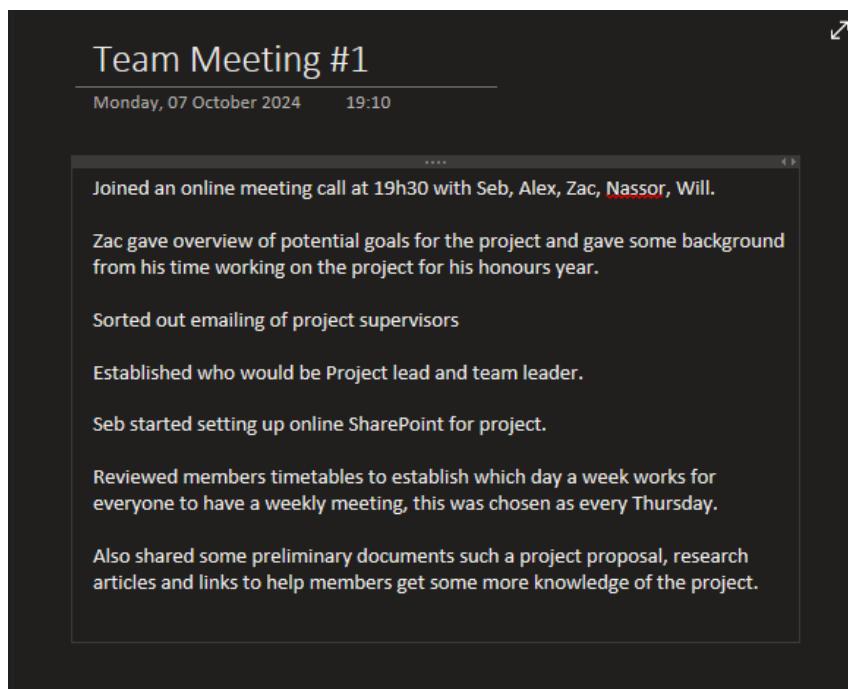
```

79 void change_state(int presetIndex) {
80     // do a calibration each time to ensure correct position
81     arch_left.write(0);
82     arch_right.write(180); // Mirror (180 - 0)
83
84     // add delay to reach location
85     delay(5000); // Adjust delay as needed for your servos
86
87     int angle = presetAngles[presetIndex];
88
89     // Update servos with the new location
90     arch_left.write(angle);
91     arch_right.write(180 - angle); // Mirror movement
92     // debug
93     Serial.print("Moving to preset ");
94     Serial.print(presetIndex + 1);
95     Serial.print(" (angle: ");
96     Serial.print(angle);
97     Serial.println(" degrees)");
98 }
99
100 void signalLedArduino() {
101     // Trigger the RGB Arduino's LED change
102     digitalWrite(led_status_pin, HIGH);
103     delay(50); // Brief pulse to trigger the other Arduino and pass
104         // debounce check
105     digitalWrite(led_status_pin, LOW);
106
107     Serial.println("Sent signal to LED Arduino");
108 }
109 void updateDisplay() {
110     lcd.clear();
111     lcd.setCursor(0, 0);
112     lcd.print("Position: ");
113     lcd.print(current_index + 1);
114
115     lcd.setCursor(0, 1);
116     lcd.print("Alt:");
117     lcd.print(presetAngles[current_index]);
118     lcd.print(" Az:");
119     lcd.print(presetAngles[current_index]);
120 }
```

Listing F.3: C++ Code of the ARCH Arduino

G. Appendix - Meeting Notes

G.1. Meeting Notes



Meeting notes from 2024-10-07

Team Meeting #3

Thursday, 17 October 2024 18:13

Met up on campus for a discussion and catch up in the morning.
Went overview the current list of projects areas that Zac had previously wrote up.
Decided to break it down into two stages: stage one being the research phase with design considerations being taken by Zac, Nassor and Will. Alex and Seb would deal with the different environmental effects.

Stage two would be looking into the signal processing with Alex and Seb, The other part being the dealing with the actual model simulations and CAD work with Zac, Nassor and Will.

Brainstormed some ideas for possible software and tech that would be made use of for the project. Zac gave really good idea of making use of Python due to it being able to interface with MATLAB and also having libraries for satellite modelling. Would also be really helpful for the signal processing side.

Had a quick chat about the 'kick-off' presentation for next week and what we think we could throw in it and possible structure. Nassor put aside spot on the SharePoint for both our Gantt chart and slides for kickoff presentation.

Meeting notes from 2024-10-17

Team Meeting #4

Monday, 18 November 2024 19:55

► Members Present: Will, Alex, Zac, Nassor, Seb
Met to discuss the details of the interim report and how we would plan to set it up. Chose to email previous supervisors and current module leader for clarification on certain aspects on the contents of the interim report. Also began working on setting up a final project plan overview and work progress to show our main supervisor before the semester ends to ensure that we are on the right track with our project and have enough scope covered.|

Meeting notes from 2024-11-18

Team Meeting #5

Thursday, 21 November 2024 17:50

Members present: Zac, Alex, Will, Nas

(This meeting note is to be paired with the word doc made during the meeting as it contains more extensive notes)

Meeting with supervisor, Roberto, on a microsoft teams meeting to discuss what he wants and expects of us for our interim report. Also helped us with establishing our scope of the project better as there were still some areas that didn't have established boundaries on them yet. Alex and Zac took down what he said and suggestions he made in a word doc for us to refer back to at later points such as any questions about the topic that were asked of him along with his layout and contents of the interim report that he was expecting.

Met up again on our project discord to further discuss what he had told us about the project and help clarify who needs to do what. Fixed the current draft report we had to better match the outline given by our supervisor.

Meeting notes from 2024-11-21

Team meeting #6

Saturday, 30 November 2024 11:00

Members present: Alex, Zac, Will, Nassor, Seb

Extra meeting that was held on Saturday morning to help Zac with finishing the Gantt chart. These fixes entailed things such as assigning members to each specific task listed in the Gantt chart and also having each member give an estimated timeline for how long they think their task will take them to complete, such as Will and Nassor discussing how much time they will need for the Ansys portion of the project. We strived to follow the advice of our supervisor to make sure that the tasks we set out can be completed in parallel with each to prevent members having to wait for certain tasks to be completed before being able to begin their section.

Meeting notes from 2024-11-30

Team meeting #7

Tuesday, 10 December 2024 19:15

Members present: Alex, Zac, ~~Nassor~~, Will, Seb

Things left to do for interim:

- 1) Fill opening section on aim, scope and objectives (take from project outline as we wrote them all there from what I can remember)
- 2) add a section on programming to lit review (Alex mentioned this)
- 3) Convert Gantt chart into the report (Zac said Microsoft project was being weird so screenshot will have to do)
- 4) Take from the Gantt chart and describe who's doing what with the remaining work
- 5) Put the RefWorks into the bibliography
- 6) Formatting everything to look good
- 7) Add appendix section to end of doc where we can stick meeting notes and other such things to help show development of project.

-Alex going to ask a license about Altium for us

Meeting notes from 2024-12-10

Team meeting #8

Monday, 03 February 2025 17:00

Members present: Alex, Will, ~~Nasoor~~, Seb

We met on campus in the library after class to discuss the usage of the Google Colab platform for python coding due to us using it in class for the upcoming semester along with its benefits.

~~Nasoor~~ and Will discussed the different CAD software's that they had been testing for potential use in the project such as Altium.

Seb went onto create a GitHub repository for all the python coding that will take place due to its ease of integration into Google colab and its benefits for collaborative use through push and pull requests for updating the repository. Alex discussed the potential boards we could make use of for the project and said that he would begin tracking down the parts and board that he had used for a previous project as he said it had potential for use.

Meeting notes from 2025-02-03

Team Meeting #9

Sunday, 16 February 2025 19:05

Members present: Zac, Will, Nass, Alex, Seb

We met online to discuss new advancements happening with our project. Alex had gone and purchased photodiodes, op-amp and got his old FPGA board from his honours thesis.

We further discussed about acquiring the model for testing purposes and so that we could figure out what adaptations could be made to it to fit our use case. Alex told us that he would go earlier to Uni tomorrow to sort out the circuit setup up with some of the lab technicians.

Will had finished the first mock-up of the CAD design for the chassis in Autodesk and added it to the group OneDrive.

I at first suggested a Trello board for some more organising of the project as since the group task are more split up now, but Zac then suggested using Notion as he had previously used it for a team project.

We would all work at adding the sections to the notion board.

We also noted that we would do a meeting later this week once all the boards and model were collected together.

Zac also mentioned possibly meeting with Roberto to show our progress either next week or potentially sooner depending on how quickly we can get everything together.

Meeting notes from 2025-02-16

Team meeting #10

Sunday, 09 March 2025 17:59

► Members present: Nassor, Will, Seb, Alex, Zac

Discussed meeting the next day to test functionality of the 1D PSD circuit with the university equipment, Will said that he would begin doing material analysis on the PCB, Nassor and Seb discussed the 3D printing of the aperture piece. Zac showed us his work on the python coding side of things and we forked the repo on [Github](#) in case of him needing help or us adding push requests to the repo.

Zac said that he would be going past RS components to purchase the remaining photodiodes needed as we only had the initial two for the circuit.

It was also planned that we would meet on Thursday to go and see the physical simulation device to take measurements and establish what needed fixing for our project to be used with it.

Alex emailed our supervisor to set up a meeting with him to discuss our progress and answer any questions.

Meeting notes from 2025-03-09

Team meeting #11

Friday, 14 March 2025 13:37

Members present: Will, Nassor, Zac, Alex, Seb

This was a meeting with our supervisor Roberto to update him on our groups progress so far.

Alex explained his work on the circuit and how he will soon be able to transfer the circuit from the test breadboard to a soldered final circuit.

Zac showed his progress on the python simulation for testing purposes for the current 1D setup, Seb did mention to Roberto how we discussed looking at just simulation and comparing 1D to 2D as we're not sure about building a physical 2D sensor circuit.

Will and Nassor showed their progress with the CAD work and finite element analysis for the pcb.

Seb and Nassor discussed the progress on modelling and 3D printing of the aperture for the physical circuit testing.

Roberto expressed that he is happy with our progress so far with the project and again reminded us to pay attention to the separate handin dates for the project stages. Such as mentioning that once we have the data it would be advisable to prepare and practice the group presentation since that is the first handin.

Also he reminded us that when writing the thesis document to ensure consistency in our writing between us as we would all be working on separate parts.

Meeting notes from 2025-03-14

Team meeting #12

Sunday, 23 March 2025 18:02

Nass and Seb discussed more about the 3D printing of the aperture for the Photodiodes. Alex clarified that we won't be having the diodes mounted to a breadboard but rather placed into the 3D printed casing and wire the connectors from each back to the OpAmp board.

Alex also mentioned that Roberto got back to him about the 3rd pin on the photodiodes not being used, and he explained that they are only used in high frequency cases to help reduce noise, but since ours is using low frequency we could ignore them.

Nassor showed his new redesign of the aperture for the mounting of the photodiodes to us and would begin printing the design tonight for test fitting and measuring in the lab tomorrow as the slicer software said the print would only be 1h20m, so if further changes to the design need to be made we can easily and quickly reprint the casing.

Zac told us that the simulation side of things had progressed really well and he had implemented the movement and height changes for matching the arch that the practical test would be using. He did mention that there was still a last bit needing work done on data collection within the program.

Will and Nassor said that the material analysis had progressed and should be starting the finite element analysis soon.

As a team we quickly reviewed the feedback comments on our interim report as we were talking about writing our final report, so were double checking what parts needed fixing and what to focus on.

We put a goal of having the physical test data collected by the end of the week.

Meeting notes from 2025-03-23

H. Appendix Software Model Code

H.1. Main Script: main.py

```
1 import os
2 import time
3
4 from memory_profiler import profile
5 import random
6 from arcRotation import arc_movement_vector, rotation_rings
7 from intersectionCalculations import intersection_wrapper # Import for
     calculating line-plane intersection
8 from line import Line # Import for Line object
9 from plane import Plane # Import for Plane object
10 from areas import Areas # Import for target areas
11 import numpy as np # For mathematical operations
12 import plotly.graph_objects as go # For 3D visualization
13
14 import csv
15
16 import logging
17
18 import plotly.io as pio
19 import os
20
21 from PIL import Image
22 import glob
23
24 # Valid logging levels "DEBUG", "INFO", "WARNING", "ERROR", "CRITICAL"
25
26
27 def prepare_output(results_path):
28     """
29         Prepares the output file by creating it if it does not exist
30         and adding a header row.
31     """
32     print("Preparing output")
33
34     # Ensure the 'results_path' is resolved relative to the project
35     # directory
```



```

77
78 def crop_gif_center(input_gif="animation.gif", output_gif=""
79     animation_cropped.gif", crop_width=400, crop_height=300):
80     """
81     Crops the center of each frame in a GIF and saves a new GIF.
82     :param input_gif: Path to input GIF.
83     :param output_gif: Path to save cropped GIF.
84     :param crop_width: Desired width of cropped region.
85     :param crop_height: Desired height of cropped region.
86     """
87     with Image.open(input_gif) as im:
88         frames = []
89         try:
90             while True:
91                 frame = im.copy().convert("RGBA")
92                 width, height = frame.size
93                 left = (width - crop_width) // 2
94                 top = (height - crop_height) // 2
95                 right = left + crop_width
96                 bottom = top + crop_height
97                 cropped = frame.crop((left, top, right, bottom))
98                 frames.append(cropped)
99                 im.seek(im.tell() + 1)
100            except EOFError:
101                pass
102
103            if frames:
104                frames[0].save(
105                    output_gif,
106                    format="GIF",
107                    append_images=frames[1:],
108                    save_all=True,
109                    duration=im.info.get("duration", 500),
110                    loop=0
111                )
112                print(f"Cropped GIF saved as {output_gif}")
113            else:
114                print("No frames found in input GIF.")
115
116 def crop_image_center(input_path="static_plot.png", output_path=""
117     static_plot_cropped.png",
118                     crop_width=400, crop_height=300):
119     """
120     Crops the center of a single image (PNG, JPEG, etc.)
121     """
122     with Image.open(input_path) as img:

```

```

122     width, height = img.size
123     left = (width - crop_width) // 2
124     top = (height - crop_height) // 2
125     right = left + crop_width
126     bottom = top + crop_height
127     cropped = img.crop((left, top, right, bottom))
128     cropped.save(output_path)
129     print(f"Cropped static image saved as {output_path}")
130
131
132 def initialise_planes_and_areas(config):
133     """
134     Initialises all planes and the target area.
135     Returns: sensorPlane, sourcePlane, aperturePlane, and sensorArea
136     """
137
138     # Define the source plane
139     # Defines its position (centre point), its direction (facing down)
140     sourcePlane = Plane("Source Plane", **config.planes["source_plane"])
141
142     # Define the sensor plane
143     sensorPlane = Plane("Sensor Plane", **config.planes["sensor_plane"])
144
145     # Define the intermediate plane
146     aperturePlane = Plane("Aperture Plane", **config.planes["aperture_plane"])
147
148     # Extract sensor keys from JSON file
149     sensor_keys = config.sensor_areas.keys()
150
151     # Define the sensor area (target area on the sensor plane)
152     sensorAreas = [Areas(**config.sensor_areas[sensor]) for sensor in
153     sensor_keys]
154
155     # Extract aperture keys from JSON aperture_areas
156     aperture_keys = config.aperture_areas.keys()
157     apertureAreas = [Areas(**config.aperture_areas[aperture]) for
158     aperture in aperture_keys]
159
160     return sensorPlane, sourcePlane, aperturePlane, sensorAreas,
161     apertureAreas
162
163     """
164
165     Initialises a 3D plot using Plotly.
166     Generates a global axis for the plot.

```

```

165     Returns: A Plotly figure object.
166
167     """
168
169     lims = 20
170     fig = go.Figure()
171     fig.update_layout(
172         scene=dict(
173             xaxis_title='X',
174             yaxis_title='Y',
175             zaxis_title='Z',
176             aspectmode="cube", # Ensures uniform scaling
177             xaxis=dict(title="X-Axis", range=[-lims, lims]), # Set
178             equal_ranges
179             yaxis=dict(title="Y-Axis", range=[-lims, lims]), # Adjust
180             based_on_your_data
181             zaxis=dict(title="Z-Axis", range=[-lims, lims])) # Keep Z
182             range_similar
183         ),
184         title="Sensor illumination simulation"
185     )
186
187     global_axis = [sensorPlane.right, sensorPlane.up, sensorPlane.
188 direction]
189
190     axis_colours = ['red', 'green', 'blue']
191     axis_names = ['Right (x)', 'Up (y)', 'Normal (z)']
192
193     for i in range(3):
194         unit_vector = global_axis[i] / np.linalg.norm(global_axis[i]) # Normalize
195         start = np.array([0, 0, 0]) # Origin of local axes at plane's
196         position
197         end = 0 + unit_vector # Unit length
198
199         fig.add_trace(go.Scatter3d(
200             x=[start[0], end[0]],
201             y=[start[1], end[1]],
202             z=[start[2], end[2]],
203             mode='lines+markers',
204             line=dict(color=axis_colours[i], width=5),
205             marker=dict(size=8, color=axis_colours[i], opacity=0.8),
206             name=f"{axis_names[i]}",
207             showlegend=True,
208             hovertext=[f"Global axis: {axis_names[i]}"] # Appears when
209             hovering
210         ))
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304

```

```

205     fig.update_traces(showlegend=True)
206     return fig
207
208
209 def visualise_environment(fig, planeObject, colour): # sensorPlane,
210     sourcePlane, aperturePlane, sensorArea):
211     """
212     Adds planes and areas to the 3D plot for visualization.
213     Returns: Updated Plotly figure.
214     """
215     fig = planeObject.planes_plot_3d(fig, colour)
216     return fig
217
218 def update_lines_global_positions(lines, new_source_plane):
219     """
220     Updates the global positions of all lines based on their local
221     positions within new source plane.
222     :return:
223         lines: new list of lines with updated global positions.
224     """
225     # Initialize their global positions based on the plane
226     for line in lines:
227         line.update_global_position(new_source_plane)
228         line.direction = new_source_plane.direction
229
230
231 def create_lines_from_plane(source_plane, num_lines):
232     """
233     Generates random line positions in the plane's local coordinate
234     system.
235
236     Args:
237         source_plane (Plane): The source plane object.
238         num_lines (int): Number of lines to generate.
239
240     Returns:
241         list: List of Line objects.
242     """
243     local_positions = source_plane.random_points(num_lines) # Local
244     coordinates
245     # print(f"Local positions: {local_positions}")
246     # print(f"Number of lines: {len(local_positions)}")
247     lines = [

```

```

248         Line([x, y, 0], source_plane.direction, line_id=idx)
249         for idx, (x, y) in enumerate(local_positions)
250     ]
251
252     return lines
253
254
255 def intersection_checking(targetArea, intersection_coordinates):
256     """
257     Gets input of target areas and coordinate of intersection
258     Checks if the intersection point is in the target area.
259     """
260     result = None
261
262     for target in targetArea:
263         # Check if the intersection point is in the target area
264         result = target.record_result(intersection_coordinates)
265         # logging.debug(f"Checking intersection with {target.title}...")
266
267         if result == 1: # Hit occurs
268             return 1, target
269         if result == 0: # Miss
270             return 0, 0
271     else:
272         return -1, 0
273
274
275 def evaluate_line_results(sensorPlane, sensorArea, aperturePlane,
276                           apertureAreas, lines):
277     """
278     Checks intersections of lines with the sensor plane and evaluates
279     whether they hit the target area.
280
281     Updates line object internal parameters with the results.
282
283     Args:
284         sensorPlane: The plane intersecting with the lines.
285         sensorArea: List of all sensor objects - target areas to
286                     evaluate hits.
287         aperturePlane:
288         apertureAreas:
289         lines: List of Line objects.
290
291     Returns:
292         hit: number of hits.
293         miss: number of misses.
294     """

```

```

292     hit = 0
293     miss = 0
294     hit_list = []
295     miss_list = []
296
297     # Reset previous sensor illumination values
298     for sensors in sensorArea:
299         sensors.illumination = 0
300
301     for line in lines:
302         line.result = 0
303         # Calculate intersection between the line and the aperture plane
304         aperture_intersection_coordinates = intersection_wrapper(
305             aperturePlane, line)
306         # Set intersection coordinate of line object
307         line.intersection_coordinates =
308             aperture_intersection_coordinates
309
310         # Check if intersection with apertures
311         aperture_intersection, _ = intersection_checking(apertureAreas,
312             aperture_intersection_coordinates)
313         if aperture_intersection == 1: # Hit, at apertures
314
315             # Get intersection coordinates with sensor plane
316             sensor_intersection_coordinates = intersection_wrapper(
317                 sensorPlane, line)
318
319             # Check intersection with sensor areas
320             sensor_intersection, sensor = intersection_checking(
321                 sensorArea, sensor_intersection_coordinates)
322             line.intersection_coordinates =
323                 sensor_intersection_coordinates
324
325             if sensor_intersection == 1: # Intersection occurs at
326                 sensor
327                     hit, line.result, sensor.illumination = hit + 1, 1,
328                     sensor.illumination + 1
329
330                     hit_list.append(line.line_id)
331
332                     continue # Move to next line
333                     if sensor_intersection == 0:
334                         miss += 1
335
336                     miss_list.append(line.line_id)
337
338                     continue

```

```

331     else:
332
333         miss += 1
334         miss_list.append(line.line_id)
335         continue
336
337     return hit, miss, hit_list, miss_list
338
339
340 def handle_results(sensor_objects, sim_idx, idx, config):
341     """
342     Logs one row of hit counts per sensor for a given simulation and arc
343     position.
344     Structure: [sim, idx, Sensor A, Sensor B, ..., Sensor N]
345     """
346
347     file_path = "../data/sensor_results.csv"
348     write_header = not os.path.exists(file_path) or (sim_idx == 0 and
349     idx == 0)
350
351     # Prepare row
352     row_data = [sim_idx, idx]
353     sensor_titles = [sensor.title for sensor in sensor_objects]
354     sensor_hits = [sensor.illumination for sensor in sensor_objects]
355
356     # Write header if needed
357     if write_header:
358         header = ["sim", "idx"] + sensor_titles
359         with open(file_path, "w", newline="") as f:
360             writer = csv.writer(f)
361             writer.writerow(header)
362
363     # Write data row
364     with open(file_path, "a", newline="") as f:
365         writer = csv.writer(f)
366         writer.writerow(row_data + sensor_hits)
367
368
369 def do_rotation(theta, axis):
370     """
371     Gets rotation matrix for specified axis and angle.
372
373     Args:
374         theta: The angle of rotation in radians.
375         axis: The axis of rotation.
376
377     Returns:

```

```

376     The rotation matrix for the specified axis and angle.
377 """
378
379     if axis == "x":
380         R = np.array([
381             [1, 0, 0],
382             [0, np.cos(theta), -np.sin(theta)],
383             [0, np.sin(theta), np.cos(theta)]
384         ])
385     elif axis == "y":
386         R = np.array([
387             [np.cos(theta), 0, np.sin(theta)],
388             [0, 1, 0],
389             [-np.sin(theta), 0, np.cos(theta)]
390         ])
391     elif axis == "z":
392         R = np.array([
393             [np.cos(theta), -np.sin(theta), 0],
394             [np.sin(theta), np.cos(theta), 0],
395             [0, 0, 1]
396         ])
397     else:
398         print("Invalid axis")
399         R = np.array([1, 1, 1])
400
401
402
403 def rotation_test(angle, axis, vectors):
404 """
405 Test function for rotation matrix.
406 :param angle:
407 :param axis:
408 :param vectors:
409 :return:
410     Print rotation matrix.
411 """
412
413     rotation_matrix = do_rotation(angle, axis)
414
415     # print(f"Rotation matrix for {axis} axis: \n{rotation_matrix}")
416     print(f"Rotation about {axis} axis by {np.degrees(angle)}degree {angle} rad")
417     for vector in vectors:
418         rotated_vector = np.dot(rotation_matrix, vector)
419         print(f"{vector} rotated -> {np.round(rotated_vector, 2)}")
420
421

```

```

422 def setup_initial_pose(source_plane, theta, rotation_axis, all_positions
423     ):
424     """
425     Sets up the initial position and orientation of the plane before it
426     starts moving along the arc.
427
428     1. Creates copy of the source plane.
429     2. Applies an initial rotation to the align the plane
430     3. Translates the plane to the starting position for the arc
431
432     Args:
433         source_plane (Plane): The original source plane from which
434         movement begins.
435         theta (float): The rotation angle (in degrees) applied before
436         movement.
437         rotation_axis (str): The axis of rotation.
438         all_positions (list): Contains all positions around the arc
439
440     Returns:
441         Plane: The transformed plane at the starting position of the arc
442     .
443     """
444
445     # Create a copy of the source plane
446     start_pose_plane = Plane(
447         f"Copy of source",
448         source_plane.position,
449         source_plane.direction,
450         source_plane.width,
451         source_plane.length
452     )
453
454     # Before movement, print initial pose
455     start_pose_plane.print_pose()
456
457     # Apply initial rotation to align normal vector
458     start_pose_plane.rotate_plane(do_rotation(np.radians(theta)),
459         rotation_axis))
460     start_pose_plane.title = f"Plane rotated {theta:.0f}degree in {rotation_axis}-axis"
461     start_pose_plane.print_pose()
462
463     # Set position to the first computed arc position instead of
464     # translating manually
465     logging.debug(f"Moving to initial arc position: {np.round(
466         all_positions[0], 2)}")
467

```

```

460 # start_pose_plane.position = np.array(all_positions[0])
461
462 translation_vector = arc_movement_vector(start_pose_plane,
463 all_positions[0])
464 start_pose_plane.translate_plane(translation_vector)
465
466 start_pose_plane.title = "Plane moved to initial arc position"
467 start_pose_plane.print_pose()
468
469
470
471 def generate_arc_animation(fig, rotated_planes, lines_traces, results):
472 """
473 Generates an animated visualization of the arc movement.
474
475 Data type notes --
476     Plane trace is type list, shape (length rotated_planes,)
477     Plane trace [0] type: <class 'plotly.graph_objs._mesh3d.Mesh3d
478 '>: shape ()
479     Axis trace is type list, shape (length rotated_planes, 3)
480     Axis trace [0] type: <class 'list'>: shape (3,)
481     Line trace is type list, shape (length rotated_planes, 2)
482     Line trace [0] type: <class 'list'>: shape (2,)
483
484 Args:
485     fig (Plotly Figure): The figure used for visualization.
486     rotated_planes (list): The list of planes from
487     move_plane_along_arc().
488     lines_traces (list): List of Line objects for each plane.
489     results:
490
491 Returns:
492     fig (Plotly Figure): Updated figure with animation.
493 """
494
495 plane_trace = []
496 axis_traces = []
497 frame_titles = list(np.zeros(len(rotated_planes)))
498 percentage = []
499
500 num_frames = len(rotated_planes)
501
502 colours = ["yellow"] + ["yellow"] * (len(rotated_planes) - 1)
503
504 for idx, plane in enumerate(rotated_planes):
505     # Debugging - Check if planes are being added
506     logging.debug(f"Preparing elements for frame {idx} for plane at
position {plane.position}")

```

```

503
504     # Get plane and axis traces
505     plane_trace.append(plane.planes_plot_3d(go.Figure(), colours[idx]
506                           ].data[
507                               0]) # makes plane_trace[idx] type = "
508     plotly.graph_objs._mesh3d.Mesh3d"
509     axis_traces.append(list(plane.plot_axis(go.Figure()).data)) # makes axis_traces[idx] type = "tuple"
510
511     total = results[idx][0] + results[idx][1]
512
513     percentage.append((results[idx][0] / total) * 100)
514
515     # frame_titles[idx] = f"Position {idx} - Hits {results[idx]
516     # [0]:.0f}, Misses {results[idx][1]:.0f}"
517     frame_titles[idx] = f"Position {idx} - Hits {percentage[idx]:.0f}
518     } "
519
520
521     # check_fig_data(fig)
522
523     for traces in fig.data:
524         fig.add_trace(traces)
525
526
527     # Prepare the actual frames
528     frames = [
529         go.Frame(
530             data=[
531                 plane_trace[i], # Single plane
532                 *axis_traces[i], # Three axis traces
533                 *lines_traces[i], # Two line traces
534             ],
535             name=f"Frame {i}",
536             layout=go.Layout(title=frame_titles[i])
537         )
538         for i in range(num_frames)
539     ]
540
541     # Add animation controls
542     fig.update_layout(
543         updatemenus=[{
544             "buttons": [
545                 {
546                     "args": [None, {"frame": {"duration": 500, "redraw": True}, "fromcurrent": True}],
547                     "label": "Play",
548                     "method": "animate"
549                 },
550             ],
551             "direction": "left",
552             "x": 0,
553             "xanchor": "right",
554             "y": 0,
555             "yanchor": "bottom"
556         }
557     )

```

```

544         {
545             "args": [[None], {"frame": {"duration": 0, "redraw": True}, "mode": "immediate",
546                         "transition": {"duration": 0}}],
547             "label": "Pause",
548             "method": "animate"
549         }
550     ],
551     "direction": "left",
552     "pad": {"r": 10, "t": 87},
553     "showactive": False,
554     "type": "buttons",
555     "x": 0.1,
556     "xanchor": "right",
557     "y": 0,
558     "yanchor": "top"
559 ],
560 # Add a slider for manual frame selection
561 sliders=[{
562     "active": 0,
563     "yanchor": "top",
564     "xanchor": "left",
565     "currentvalue": {
566         "font": {"size": 16},
567         "prefix": "Position: ",
568         "visible": True,
569         "xanchor": "right"
570     },
571     "transition": {"duration": 300, "easing": "cubic-in-out"},
572     "pad": {"b": 10, "t": 50},
573     "len": 0.9,
574     "x": 0.1,
575     "y": 0,
576     "steps": [
577         {
578             "args": [
579                 [f"Frame {k}"],
580                 {"frame": {"duration": 300, "redraw": True},
581                 "mode": "immediate",
582                 "transition": {"duration": 300}}
583             ],
584             "label": str(k),
585             "method": "animate"
586         }
587         for k in range(num_frames)
588     ]
589 }]

```

```

590 )
591
592 # Set the frames to the figure
593 fig.frames = frames
594
595 # output_dir = "animation_frames"
596 # os.makedirs(output_dir, exist_ok=True)
597 #
598 # # Save the static scene objects (sensor plane, aperture, areas,
599 # etc.)
600 # static_traces = list(fig.data[:]) # Copy existing traces (before
601 # animation)
602 #
603 # for i in range(len(rotated_planes)):
604 #     temp_fig = go.Figure()
605 #
606 #     # Add static traces (sensor plane, areas, etc.)
607 #     for trace in static_traces:
608 #         temp_fig.add_trace(trace)
609 #
610 #     # Add animated elements for current frame
611 #     temp_fig.add_trace(plane_trace[i])
612 #     for axis_trace in axis_traces[i]:
613 #         temp_fig.add_trace(axis_trace)
614 #     for line_trace in lines_traces[i]:
615 #         temp_fig.add_trace(line_trace)
616 #
617 #     temp_fig.update_layout(
618 #         scene=fig.layout.scene,
619 #         title=frame_titles[i],
620 #         autosize=False,
621 #         width=1000,
622 #         height=800,
623 #         margin=dict(l=0, r=0, t=40, b=0),
624 #         scene_camera=dict(eye=dict(x=1.2, y=1.2, z=1.2)),
625 #         showlegend=False)
626 #
627 #     # Export the frame
628 #     pio.write_image(temp_fig, f"{output_dir}/frame_{i:03d}.png",
629 # width=1000, height=800, scale=3)
630
631 def generate_static_arc_plot(config, fig, rotated_planes, line_objects):
632 """
633 Generates a static 3D plot of all plane positions in the arc.

```

```

634
635     Args:
636         fig (Plotly Figure): The figure used for visualization.
637         rotated_planes (list): The list of planes from
638         move_plane_along_arc().
639
640     Returns:
641         fig (Plotly Figure): Updated figure with static planes plotted.
642     """
643
644     for idx, plane in enumerate(rotated_planes):
645         if idx == 0:
646             fig = plane.planes_plot_3d(fig, "yellow")
647         else:
648             fig = plane.planes_plot_3d(fig, "blue")
649
650         fig = plane.plot_axis(fig) # Adds axis vectors
651
652         # logging.debug(f"Adding line traces for plane {idx}")
653         # for line in line_objects[idx]:
654         #     fig.add_trace(line)
655
656     # Match the animated plot layout
657     fig.update_layout(
658         autosize=False,
659         width=1000,
660         height=800,
661         margin=dict(l=0, r=0, t=40, b=0),
662         scene_camera=dict(eye=dict(x=1.2, y=1.2, z=1.2)),
663         showlegend=False
664     )
665
666     if config.output["save_static_png"]:
667         logging.info("Saving static plot as PNG file.")
668         pio.write_image(fig, "static_plot.png", width=1000, height=800,
669         scale=1)
670         logging.info("Static plot saved as static_plot.png")
671     else:
672         logging.info("Static plot image disabled.")
673
674
675
676 def plane_pose_initialisation(original_plane):
677     """

```

```

678     Creates a copied plane object for initial pose.
679
680     :arg:
681         original_plane: Plane object to be copied.
682
683     :return:
684         new_plane: Copied plane object.
685     """
686
687     new_plane = Plane(f"New plane", original_plane.position,
688                       original_plane.direction, original_plane.width,
689                           original_plane.length)
690
691     new_plane.right, new_plane.up, new_plane.direction = original_plane.
692     right, original_plane.up, original_plane.direction
693
694
695     return new_plane
696
697
698
699 def calculate_rotation_matrix(position, direction):
700     """
701
702     Takes input position and direction to calculate arbitrary rotation
703     matrix.
704
705     Using Rodrigues' rotation formula.
706
707     :arg:
708         position (3x1 matrix): Position vector.
709         direction (3x1 matrix): Direction vector.
710         idx (int): Plane index. Used for logging.
711
712     :return:
713         R (3x3 matrix): Rotation matrix.
714         required_angle (float): Angle (rad) of rotation in calculated
715         arbitrary rotation matrix.
716     """
717
718
719     logging.debug("Calculating new rotation matrix")
720
721     # Calculate target local z axis
722     target_z = np.array([0, 0, 0]) - position
723
724     # Normalized z-direction
725     target_z_norm = target_z / np.linalg.norm(target_z, keepdims=True)
726
727
728     # Calculate axis of rotation
729     required_rotation_axis = np.cross(direction, target_z_norm)
730     normalised_axis = required_rotation_axis / np.linalg.norm(
731         required_rotation_axis, keepdims=True)
732
733
734     # Calculate angle of rotation
735     required_angle = np.arccos(np.clip(np.dot(direction, target_z_norm),
736                                 -1.0, 1.0))
737
738
739     # Apply Rodrigues' rotation formula
740     # Compute skew-symmetric cross-product matrix of rotation_axis

```

```

719     K = np.array([
720         [0, -normalised_axis[2], normalised_axis[1]],
721         [normalised_axis[2], 0, -normalised_axis[0]],
722         [-normalised_axis[1], normalised_axis[0], 0]
723     ])
724
725     # Rodrigues' rotation formula
726     R = (
727         np.eye(3) +
728         np.sin(required_angle) * K +
729         (1 - np.cos(required_angle)) * np.dot(K, K)
730     )
731
732     return R, required_angle
733
734
735 def determine_movement_type(idx, sequence_ID, secondary_angle):
736     """
737     Checks for movement type
738     Simplified logic from previous implementation
739     :arg:
740         idx (int): Plane index. Used for logging.
741         sequence_ID (int): Indicates type of movement sequence. (2 =
742         horizontal circles, or 1 = vertical circles)
743         secondary_angle (list): Contains list of angles (ID = 2/1, theta
744         /phi) from the polar coordinates of each position.
745     :return:
746         1 = First position
747         2 = Horizontal circles
748         3 = Same meridian
749         4 = Different meridian
750
751         None = Problem
752     """
753
754     if idx == 0:
755         # First position
756         return 1
757     elif sequence_ID == 2:
758         # Horizontal circles
759         return 3
760     elif sequence_ID == 1:
761         # Vertical circles
762         if secondary_angle[idx - 1] == secondary_angle[idx]:
763             # Same meridian
764             return 3
765         else:
766             # Different meridian

```

```

764         return 2
765
766     return None
767
768
769 def initialise_new_circle(plane, position):
770     """
771     Creates a copy of a plane, calculates translation vector and applies
772     it to the plane, moving it to the next position.
773
774     :arg:
775         plane (Plane): Plane object.
776         position (3x1 matrix): Position vector.
777         direction (3x1 matrix): Direction vector.
778         idx (int): Plane index. Used for logging.
779
780     :return:
781     """
782
783     # Create copy of a plane
784     new_plane = plane_pose_initialisation(plane)
785
786     # Calculate new translation vector
787     translation_vector = arc_movement_vector(new_plane, position)
788
789     # Apply translation
790     new_plane.translate_plane(translation_vector)
791
792     return new_plane
793
794
795 def primary_rotation_handling(sequence_ID, new_plane, next_position,
796     required_angle, rotation_axis):
797     """
798     Handles the differing primary rotation mechanisms depending on type
799     of movement sequence.
800
801     For vertical circles, the primary rotation matrix has to be
802     generated for each new circle (about arbitrary axis).
803
804     For horizontal circles, the primary rotation matrix is standard
805     about a defined axis.
806
807     :return:
808         primary_axis
809         required_angle
810         plane (object)
811     """
812
813     R_p = None
814     # required_angle = None
815

```

```

806     if sequence_ID == 1: # Vertical circles
807         R_p, required_angle = calculate_rotation_matrix(next_position,
808             new_plane.direction)
809         logging.debug(
810             f"Required angle for new primary rotations: {np.round(np.
811 degrees(required_angle), 2)}degree about arbitrary axis")
812
813     elif sequence_ID == 2: # Horizontal circles
814         # Apply rotation to align with the origin in the z axis
815         R_p = do_rotation(required_angle, rotation_axis[0])
816         logging.debug(f"Rotating {np.round(np.degrees(required_angle),
817             2)}degree around {rotation_axis[0]}-axis")
818
819     return R_p, required_angle, new_plane
820
821
822
823 def move_plane_along_arc(start_plane, all_positions, primary_angle,
824     rotation_axis, secondary_angle, sequence_ID):
825     """
826     Moves the plane along a predefined arc while updating line positions
827     .
828
829     Args:
830         start_plane (Plane): A plane object representing the initial
831             position and orientation.
832
833         all_positions (list): A list of position vectors [x, y, z]
834             corresponding to the points along the arc where the plane will be
835             moved.
836
837             Each item in this list represents a point in 3D space.
838
839         primary_angle (float): Rotation angle applied at each step (
840             radians) by which to increment the plane's orientation.
841
842         rotation_axis (list): Specifies the axes about which the planes
843             rotate at each step.
844
845         secondary_angle (list): Contains list of angles (theta / phi,
846             where ID = 2/1) from the polar coordinates of each position.
847
848         sequence_ID (int): Indicates type of movement sequence. (2 =
849             horizontal circles, or 1 = vertical circles)
850
851     Returns:
852         rotated_planes (list): Transformed plane objects at each step.
853         fig (Plotly figure): Updated visualization.
854     """

```

```

841     rotated_planes = []
842
843
844     R_p = None
845
846     # exit(2)
847     for idx, position in enumerate(all_positions):
848         logging.info(f"Performing arc movement {idx}")
849
850         if sequence_ID == 3:
851             logging.debug(f"Step {idx}: Rigid arc facing origin")
852
853             if idx == 0:
854                 new_plane = initialise_new_circle(start_plane, position)
855                 new_plane.title = f"Plane {idx} - Start"
856             else:
857                 new_plane = initialise_new_circle(rotated_planes[idx - 1], position)
858                 new_plane.title = f"Plane {idx}"
859
860                 # Calculate rotation to face origin
861                 R, angle = calculate_rotation_matrix(position, new_plane.direction)
862                 new_plane.rotate_plane(R)
863                 rotation_angle_deg = np.degrees(angle)
864                 logging.debug(f"Plane {idx} rotated {np.round(np.degrees(angle), 2)}degree to face origin")
865
866                 rotated_planes.append(new_plane)
867                 new_plane.print_pose()
868                 continue # Skip other movements
869
870             movement_type = determine_movement_type(idx, 1, secondary_angle)
871
872             logging.debug(f"Movement type: {movement_type}")
873             # Movement types:
874             # 1 = First position
875             # 2 = Horizontal circles
876             # 3 = Same meridian
877             # 4 = Different meridian
878             logging.debug(f"Current secondary angle: {np.round(np.degrees(secondary_angle[idx])), 2)}degree")
879
880             if movement_type is None:
881                 logging.error(f"Error: Unable to determine movement type for position {idx}")
882                 return None

```

```

883
884     elif movement_type == 1:
885
886         logging.debug(f"Step {idx}: First position")
887
888         # Create copy of original plane and translate it to new
889         # position
890         new_plane = initialise_new_circle(start_plane, position)
891         new_plane.title = f"Plane {idx} - New Circle"
892
893         # Set primary rotation matrix
894         R_p = do_rotation(primary_angle, rotation_axis[0])
895
896         logging.debug(
897             f"Primary rotation matrix set by {np.round(np.degrees(
898                 primary_angle), 2)}degree / {np.round(primary_angle, 2)} rad about {
899                 rotation_axis[0]}-axis")
900         logging.debug(f"Primary matrix: \n{np.round(R_p, 2)} \n")
901
902
903
904     elif movement_type == 2:
905
906         logging.debug(f"Step {idx}: Different meridian")
907
908         # Create copy of original plane and translate it to new
909         # position
910         new_plane = initialise_new_circle(start_plane, position)
911         new_plane.title = f"Plane {idx} - New Circle"
912
913         secondary_rotation_angle = secondary_angle[idx]
914         logging.debug(
915             f"Current secondary angle {np.round(np.degrees(
916                 secondary_angle[idx]), 2)}degree previous {np.round(np.degrees(
917                 secondary_angle[idx - 1]), 2)}degree")
918
919         # Get secondary rotation matrix
920         R_s = do_rotation(secondary_rotation_angle, rotation_axis
921                         [1])
922
923         # Apply secondary rotation matrix
924         new_plane.rotate_plane(R_s)
925
926
927         logging.debug(
928             f"Secondary rotation {np.round(np.degrees(
929                 secondary_rotation_angle), 2)}degree around {rotation_axis[1]}-axis
930                 applied")

```

```

921         # Generate primary rotation matrix
922         R_p, required_angle, new_plane = primary_rotation_handling(
923             sequence_ID, new_plane, all_positions[idx + 1],
924             primary_angle, rotation_axis)
925             primary_angle = required_angle
926
927             logging.debug(f"Primary matrix: \n{np.round(R_p, 2)} \n")
928
929     else:
930
931         logging.debug(f"Step {idx}: Same meridian")
932
933         # Create copy of previous plane and translate it to new
934         position
935         new_plane = initialise_new_circle(rotated_planes[idx - 1],
936         position)
937         new_plane.title = f"Plane {idx}"
938
939         logging.debug(f"Plane {idx} initial direction: {np.round(
940         new_plane.direction, 2)}")
941
942         # Apply rotation
943         new_plane.rotate_plane(R_p)
944         logging.debug(f"Plane {idx}: Applying primary rotation {np.
945         round(np.degrees(primary_angle), 2)}degree")
946         logging.debug(f"Plane {idx} rotated direction: {np.round(
947         new_plane.direction, 2)}")
948
949         # Append new plane from ANY above ~~~
950         rotated_planes.append(new_plane)
951
952         new_plane.print_pose()
953
954     return rotated_planes
955
956
957 def visualise_intersections(fig, lines):
958     """
959
960     Checks intersection results, and adds to plot to indicate results -
961     hits and misses in green and red.
962
963     Args:
964         fig: The graphic object to update.
965         lines: List of Line objects.
966
967     Returns:

```

```

960         Updated Plotly figure
961
962     """
963     for line in lines:
964         if line.result == 1: # Hit
965             fig = line.plot_lines_3d(fig, "green")
966         else: # Miss
967             fig = line.plot_lines_3d(fig, "red")
968
969     return fig
970
971 def visualise_intersections_seq(line):
972     """
973     Checks intersection results, and adds to plot to indicate results -
974     hits and misses in green and red.
975
976     Args:
977         line: List of Line objects.
978
979     Returns:
980         Updated Plotly figure
981     """
982
983     if line.result == 1:
984         color = 'green'
985     else:
986         color = 'red'
987
988     x = [line.position[0], line.intersection_coordinates[0]]
989     y = [line.position[1], line.intersection_coordinates[1]]
990     z = [line.position[2], line.intersection_coordinates[2]]
991
992     scatter_obj = go.Scatter3d(
993         x=x, y=y, z=z,
994         mode='lines',
995         showlegend=False,
996         line={'color': color, 'width': 3}
997     )
998
999
1000 def check_fig_data(fig):
1001     logging.debug("\n")
1002     logging.debug(f"Number of traces: {len(fig.data)}")
1003     for idx, trace in enumerate(fig.data):
1004         logging.debug(

```

```

1006             f"Trace {idx}: Type = {type(trace)}, Name = {trace.name if
1007             hasattr(trace, 'name') else 'Unnamed'})")
1008
1009 def prepare_line_samples(lines, line_list, sample_size):
1010     """
1011         Samples list of lines for visualisation
1012         Returns graphic objects of random lines
1013
1014     :arg:
1015         lines: List of Line objects
1016         line_list: line_id list
1017         sample_size: Number of lines to be selected
1018     :return:
1019     """
1020
1021     # exit(2)
1022
1023     lines_graphics = []
1024     # Validate sample size
1025
1026     if line_list is None or len(line_list) == 0:
1027         logging.debug("No lines to sample.")
1028         return None
1029
1030     logging.debug(f"Sample size: {sample_size}")
1031     logging.debug(f"Number of lines: {len(line_list)}")
1032
1033     if sample_size > len(line_list):
1034         logging.warning(f" Sample size {sample_size} is larger than
1035             number of lines {len(line_list)}.")
1036         sample_size = len(line_list)
1037
1038     # Extract samples
1039     sampled_lines = random.sample(line_list, sample_size)
1040
1041     for samples in sampled_lines:
1042         lines_graphics.append(visualise_intersections_seq(lines[samples]))
1043             # Stores line objects
1044
1045     logging.debug(f"Returning {len(lines_graphics)} lines for
1046         visualisation.")
1047     return lines_graphics
1048
1049
1050 def rigid_arc_rotation(radius, arc_resolution_deg, tilt_angles):
1051     """

```

```

1049     Generates 3D coordinates along a semicircular arc in the x-z plane,
1050     then applies a series of rotations about the x-axis using the
1051     provided tilt angles.
1052
1053     Args:
1054         radius (float): Radius of the arc.
1055         arc_resolution_deg (float): Angle increment for arc sampling.
1056         tilt_angles (list or array): List of angles to rotate arc about
1057             x-axis.
1058
1059     Returns:
1060         np.ndarray: Stacked array of all rotated arc positions in
1061             Cartesian coordinates (shape: [N_total, 3])
1062
1063     """
1064     # Generate arc angles from 0 to 180 degrees (semi-circle)
1065     arc_angles = np.arange(0, 180 + arc_resolution_deg,
1066                           arc_resolution_deg)
1067     theta_arc = np.radians(arc_angles)
1068
1069     # Arc in x-z plane
1070     x = radius * np.cos(theta_arc)
1071     y = np.zeros_like(x)
1072     z = radius * np.sin(theta_arc)
1073
1074     arc_points = np.vstack((x, y, z))  # shape: [3, N]
1075
1076     all_rotated_positions = []
1077
1078     with open("../data/rigid_arc_angles.csv", "w", newline="") as f:
1079         writer = csv.writer(f)
1080         writer.writerow(["tilt_angle_deg", "arc_angle_deg"])  # Header
1081         row
1082
1083         for tilt_angle_deg in tilt_angles:
1084             # Rotation matrix about x-axis
1085             tilt_rad = np.radians(tilt_angle_deg)
1086             R_x = np.array([
1087                 [1, 0, 0],
1088                 [0, np.cos(tilt_rad), -np.sin(tilt_rad)],
1089                 [0, np.sin(tilt_rad), np.cos(tilt_rad)]
1090             ])
1091
1092             for arc_angle in arc_angles:
1093                 with open("../data/rigid_arc_angles.csv", "a", newline="")
1094                 as f:
1095                     writer = csv.writer(f)
1096                     writer.writerow([tilt_angle_deg, arc_angle])

```

```

1090
1091     # Apply rotation
1092     rotated_arc = R_x @ arc_points  # shape: [3, N]
1093     all_rotated_positions.append(rotated_arc.T)  # shape: [N, 3]
1094
1095     logging.debug(f"Rotated arc size: {np.shape(rotated_arc.T)}")
1096
1097     logging.debug(f"Rotated arc: {np.shape(all_rotated_positions)})")
1098
1099
1100
1101     return np.vstack(all_rotated_positions)  # shape: [N_total, 3]
1102
1103
1104 def log_parameters(primary_angle, secondary_angle, rotation_step,
1105                     rotation_axis, sequence_ID):
1106     # Display simulation parameters
1107     logging.debug(f"sequence ID: {sequence_ID}")
1108     logging.debug(f"Primary angle: {primary_angle}")
1109     logging.debug(f"Secondary angle: {secondary_angle}")
1110     logging.debug(f"Rotation step: {np.round(np.degrees(rotation_step),
1111                                         2)}")
1112     logging.debug(f"Rotation axis: {rotation_axis}")
1113
1114
1115 def get_horizontal_params(config):
1116     """
1117         Returns the parameters required to set up horizontal circle style of
1118         movements
1119         Returns:
1120         """
1121     logging.info("Horizontal circles movement")
1122
1123     # Phi goes from 90 down to input arc_phi step
1124     arc_phi_angle = np.arange(
1125         90,
1126         -config.arc_movement["horizontal_secondary"],
1127         -config.arc_movement["horizontal_secondary"]))  # Secondary
1128         rotation (between meridians)
1129
1130     arc_theta_angle = config.arc_movement["horizontal_primary"]  # Primary
1131         rotation (rotate on same meridian)
1132
1133     sequence_ID = 2  # 2 for horizontal circles movement
1134
1135     rotation_axis = ["z", "y"]
1136     rotation_step = np.radians(arc_theta_angle)  # Primary rotation (

```

```

        rotate on same meridian)

1132
1133     log_parameters(arc_theta_angle, config.arc_movement[""
1134         horizontal_secondary"], rotation_step, rotation_axis,
1135             sequence_ID)
1136
1137
1138
1139 def get_vertical_params(config):
1140     """
1141         Returns the parameters for vertical circle movement.
1142     """
1143     logging.info("Vertical circles movement")
1144
1145     arc_phi_angle = config.arc_movement["vertical_primary"] # Primary
1146     rotation (rotate on same meridian)
1147     arc_theta_angle = config.arc_movement["vertical_secondary"] # Secondary
1148     rotation (rotate between meridians)
1149
1150     sequence_ID = 1 # 1 for vertical circles movement
1151
1152     rotation_axis = ["y", "z"]
1153     rotation_step = np.radians(-arc_phi_angle) # Primary rotation (
1154     rotate on same meridian)
1155
1156     log_parameters(arc_phi_angle, arc_theta_angle, rotation_step,
1157         rotation_axis, sequence_ID)
1158
1159     return arc_phi_angle, arc_theta_angle, sequence_ID, rotation_axis,
1160         rotation_step
1161
1162
1163 def get_rigid_params(config):
1164     """
1165         Returns the parameters for rigid arc movement.
1166         Also, directly computes the 'rigid_arc_positions' array.
1167     """
1168     logging.info("Rigid Arc circles movement")
1169
1170     rigid_arc_step = config.arc_movement["rigid_arc_step"]
1171
1172     tilt_angles = config.arc_movement["tilt_angles"]
1173
1174     sequence_ID = 3 # 3 for rigid arc movement

```

```

1171     rotation_axis = ["y", "z"]
1172     rotation_step = np.radians(-10)
1173
1174     # Generate positions for rotation in rigid arc
1175     rigid_arc_positions = rigid_arc_rotation(config.arc_movement["radius"]
1176                                             , rigid_arc_step, tilt_angles)
1177
1178     logging.debug(f"Rigid arc positions shape: {np.shape(
1179         rigid_arc_positions)}")
1180
1181
1182 # @profile(stream=open("memory_profile.log", "w"))
1183 def main(config, sim_idx=0, num_lines=None):
1184     if num_lines is None:
1185         num_lines = config.simulation["num_lines"]
1186     """
1187     Runs the main program
1188     1. Initialises planes and areas.
1189     2. Creates lines from the source plane.
1190     3. Sets up a 3D plot and visualises the environment, including
1191     planes and areas.
1192     4. Applies rotation to the source plane and updates the
1193     visualization.
1194     5. Rotates the lines according to the transformed source plane.
1195     6. Evaluates intersections between lines and the sensor plane,
1196     visualises results, and calculates hit/miss information.
1197     7. Displays the final 3D plot and prints the hit/miss results.
1198     """
1199
1200     results_path = config.debugging["data_csv_path"]
1201     # num_lines = config.simulation["num_lines"]
1202
1203     # ----- Step 1: Initialize planes and areas ----- #
1204     sensorPlane, sourcePlane, aperturePlane, sensorAreas, aperture_areas
1205     = initialise_planes_and_areas(config)
1206
1207     # ----- Step 2: Create lines from source plane ----- #
1208     lines = create_lines_from_plane(sourcePlane, num_lines)
1209
1210     # ----- Step 3: Create 3D plot and visualize environment ----- #
1211     fig = initialise_3d_plot(sensorPlane) # Applies plot formatting and
1212     global axes
1213
1214     # Apply visualization settings from config

```

```

1210     if config.visualization["show_sensor_plane"]:
1211         fig = visualise_environment(fig, sensorPlane, config.
1212                                     visualization["color_sensor_plane"])
1213     if config.visualization["show_source_plane"]:
1214         fig = visualise_environment(fig, sourcePlane, config.
1215                                     visualization["color_source_plane"])
1216     if config.visualization["show_aperture_plane"]:
1217         fig = visualise_environment(fig, aperturePlane, config.
1218                                     visualization["color_aperture_plane"])
1219     if config.visualization["show_sensor_area"]:
1220         for sensor in sensorAreas: # Display all defined sensors on the
1221             plot
1222             fig = visualise_environment(fig, sensor, config.
1223                                         visualization["color_sensor_area"])
1224     if config.visualization["show_aperture_area"]:
1225         for aperture in aperture_areas: # Display all defined apertures
1226             on the plot
1227             fig = visualise_environment(fig, aperture, config.
1228                                         visualization["color_aperture_area"])
1229
1230     sensorPlane.title = "Parent axis"
1231     sensorPlane.print_pose()
1232
1233     sourcePlane.title = "Source plane"
1234     sourcePlane.print_pose()
1235
1236     # ----- Step 4: Arc movements ----- #
1237     # -- Phase 1: Compute arc steps -- #
1238     # Set identify which type of movement
1239     # Prepare position P vectors throughout movement
1240
1241     # Movement style booleans
1242     horizontal_circles = config.arc_movement["horizontal_circles"]
1243     vertical_circles = config.arc_movement["vertical_circles"]
1244     rigid_arc = config.arc_movement["rigid_arc"]
1245
1246     # Initialise variables
1247     arc_phi_angle = None
1248     arc_theta_angle = None
1249     sequence_id = None
1250     rotation_axis = None
1251     rotation_step = 0.0
1252     rigid_positions = None
1253
1254     # Read from JSON config, set up for style of movement
1255     if horizontal_circles:
1256         (arc_phi_angle,

```

```

1250         arc_theta_angle,
1251         sequence_ID,
1252         rotation_axis,
1253         rotation_step) = get_horizontal_params(config)
1254
1255     # Generate movement sequence
1256     all_positions, secondary_movement = rotation_rings(
1257         sequence_ID,
1258         config.arc_movement["radius"], # Radius of arc movement
1259         arc_theta_angle, # steps of theta taken around the arc
1260         arc_phi_angle # phi levels to the spherical arc
1261     )
1262     # Correct form of secondary movement for horizontal circles
1263     reference_angle = secondary_movement[0]
1264     secondary_movement -= reference_angle # sets the secondary
angle to the starting position (not abs)
1265
1266 elif vertical_circles:
1267     (arc_phi_angle,
1268      arc_theta_angle,
1269      sequence_ID,
1270      rotation_axis,
1271      rotation_step) = get_vertical_params(config)
1272
1273     # Generate movement sequence
1274     all_positions, secondary_movement = rotation_rings(
1275         sequence_ID,
1276         config.arc_movement["radius"], # Radius of arc movement
1277         arc_theta_angle, # steps of theta taken around the arc
1278         arc_phi_angle # phi levels to the spherical arc
1279     )
1280
1281 elif rigid_arc:
1282     (arc_phi_angle,
1283      arc_theta_angle,
1284      sequence_ID,
1285      rotation_axis,
1286      rotation_step,
1287      all_positions) = get_rigid_params(config)
1288
1289     secondary_movement = np.zeros(len(all_positions)) # not needed
1290
1291 else:
1292     logging.warning("No movement")
1293     exit(3)
1294
1295 # -- Phase 2: Move to first arc position -- #

```

```

1296     start_pose_plane = setup_initial_pose(
1297         sourcePlane,
1298         config.arc_movement["initial_rotation"],
1299         config.arc_movement["rotation_axis"],
1300         all_positions
1301     )
1302
1303 # -- Phase 3: Apply the plane along the arc -- #
1304 # Move plane along arc and update lines
1305 if config.arc_movement["execute_movements"]:
1306     rotated_planes = move_plane_along_arc(
1307         start_pose_plane,
1308         all_positions,
1309         rotation_step,
1310         rotation_axis,
1311         secondary_movement,
1312         sequence_ID
1313     )
1314 else:
1315     rotated_planes = [start_pose_plane]
1316
1317 # #      ----- Step 6: Evaluate hits and visualize lines -----
1318 #
1319 logging.info(f"\n\nChecking intersections:\n")
1320 # check_fig_data(fig)
1321 line_scatter_objects = []
1322 results = np.zeros((len(rotated_planes), 2))
1323
1324 for idx, plane in enumerate(rotated_planes): # Check lines for each
1325     plane
1326     update_lines_global_positions(lines, plane)
1327     logging.info(f"{plane.title}")
1328     hit, miss, hit_list, miss_list = evaluate_line_results(
1329         sensorPlane, sensorAreas, aperturePlane, aperture_areas,
1330                                         lines)
1331     handle_results(sensorAreas, sim_idx, idx, config)
1332     logging.debug(f"{plane.title} has {hit} hits and {miss} misses")
1333
1334     results[idx, 0] = hit
1335     results[idx, 1] = miss
1336
1337     with open("../data/results.csv", "a") as results_file:
1338         results_file.write(
1339             f"{sim_idx},{idx}, {results[idx, 0]}, {results[idx,
1340             1]},{num_lines}, {config.output["Sim_title"]}\n")
1341
1342     # Sample lines for visualisation

```

```

1339     logging.debug(f"Selecting hits for visualisation for plane {idx}")
1340     lines_for_plane = []
1341
1342     hits_visualised = (prepare_line_samples(lines, hit_list, config.visualization["hits_to_display"]))
1343
1344     if hits_visualised is not None:
1345         lines_for_plane.extend(hits_visualised)
1346
1347     logging.debug(f"Selecting misses for visualisation for plane {idx}")
1348     misses_visualised = (prepare_line_samples(lines, miss_list, config.visualization["misses_to_display"]))
1349
1350     if misses_visualised is not None:
1351         lines_for_plane.extend(misses_visualised)
1352
1353     # Stores line objects for all planes
1354     line_scatter_objects.append(lines_for_plane)
1355     logging.debug(f"Line scatter objects: {type(line_scatter_objects)} length {len(line_scatter_objects)}")
1356
1357     # ----- Step 7: Display the plot and results -----
1358     # Show any plot
1359     if config.visualization["show_output_parent"]:
1360
1361         # show animation or static plot
1362         if config.visualization["animated_plot"]:
1363             fig = generate_arc_animation(fig, rotated_planes,
1364                                         line_scatter_objects, results)
1365             logging.info("Animated plot generated.")
1366
1367             # create_gif_from_frames()
1368             # crop_gif_center(crop_width=1000, crop_height=800)
1369             pio.write_html(fig, file="../output/arc_animation.html",
1370                           auto_open=True)
1371
1372         else:
1373
1374             fig = generate_static_arc_plot(config, fig, rotated_planes,
1375                                         line_scatter_objects)
1376             logging.info("Static plot generated.")
1377
1378             if config.output["save_static_png"]:
1379                 crop_image_center("static_plot.png", "static_plot_cropped.png",
1380                                   crop_width=1000, crop_height=800)

```

```

1377         fig.show()
1378     else:
1379         logging.info("Visualization disabled (show_output_parent = false
1380 ).")
1381
1382     print("\nFinished.\n")
1383
1384 #
1385 #
1386 # if __name__ == "__main__":
1387 #     main()
1388
1389
1390 @profile(stream=open("../memory_profile.log", "w"))
1391 def run_all_test(config, num_lines):
1392     for i in range(config.simulation["num_runs"]):
1393         logging.info(f"\n--- Simulation Run {i + 1} ---")
1394
1395         sim_idx = prepare_output(config.debugging["data_csv_path"])
1396         start_time = time.time()
1397         main(config, sim_idx, num_lines)
1398         end_time = time.time()
1399         runtime = end_time - start_time
1400
1401         with open("../data/results.csv", "r") as f:
1402             lines = f.readlines()
1403
1404         updated_lines = []
1405         for line in lines:
1406             if line.startswith(f"{sim_idx},"):
1407                 line = line.strip() + f",{runtime:.4f}\n"
1408             else:
1409                 line = line if line.endswith("\n") else line + "\n"
1410             updated_lines.append(line)
1411
1412         with open("../data/results.csv", "w") as f:
1413             f.writelines(updated_lines)
1414
1415
1416 if __name__ == "__main__":
1417     from config import Config
1418
1419     config = Config(file_path="C:/Users/temp/IdeaProjects/MENGProject/
config.json")
1420     # config = Config(file_path="C:/Users/temp/IdeaProjects/MENGProject/
test_configs/test_directly_below.json")

```

```

1421
1422     line_tests = [10000]
1423
1424     for num_lines in line_tests:
1425         logging.info(f"Testing {num_lines} lines")
1426         run_all_test(config, num_lines)

```

H.2. plane.py

```

1 import logging
2
3 import numpy as np
4 from matplotlib import pyplot as plt
5 import plotly.graph_objects as go # For 3D visualization
6
7 class Plane:
8     """
9         A class to represent a 3D plane with plotting and point generation
10        methods.
11
12        Attributes:
13            title (str): The name of the plane.
14            position (np.array): The position of the plane in 3D space.
15            direction (np.array): The direction vector of the plane.
16            width (float): The width of the plane.
17            length (float): The length of the plane.
18            corners (np.array): The 3D coordinates of the plane's corners.
19        """
20
21        def __init__(self, title, position, direction, width, length):
22            """
23                Initialize a Plane instance.
24
25                Args:
26                    title (str): The name of the plane.
27                    position (list or array): The center position of the plane in 3D
28                    space [x, y, z].
29                    direction (list or array): The direction vector of the plane [e.
30                    g. normal vector].
31                    width (float): The width of the plane.
32                    length (float): The length of the plane.
33
34            self.corners = None
35            self.title = title
36            self.position = np.array(position)
37            self.direction = np.array(direction)

```

```

36
37     self.width = width
38     self.length = length
39
40     # Compute local reference frame (right, up, normal)
41     self.right, self.up, self.direction = compute_local_axes(self.
direction)
42
43     # Compute initial corners using the local frame
44     self.update_corners()
45
46     # Definable colours
47     self.colour = None
48     # print(f"\n {self.title} Corners after initial definition: {self.corners}")
49
50     def update_corners(self):
51         """
52             Recalculate the plane's corner positions using its local
coordinate system.
53         """
54
55         half_width = self.width / 2
56         half_length = self.length / 2
57         # Compute corners relative to the center using the local basis
vectors
58
59         self.corners = np.array([
60             self.position + (-half_width * self.right + half_length * self.up),  # Top Left
61             self.position + (half_width * self.right + half_length * self.up),  # Top Right
62             self.position + (half_width * self.right - half_length * self.up),  # Bottom Right
63             self.position + (-half_width * self.right - half_length * self.up)  # Bottom Left
64         ])
65
66     def rotate_plane(self, rotation_matrix):
67         """
68             Rotates the plane by applying a rotation matrix to its local
coordinate system.
69
70             Args:
71                 rotation_matrix (np.array): 3x3 rotation matrix.
72
73                 # Rotate local axes
74                 self.right = np.dot(rotation_matrix, self.right)

```

```

74         self.up = np.dot(rotation_matrix, self.up)
75         self.direction = np.dot(rotation_matrix, self.direction)
76
77     # Recalculate corners after rotating the local frame
78     self.update_corners()
79     # print(f"\n {self.title} Corners after rotation: {self.corners
80     }")
81
82     # Compute local reference frame (right, up, normal)
83     # self.right, self.up, self.direction = compute_local_axes(self.
84     direction)
85     # Verify consistency
86     computed_normal = np.cross(self.right, self.up)
87     if not np.allclose(computed_normal, self.direction, atol=1e-6):
88         raise ValueError(
89             f"Right-Hand Rule Violated after rotation! Computed
normal {computed_normal} does not match expected normal {self.
direction}")
90
91
92     def translate_plane(self, translation_vector):
93         self.position = self.position + translation_vector
94
95         self.update_corners()
96         # print(f"\n {self.title} Corners after translation: {self.
97         corners}")
98
99
100    def planes_plot_3d(self, fig, colour):
101        """
102        Add a 3D representation of the plane to a Plotly figure.
103
104        Args:
105            fig (plotly.graph_objects.Figure): The figure object to
which the plane will be added.
106            colour (str): The color for the plane surface.
107
108        Returns:
109            plotly.graph_objects.Figure: Updated figure object.
110        """
111
112        # Extract corner coordinates
113        x, y, z = zip(*self.corners)
114
115
116        # Create a surface plot using the four corners
117        fig.add_trace(go.Mesh3d(
118            x=x, y=y, z=z,
119            i=[0, 0, 0, 1], # Triangle 1 (0-1-2) and Triangle 2 (0-2-3)
120            j=[1, 2, 3, 2],
121            k=[2, 3, 0, 3],

```

```

115         color=colour,
116         opacity=0.5,
117         name=f"{self.title} (Mesh)",
118         showlegend=False
119     ))
120
121     #      # Add a dummy trace for the legend
122     # fig.add_trace(go.Scatter3d(
123     #      x=[x[0]], y=[y[0]], z=[z[0]], # Single point (dummy)
124     #      mode='markers',
125     #      marker=dict(size=5, color=colour, opacity=0.5),
126     #      name=f"{self.title} (Trace)", # Legend entry
127     #      showlegend=False
128     #  ))
129
130     # # Add labels at the corner points
131     # for i, (xi, yi, zi) in enumerate(self.corners):
132     #     fig.add_trace(go.Scatter3d(
133     #         x=[xi], y=[yi], z=[zi],
134     #         mode='text',
135     #         text=f'P{i}', # Label each corner as P0, P1, P2, etc.
136     #         textposition='top center',
137     #         showlegend=False,
138     #         name=f"{self.title} (corner {i})"
139     #     ))
140
141     # print(f"Plane : {self.title}")
142     # print(f"Translated position: {self.position}")
143     # print(f"Translated corners: {self.corners}")
144
145     return fig
146
147 def plot_area(self):
148     """
149         Plot the 2D representation of the plane's area using matplotlib.
150     """
151     for i in range(len(self.corners)):
152         # Plot the plane corners
153         plt.plot(self.corners[i][0], self.corners[i][1], marker='*', color='green')
154
155         # Connect consecutive corners to form edges
156         if i < len(self.corners) - 1:
157             x_positions = np.linspace(self.corners[i][0], self.
158 corners[i + 1][0], 5)
159             y_positions = np.linspace(self.corners[i][1], self.
160 corners[i + 1][1], 5)

```

```

159         else: # Connect the last corner to the first corner
160             x_positions = np.linspace(self.corners[i][0], self.
corners[0][0], 5)
161             y_positions = np.linspace(self.corners[i][1], self.
corners[0][1], 5)
162
163             plt.plot(x_positions, y_positions, color='green') # Plot
edge lines
164
165             plt.title(f"Area of {self.title}")
166
167     def random_points(self, quantity):
168         """
169             Generate random points within the plane boundaries.
170
171         Args:
172             quantity (int): The number of random points to generate.
173
174         Returns:
175             np.stack: Returns (N,2) array of random (x, y) points on the
plane.
176         """
177
178         x = np.random.uniform(-self.width / 2, self.width / 2, quantity)
179         y = np.random.uniform(-self.length / 2, self.length / 2,
quantity)
180         return np.vstack((x, y)).T # Returns an (N,2) array
181
182     def plot_points(self, point):
183         """
184             Plot a single point on the 2D plane using matplotlib.
185
186         Args:
187             point (list or array): The (x, y) coordinates of the point
to plot.
188         """
189
190         plt.plot(point[0], point[1], marker='*', color='red')
191
192     def plot_axis(self, fig):
193         local_axis = np.array([self.right, self.up, self.direction])
194
195         local_axis_colours = ['red', 'green', 'blue']
196         local_axis_names = ['Right (x)', 'Up (y)', 'Normal (z)']
197
198         for i in range(3):

```

```

200         unit_vector = local_axis[i] / np.linalg.norm(local_axis[i])
201     # Normalize
202     start = self.position # Origin of local axes at plane's
203     position
204     end = self.position + unit_vector # Unit length
205
206     207     fig.add_trace(go.Scatter3d(
208         x=[start[0], end[0]],
209         y=[start[1], end[1]],
210         z=[start[2], end[2]],
211         mode='lines+markers',
212         line=dict(color=local_axis_colours[i], width=4),
213         marker=dict(size=3, color=local_axis_colours[i], opacity
214 =0.8),
215         name=local_axis_names[i],
216         showlegend=False
217     ))
218
219     220     return fig
221
222
223     224     def print_pose(self):
225         # Format the numpy arrays properly for printing
226         right_str = ", ".join(f"{x:.2f}" for x in self.right)
227         up_str = ", ".join(f"{x:.2f}" for x in self.up)
228         direction_str = ", ".join(f"{x:.2f}" for x in self.direction)
229
230         position_str = ", ".join(f"{x:.2f}" for x in self.position)
231
232         logging.debug(f"{self.title} --- \n position: [{position_str}]\n
233             right(red) (x): [{right_str}]\n up(green) (y): [{up_str}]\n
234             normal(blue) (z): [{direction_str}]\n")
235
236         # corners_str = ", ".join(
237         #     f"[{', '.join(f'{val:.2f}' for val in corner)}]" for
238         corner in self.corners)
239         # print(f"Corners: {corners_str}")
240
241
242     243     def compute_local_axes(normal):
244         """
245             Compute a local coordinate system (right, up, normal) based on a
246             given normal vector.
247
248             Args:
249                 normal (np.array): The plane's normal vector.
250
251             Returns:

```

```

240     right (np.array): Right vector (width direction).
241     up (np.array): Up vector (length direction).
242     normal (np.array): Normalized normal vector.
243     """
244
245     normal = normal / np.linalg.norm(normal) # Ensure it's a unit
246     vector
247
248     # Defines the parent / world up vector (y)
249     world_up = np.array([0, 1, 0])
250
251     # If normal is parallel to world_up, use a different reference
252     if np.abs(np.dot(normal, world_up)) > 0.99: # Nearly parallel case
253         world_up = np.array([1, 0, 0]) # Switch to an alternative
254         reference
255
256     # Compute right vector (cross product of world_up and normal)
257     right = np.cross(world_up, normal)
258     right /= np.linalg.norm(right) # Normalize to ensure unit length
259
260     # Compute up vector (cross product of normal and right)
261     up = np.cross(normal, right)
262
263     # Ensure normalization
264     up /= np.linalg.norm(up)
265
266     computed_normal = np.cross(right, up)
267     if not np.allclose(computed_normal, normal, atol=1e-6):
268         raise ValueError(f"Right-Hand Rule Violated! Computed normal {computed_normal} does not match expected normal {normal}")
269
270     # print(f"Right: {right} Up: {up} Normal: {normal}")
271     return right, up, normal # Return orthonormal basis

```

H.3. line.py

```

1 import logging
2
3 import numpy as np
4 import plotly.graph_objects as go
5
6 class Line:
7     def __init__(self, local_position, direction, line_id):
8         """
9             Initializes a line with its position relative to a plane.
10
11         Args:
12             local_position (list): The position of the line in the plane

```

```

    's local coordinate system.

13         direction (array): The initial direction (same as plane's
normal).

14         """
15
16         self.line_id = line_id
17         self.local_position = np.array(local_position) # Stored in
local coordinates
18         self.position = None # Will be computed in global coordinates
19         self.direction = np.array(direction)

20
21         self.intersection_coordinates = None
22         self.result = None

23     def plot_lines_3d(self, fig, color):
24         """
25             Adds the line to the 3D plot.

26
27             Args:
28                 fig (Plotly figure): The figure to plot.
29                 color (str): Line color.

30
31             Returns:
32                 fig (Plotly figure): Updated figure.
33         """
34
35         if self.result == 1:
36             color = 'green'
37         else:
38             color = 'red'

39
40         x = [self.position[0], self.intersection_coordinates[0]]
41         y = [self.position[1], self.intersection_coordinates[1]]
42         z = [self.position[2], self.intersection_coordinates[2]]

43
44         fig.add_trace(go.Scatter3d(x=x, y=y, z=z, mode='lines',
showlegend=False,
45                                     line={'color': color, 'width': 3}))

46
47         return fig

48
49     def update_position(self, plane):
50         """
51             Updates the line's new position, after rotation of the plane
52
53             :return: global coordinate of line starting point
54         """
55
56         self.position = plane.position + np.dot(self.local_position, np.

```

```

    vstack((plane.right, plane.up, plane.direction)))
56
57     self.position = np.add(plane.position, np.add(self.position[0] *
58     plane.right, self.position[1] * plane.up, self.position[2] * plane.
59     direction))
60
61     # self.direction = plane.direction
62     # print(f"New position = {self.position}")
63
64     def update_global_position(self, plane):
65         """
66
67             Updates the global position of the line based on the transformed
68             plane.
69
70             Args:
71                 plane (Plane): The updated plane after movement.
72
73             # logging.debug(f"Line has local position {self.local_position}
74             and direction {self.direction}")
75
76             # if self.position is None:
77             #     logging.debug("Global position not defined yet.")
78             # else:
79             #     logging.debug(f"Current global position: {self.position}")
80
81
82             self.position = plane.position + (self.local_position[0] * plane
83             .right +
84                         self.local_position[1] * plane
85             .up +
86                         self.local_position[2] * plane
87             .direction)
88
89             self.direction = plane.direction
90
91
92             # logging.debug(f"New global position: {self.position}")
93
94     def print_info(self):
95
96
97         logging.debug(f"Line {self.line_id} Position = {self.
98         local_position[0]}, {self.local_position[1]}, {self.local_position
99         [2]}")
100
101         logging.debug(f"Line {self.line_id} Direction = {self.direction}
102         ")
103
104         logging.debug(f"Line {self.line_id} Result = {self.result}")
105         logging.debug(f"Line {self.line_id} Intersection coordinates = {
106         self.intersection_coordinates}")
107
108         logging.debug(f"Line {self.line_id} Global position = {self.
109         position}")

```

H.4. intersectionCalculations.py

```

1 from line import Line
2 from plane import Plane
3 import numpy as np
4
5 def direction_vectors(array1, array2):
6     check_direction = np.multiply(array1, array2)
7     check_direction = np.sum(check_direction)
8
9     return check_direction
10
11
12 def compute_t(nP, nA, nU):
13     return (nP - nA) / nU
14
15
16 def calculate_intersection(line, t):
17     x = line.direction[0] * t + line.position[0]
18     y = line.direction[1] * t + line.position[1]
19     z = line.direction[2] * t + line.position[2]
20
21     #coordinates = (x,y,z)
22     coordinates = np.array([x,y,z])
23     return coordinates
24
25 def intersection_wrapper(sensorPlane, line1):
26     nU = direction_vectors(sensorPlane.direction, line1.direction)
27
28     if nU != 0:
29         nA = np.dot(sensorPlane.direction, line1.position)
30         nP = np.dot(sensorPlane.direction, sensorPlane.position)
31
32         # print(f"nA = {nA}")
33         # print(f"nP = {nP}")
34         # print(f"nU = {nU}")
35
36         x = compute_t(nP, nA, nU)
37         IntersectionCoordinates = calculate_intersection(line1, x)
38
39         return IntersectionCoordinates
40     else:
41         # print(nU)
42         print(f"Intersection not possible for nU vector: {nU}")
43         # exit(1)
44         return None

```

H.5. areas.py

```

1 import numpy as np
2
3
4 from plane import compute_local_axes
5
6
7 class Areas:
8     def __init__(self, title, position, direction, width, length):
9
10         self.corners = None
11         self.title = title
12         self.position = np.array(position)
13         self.direction = np.array(direction)
14
15         self.width = width
16         self.length = length
17
18         # Compute local reference frame (right, up, normal)
19         self.right, self.up, self.normal = compute_local_axes(self.
direction)
20
21         # Stores the 'illumination' results
22         self.illumination = None
23
24         # Compute initial corners using the local frame
25         self.update_corners()
26
27         # print(self.corners)
28
29     def update_corners(self):
30         """
31             Recalculate the area's corner positions using its local
32             coordinate system.
33         """
34
35         half_width = self.width / 2
36         half_length = self.length / 2
37         # Compute corners relative to the center using the local basis
38         # vectors
39
40         self.corners = np.array([
41             self.position + (-half_width * self.right + half_length * self.up),  # Top Left
42             self.position + (half_width * self.right + half_length * self.up),  # Top Right
43             self.position + (half_width * self.right - half_length * self.up),  # Bottom Right
44             self.position + (-half_width * self.right - half_length * self.up)  # Bottom Left
45         ])

```

```

41     ])
42
43
44     ## Checking for intersection between area and intersection (with
45     # sensor plane) coordinates
46     def record_result(self, cords):
47         # True, if intersection x coordinate is within area boundary (x
48         # min and x max)
49         if (self.position[0] - (self.width / 2) <= cords[0] <= self.
50             position[0] + (self.width / 2)
51                 and # True, if intersection y coordinate is within area
52                 boundary (y min and y max)
53                     self.position[1] - (self.length / 2) <= cords[1] <= self.
54             position[1] + (self.length / 2)):
55                 return 1
56             else:
57                 return 0
58
59     def planes_plot_3d(self, fig, colour):
60         """
61             Add a 3D representation of the plane to a Plotly figure.
62
63             Args:
64                 fig (plotly.graph_objects.Figure): The figure object to
65                 which the plane will be added.
66                 colour (str): The color for the plane surface.
67
68             Returns:
69                 plotly.graph_objects.Figure: Updated figure object.
70
71         """
72         from plotly import graph_objects as go
73
74         # Extract corner coordinates
75         x, y, z = zip(*self.corners)
76
77         # Create a surface plot using the four corners
78         fig.add_trace(go.Mesh3d(
79             x=x, y=y, z=z,
80             color=colour,
81             opacity=0.5,
82             name=self.title
83         ))
84
85         # Add a dummy trace for the legend
86         fig.add_trace(go.Scatter3d(
87             x=[x[0]], y=[y[0]], z=[z[0]], # Single point (dummy)
88             mode='markers',
89

```

```

82         marker=dict(size=5, color=colour, opacity=0.5),
83         name=self.title, # Legend entry
84         showlegend=True
85     ))
86
87     # # Add labels at the corner points
88     # for i, (xi, yi, zi) in enumerate(self.corners):
89     #     fig.add_trace(go.Scatter3d(
90     #         x=[xi], y=[yi], z=[zi],
91     #         mode='text',
92     #         text=f'P{i}', # Label each corner as P0, P1, P2, etc.
93     #         textposition='top center',
94     #         showlegend=False,
95     #         name = f"{self.title} (corner {i})"
96     #     ))
97
98
99     return fig

```

H.6. config.py

```

1 import json
2 import logging
3
4 import numpy as np
5
6 class Config:
7     def __init__(self, file_path=None, data=None):
8         if data is not None:
9             self.load_from_dict(data)
10        elif file_path:
11            with open(file_path, "r") as f:
12                data = json.load(f)
13                self.load_from_dict(data)
14        else:
15            raise ValueError("Must provide either file_path or data")
16
17    def load_from_dict(self, data):
18        self.planes = data["planes"]
19        self.sensor_areas = data["sensor_areas"]
20        self.aperture_areas = data["aperture_areas"]
21        self.arc_movement = data["arc_movement"]
22        self.simulation = data["simulation"]
23        self.intersection = data["intersection"]
24        self.visualization = data["visualization"]
25        self.debugging = data["debugging"]
26        self.performance = data["performance"]

```

```

27     self.output = data["output"]
28
29     log_level = self.debugging.get("logging_level", "INFO").upper()
30     logging.basicConfig(level=getattr(logging, log_level, logging.
31     INFO))
32 config = Config(file_path="../config.json")

```

H.7. config.json

```

1  {
2      "planes": {
3          "source_plane": {
4              "position": [0, 0, 1],
5              "direction": [0,0,-1],
6              "width": 10,
7              "length": 10
8          },
9          "sensor_plane": {
10             "position": [0,0,0],
11             "direction": [0,0,-1],
12             "width": 10,
13             "length": 10
14         },
15         "aperture_plane": {
16             "position": [0,0,1],
17             "direction": [0,0,1],
18             "width": 5,
19             "length": 5
20         }
21     },
22     "sensor_areas": {
23         "sensor_A": {
24             "title": "A0",
25             "position": [0, -3.25, 0],
26             "direction": [0,0,1],
27             "width": 1.5,
28             "length": 1.5
29         },
30         "sensor_B": {
31             "title": "A1",
32             "position": [0, -1.25, 0],
33             "direction": [0,0,1],
34             "width": 1.5,
35             "length": 1.5
36         },
37         "sensor_C": {

```

```

38     "title": "A2",
39     "position": [-1, 2.25, 0],
40     "direction": [0,0,1],
41     "width": 1.5,
42     "length": 1.5
43   },
44   "sensor_D": {
45     "title": "A3",
46
47     "position": [1, 2.25, 0],
48     "direction": [0,0,1],
49     "width": 1.5,
50     "length": 1.5
51   }
52 },
53 "aperture_areas": {
54   "Aperture_A": {
55     "title": "Aperture_A",
56     "position": [0, -2.25, 1],
57     "direction": [0, 0, 1],
58     "width": 4,
59     "length": 2
60   },
61   "Aperture_B": {
62     "title": "Aperture_B",
63     "position": [0, 2.25, 1],
64     "direction": [0, 0, 1],
65     "width": 2,
66     "length": 4
67   }
68 },
69 "arc_movement": {
70   "execute_movements": true,
71   "initial_rotation": 90.0,
72   "rotation_axis": "y",
73   "radius": 10.0,
74   "horizontal_circles": false,
75   "horizontal_primary": 60,
76   "horizontal_secondary": 30,
77   "vertical_circles": false,
78   "vertical_primary": 10,
79   "vertical_secondary": [
80     0,
81     65,
82     130
83   ],
84   "rigid_arc": true,

```

```

85     "rigid_arc_step": 0.5,
86     "tilt_angles": [
87         0
88     ]
89 },
90   "simulation": {
91     "num_lines": 10000,
92     "num_runs": 1
93 },
94   "intersection": {
95     "max_distance": 100,
96     "strict_mode": true,
97     "tolerance": 0.01
98 },
99   "visualization": {
100     "show_output_parent": false,
101     "show_sensor_plane": true,
102     "show_source_plane": false,
103     "show_sensor_area": true,
104     "show_aperture_area": true,
105     "show_aperture_plane": false,
106     "color_sensor_plane": "red",
107     "color_source_plane": "yellow",
108     "color_aperture_plane": "green",
109     "color_sensor_area": "green",
110     "color_aperture_area": "black",
111     "show_axes": true,
112     "animated_plot": false,
113     "frame_rate": 500,
114     "hits_to_display": 2,
115     "misses_to_display": 2
116 },
117   "debugging": {
118     "enable_logging": true,
119     "logging_level": "INFO",
120     "data_csv_path": "data/results.csv"
121 },
122   "performance": {
123     "enable_memory_profiling": true,
124     "optimization_level": "basic"
125 },
126   "output": {
127     "save_static_png": false,
128     "save_animated_gif": false,
129     "Sim_title": "Basic"
130   }
131 }
```

H.8. arcRotation.py

```
1 import logging
2
3 import numpy as np
4
5
6 def convert_to_cartesian(rho, theta, phi):
7     """
8         Converts spherical coordinates (rho, theta, phi) back to Cartesian (
9             x, y, z).
10
11    Args:
12        rho (float): Radius (distance from origin).
13        theta (float): Azimuth angle in **radians**.
14        phi (float): Elevation angle in **radians**.
15
16    Returns:
17        np.array: Cartesian coordinates [x, y, z].
18    """
19
20    x = rho * np.sin(phi) * np.cos(theta)
21    y = rho * np.sin(phi) * np.sin(theta)
22    z = rho * np.cos(phi)
23
24    cartesian = np.array([x, y, z])
25
26    # print(f"X: {cartesian[0]:.2f}, Y: {cartesian[1]:.2f}, Z: {cartesian[2]:.2f}")
27
28
29 def arc_movement_coordinates(radius, theta_angle, phi_angle=90):
30     """
31         Takes input angle for arc rotation and returns cartesian coordinates
32         for each position.
33
34         :param:
35             theta_angle: Increment angle for arc rotation, angle by which
36             plane is rotated around global origin
37             radius: Radius of arc rotation
38
39         :returns:
40             cartesianCoords (np.array): Contains cartesian coordinates for
41             each position.
42             polarCoords (np.array): Contains polar coordinates for each
43             position.
```

```

40 """
41 # Predefine array to store output cartesian coords
42 cartesianCoords = []
43 polarCoords = []
44
45 print(f"Arc movement coordinates: radius = {radius}, theta = {theta_angle}, phi = {phi_angle}")
46
47 # Calculate the positions around the xy axis for arc rotation
48
49 thetas = np.arange(0, 360, theta_angle) # Generate values for theta
50 at each increment
51 print(f"Vary theta, at constant phi: angle_steps = {thetas}")
52 for idx, i in enumerate(thetas):
53     polar = [radius, np.radians(i),
54              np.radians(phi_angle)] # Each position around the arc
55 rotation, has coordinate defined in polar form
56
57 # Store polar coords
58 polarCoords.append(polar)
59 # Convert polar form to cartesian form
60 cartesian = convert_to_cartesian(polar[0], polar[1], polar[2])
61 cartesianCoords.append(cartesian)
62
63 print(f"Polar coordinates: {polarCoords}")
64
65 return cartesianCoords, polarCoords
66
67 def get_arc_coordinates(sequence_ID, radius, constant_angle,
68 stepping_angle):
69 """
70 - Takes radius, and two angles
71 - Makes a list of polar coordinates between 0 degrees and 180 / 360,
72 in steps of stepping_angle,
73     at constant radius and constant other input angle
74 - Converts polar coordinates to cartesian coordinates
75 - Returns cartesian & polar coordinates for each position
76 :param:
77     sequence_ID: 1 for varying theta, 2 for varying phi
78     radius
79     constant_angle
80     stepping_angle
81
82 :return:
83 """
84
85 # Holds the output angles
86 cartesianCoords = []

```

```

82 polarCoords = []
83
84 if sequence_ID == 2: # Vary theta, at constant phi: angle_steps = {angle_steps}
85
86     # Define angle steps
87     angle_steps = np.arange(0, 360, stepping_angle)
88     logging.debug(f"Angle steps: {angle_steps}")
89     print(f"Vary theta, at constant phi: angle_steps = {angle_steps}")
90
91     for steps in angle_steps:
92         # For varying theta: [radius, theta (varying), phi (constant)]
93         polar = [radius, np.radians(steps), np.radians(constant_angle)]
94
95         # Store polar coords
96         polarCoords.append(polar)
97
98         # Convert polar form to cartesian form
99         cartesianCoords.append(convert_to_cartesian(polar[0], polar[1], polar[2]))
100
101 if sequence_ID == 1: # Vary phi, at constant theta: angle_steps = {angle_steps}
102     angle_steps = np.arange(90, -90, -stepping_angle)
103
104     for steps in angle_steps:
105         # For varying phi: [radius, theta (constant), phi (varying)]
106         polar = [radius, np.radians(constant_angle), np.radians(steps)]
107
108         # Store polar coords
109         polarCoords.append(polar)
110
111         # Convert polar form to cartesian form
112         cartesianCoords.append(convert_to_cartesian(polar[0], polar[1], polar[2]))
113
114     return cartesianCoords, polarCoords
115
116
117 def rotation_rings(sequence_ID, radius, angle_theta, angle_phi):
118     """
119     Create a list of points around the arc rotation, and return
120     cartesian coordinates for each position.

```

```

120     Directly generate list of secondary angles for arc rotation, and
121     return cartesian coordinates for each position.
122     """
123
124     all_points = []
125     allPositions_polar = []
126     i = 0
127
128     if sequence_ID == 2: # Get variations of theta, for each value of
129         phi
130         logging.debug(f"Phi angles {angle_phi}")
131
132         for phi_angles in angle_phi:
133
134             # Returns coordinates around circle
135             list_point, list_points_polar = get_arc_coordinates(
136             sequence_ID, radius, phi_angles, angle_theta)
137
138             # Extend the all_points list with the new points
139             all_points.extend(list_point) # list_point is a list of
140             numpy arrays
141             allPositions_polar.extend(list_points_polar)
142
143             logging.debug(f"For phi angle {phi_angles}, generated {len(
144             list_point)} points")
145
146             i = i + 1
147
148             if sequence_ID == 1:
149                 for theta_angles in angle_theta:
150
151                     # Returns coordinates around circle
152                     list_point, list_points_polar = get_arc_coordinates(
153                     sequence_ID, radius, theta_angles, angle_phi)
154
155                     # Extend the all_points list with the new points
156                     all_points.extend(list_point) # list_point is a list of
157                     numpy arrays
158                     allPositions_polar.extend(list_points_polar)
159
160                     logging.debug(f"For theta angle {theta_angles}, generated {
161                     len(list_point)} points")
162                     i = i + 1
163
164
165         # Convert the list of arrays
166         all_points = np.array(all_points)
167         allPositions_polar = np.array(allPositions_polar)

```

```

159     logging.debug(f"Generated {len(all_points)} points for rotation
rings \n")
160
161     for i in range(len(all_points)):
162         print(f"Point {i}: ({np.round(all_points[i][0], 2)}, {np.round(
all_points[i][1], 2)}, {np.round(all_points[i][2], 2)})"
163             f" ({allPositions_polar[i][0]}, {np.round(np.degrees(
allPositions_polar[i][1]),2)}, {np.round(np.degrees(
allPositions_polar[i][2]),2)})")
164
165     return all_points, allPositions_polar[:, sequence_ID]
166
167 def arc_movement_vector(plane_object, coords):
168     """
169     Gets current position of plane and new position after and calculates
the vector between them
170
171     :arg: plane_object
172         plane_object (Plane): Contains position and orientation of plane
173
174         coords (np.array): Contains cartesian coordinates for next
position.
175
176         :return: translationVector (np.array): Contains vector from current
position to new position.
177
178         """
179
# translationVector = plane_object.position - coords # Incorrect
177     translationVector = coords - plane_object.position
178
179     return translationVector

```

H.9. analyse_results.py

```

1 import os
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 from utils_io import load_csv
5 from utils_metrics import compute_hit_percentage, compute_cost_per_gain
6 import numpy as np
7
8
9 def get_sim_title():
10     try:
11         df = load_csv("../data/results.csv")
12         return df["sim title"].iloc[-1]
13     except:
14         return "Simulation"
15

```

```

16
17 def plot_hit_percentage_combined():
18     df = load_csv("../data/results.csv")
19     sensor_df = load_csv("../data/sensor_results.csv")
20     sim_title = get_sim_title()
21
22     fig, axs = plt.subplots(1, 2, figsize=(14, 5))
23
24     # Overall Hit % per Arc Position
25     for ray_count in df["ray count"].unique():
26         subset = df[df["ray count"] == ray_count]
27         grouped = subset.groupby("idx")[["hits", "misses"]].mean()
28         grouped["hit_percentage"] = compute_hit_percentage(grouped)
29         axs[0].plot(grouped.index, grouped["hit_percentage"], marker="o",
30 , label=f"{ray_count} rays")
31
32     axs[0].set_title("Overall Hit %")
33     axs[0].set_xlabel("Arc Index (idx)")
34     axs[0].set_ylabel("Hit Percentage (%)")
35     axs[0].legend()
36     axs[0].grid(True)
37
38     # Per-Sensor Hit %
39     melted = sensor_df.melt(id_vars=["sim", "idx"], var_name="sensor",
40 value_name="hits")
41     totals = melted.groupby(["sim", "idx"])["hits"].sum().reset_index(
42 name="total_hits")
43     merged = pd.merge(melted, totals, on=["sim", "idx"])
44     merged["hit_pct"] = merged["hits"] / merged["total_hits"] * 100
45
46     avg_sensor_hits = merged.groupby(["sensor", "idx"])["hit_pct"].mean()
47     .reset_index()
48
49     for sensor, group in avg_sensor_hits.groupby("sensor"):
50         axs[1].plot(group["idx"], group["hit_pct"], marker="o", label=sensor)
51
52     axs[1].set_title("Per-Sensor Hit %")
53     axs[1].set_xlabel("Arc Index (idx)")
54     axs[1].set_ylabel("Hit Percentage (%)")
55     axs[1].legend()
56     axs[1].grid(True)
57
58     # plt.suptitle(f"Hit Percentage Analysis - {sim_title}", fontsize=16)
59     plt.suptitle(f"Hit Percentage Analysis", fontsize=16)
60     plt.tight_layout()

```

```

57 plt.show()
58
59
60 def plot_runtime_vs_gain():
61     df = load_csv("../data/results.csv")
62     grouped = df.groupby("ray count").agg({
63         "hits": "sum",
64         "misses": "sum",
65         "runtime": "mean"
66     }).reset_index()
67
68     grouped["hit_percentage"] = compute_hit_percentage(grouped)
69     grouped["hit_gain"] = grouped["hit_percentage"].diff().fillna(0)
70     grouped["cost_per_gain"] = compute_cost_per_gain(grouped["runtime"],
71     grouped["hit_percentage"])
72
73     fig, axs = plt.subplots(1, 3, figsize=(16, 5))
74     axs[0].plot(grouped["ray count"], grouped["runtime"], marker="o",
75     color="red")
76     axs[0].set_title("Average Runtime vs Ray Count")
77
78     axs[1].plot(grouped["ray count"], grouped["hit_gain"], color="green")
79     axs[1].set_title("Marginal Gain in Hit %")
80
81     axs[2].plot(grouped["ray count"], grouped["cost_per_gain"], marker="o",
82     color="purple")
83     axs[2].set_title("Cost per Hit % Gain")
84
85     for ax in axs:
86         ax.set_xlabel("Ray Count")
87         ax.grid(True)
88
89     axs[0].set_ylabel("Runtime (s)")
90     axs[1].set_ylabel("Hit % Gain")
91     axs[2].set_ylabel("Seconds per % Gain")
92     plt.suptitle(f"Runtime and Efficiency Analysis", fontsize=16)
93     plt.tight_layout()
94     plt.show()
95
96
97 def compare_sim_vs_real():
98     sim_df = load_csv("../data/sensor_results.csv")
99     phy_df = load_csv("../data/physical_data_messy.csv")
100    angle_df = load_csv("../data/rigid_arc_angles.csv")
101
102    sim_data = sim_df.iloc[:, 2:] / sim_df.iloc[:, 2:].sum(axis=1).

```

```

values[:, None] * 100
100 sim_pos = angle_df['arc_angle_deg'].unique()
101
102 phy_filtered = phy_df.iloc[:, 6:]
103 phy_time = phy_df.iloc[:, 1]
104
105 fig, axes = plt.subplots(1, 2, figsize=(12, 5))
106 for col in sim_data.columns:
107     axes[0].plot(sim_pos, sim_data[col], label=col)
108 axes[0].set_title("Simulated (Scaled)")
109
110 for col in phy_filtered.columns:
111     axes[1].plot(phy_time, phy_filtered[col], label=col)
112 axes[1].set_title("Experimental (Filtered)")
113
114 for ax in axes:
115     ax.set_xlabel("Position")
116     ax.set_ylabel("Response")
117     ax.grid(True)
118     ax.legend()
119
120 plt.suptitle("Simulated vs Experimental Sensor Response", fontsize=16)
121 plt.tight_layout()
122 plt.show()
123
124
125 def sensor_surface_plots():
126     from mpl_toolkits.mplot3d import Axes3D
127
128     angle_df = load_csv("../data/rigid_arc_angles.csv")
129     sensor_df = load_csv("../data/sensor_results.csv")
130
131     num_sensors = sensor_df.shape[1] - 2 # exclude sim and idx
132     cols = int(np.ceil(np.sqrt(num_sensors)))
133     rows = int(np.ceil(num_sensors / cols))
134
135     fig, axes = plt.subplots(rows, cols, figsize=(18, 6), subplot_kw={'projection': '3d'})
136     if rows * cols > 1:
137         axes = axes.flatten()
138     else:
139         axes = [axes]
140
141     sensor_data = sensor_df.iloc[:, 2:]
142
143     for idx in range(num_sensors):

```

```

144     combined = angle_df.copy()
145     combined["sensor_response"] = sensor_data.iloc[:, idx].values
146
147     pivot = combined.pivot(index="tilt_angle_deg", columns="arc_angle_deg", values="sensor_response")
148     X, Y = np.meshgrid(pivot.columns.values, pivot.index.values)
149     Z = pivot.values
150
151     axes[idx].plot_surface(X, Y, Z, cmap='plasma')
152     axes[idx].set_title(f"Sensor {sensor_data.columns[idx]}")
153     axes[idx].set_xlabel('Arc Angle (degree)')
154     axes[idx].set_ylabel('Tilt Angle (degree)')
155     axes[idx].set_zlabel('Hits')
156
157     plt.suptitle(f'Sensor Response Surface Plots - {get_sim_title()}', font-size=16)
158     plt.tight_layout()
159     plt.show()
160
161 def plot_per_test_summary():
162     df = load_csv("../data/results.csv")
163     sensor_df = load_csv("../data/sensor_results.csv")
164
165     if "sim title" not in df.columns:
166         print("Missing 'sim title' column in results.csv")
167         return
168
169     # Merge overall and sensor-level data
170     sensor_df = sensor_df.copy()
171     sensor_df["sim title"] = df["sim title"] # add test names to sensor file
172
173     grouped = df.groupby("sim title") [["hits", "misses"]].sum()
174     grouped["hit_percentage"] = compute_hit_percentage(grouped)
175
176     # Overall Hit % per Test
177     fig, axs = plt.subplots(1, 2, figsize=(14, 5))
178
179     axs[0].bar(grouped.index, grouped["hit_percentage"], color="skyblue")
180     axs[0].set_title("Overall Hit % by Test Case")
181     axs[0].set_ylabel("Hit Percentage (%)")
182     axs[0].tick_params(axis='x', rotation=45)
183     axs[0].grid(True)
184
185     # Per-Sensor Hit % per Test
186     melted = sensor_df.melt(id_vars=["sim", "idx", "sim title"],

```

```

var_name="sensor", value_name="hits")
187 totals = melted.groupby(["sim title", "sim", "idx"])["hits"].sum().
reset_index(name="total_hits")
188 merged = pd.merge(melted, totals, on=["sim title", "sim", "idx"])
189 merged["hit_pct"] = merged["hits"] / merged["total_hits"] * 100
190
191 sensor_avg = merged.groupby(["sim title", "sensor"])["hit_pct"].mean().
unstack()
192
193 sensor_avg.plot(kind="bar", stacked=False, ax=axs[1])
194 axs[1].set_title("Average Hit % per Sensor by Test")
195 axs[1].set_ylabel("Hit Percentage (%)")
196 axs[1].tick_params(axis='x', rotation=45)
197 axs[1].legend(title="Sensor")
198 axs[1].grid(True)
199
200 plt.suptitle("Test-by-Test Summary Analysis", fontsize=16)
201 plt.tight_layout()
202 plt.show()
203
204
205 def menu():
206     options = {
207         "1": ("Plot Overall and Per-Sensor Hit %",
208               plot_hit_percentage_combined),
209         "2": ("Plot Runtime vs Gain and Cost Analysis",
210               plot_runtime_vs_gain),
211         "3": ("Compare Simulated vs Real Sensor Data",
212               compare_sim_vs_real),
213         "4": ("Generate Sensor Response Surface Plots",
214               sensor_surface_plots),
215         "5": ("Plot Overall and Per-Sensor Hit % by Test Case",
216               plot_per_test_summary),
217         "6": ("Exit", exit)
218     }
219
220     while True:
221         print("\nSelect an analysis option:")
222         for key, (desc, _) in options.items():
223             print(f" {key}. {desc}")
224
225         choice = input("Enter choice: ")
226         if choice in options:
227             print(f"\nRunning: {options[choice][0]}\n")
228             options[choice][1]()
229         else:
230             print("Invalid choice. Try again.")

```

```

226
227
228 if __name__ == "__main__":
229     menu()

```

H.10. interface.py

```

1 import tkinter as tk
2 from tkinter import messagebox
3 from tkinter import font
4 from tkinter import ttk
5
6 import ttkbootstrap as ttk
7 from ttkbootstrap.constants import *
8
9 import json
10 import ast
11 from config import Config
12 from main import run_all_test
13
14 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
15 from matplotlib.figure import Figure
16
17 CONFIG_FILE = "../config.json"
18
19 READABLE_NAMES = {
20     # Sections
21     "planes": "Planes",
22     "sensor_areas": "Sensor Areas",
23     "aperture_areas": "Aperture Areas",
24     "arc_movement": "Arc Movement",
25     "simulation": "Simulation Settings",
26     "intersection": "Intersection Controls",
27     "visualization": "Visualisation",
28     "debugging": "Debugging Options",
29     "performance": "Performance Tuning",
30
31     "num_lines": "Number of Lines",
32     "num_runs": "Number of Runs",
33     "max_distance": "Maximum Distance",
34     "strict_mode": "Strict Mode",
35     "tolerance": "Tolerance",
36     "show_sensor_plane": "Show Sensor Plane",
37     "show_source_plane": "Show Source Plane",
38     "rotation_axis": "Rotation Axis",
39     "frame_rate": "Frame Rate"
40 }

```

```

41
42 import matplotlib.patches as patches
43
44 def plot_sensor_layout_in_frame(frame, config_obj):
45     # Clear frame if there's an existing plot
46     for widget in frame.winfo_children():
47         widget.destroy()
48
49     # Sim dimensions
50     sim_width = 10
51     sim_height = 10
52     sensor_positions = get_simulated_sensor_positions()
53
54     sensor_keys = ["sensor_A", "sensor_B", "sensor_C", "sensor_D"]
55
56     for key, pos in zip(sensor_keys, sensor_positions):
57         # Update the position field (Z = 0)
58         config_obj.sensor_areas[key]["position"] = [pos[0], pos[1], 0]
59
60     # Create a matplotlib Figure
61     fig = Figure(figsize=(5, 5), dpi=100)
62     ax = fig.add_subplot(111)
63     ax.set_aspect('equal')
64
65     # Draw the mount
66     ax.add_patch(patches.Rectangle(
67         (-sim_width / 2, -sim_height / 2),
68         sim_width, sim_height,
69         linewidth=1,
70         edgecolor='black',
71         facecolor='lightgrey',
72         linestyle='--',
73         label='Mounting Surface'
74     ))
75
76     # Plot sensors
77     for idx, (x, y) in enumerate(sensor_positions):
78         ax.plot(x, y, 'ro')
79         ax.text(x + 0.2, y + 0.2, f"A{idx}", fontsize=9)
80
81     ax.set_xlim(-sim_width / 2 - 1, sim_width / 2 + 1)
82     ax.set_ylim(-sim_height / 2 - 1, sim_height / 2 + 1)
83     ax.set_xlabel('X Position (sim units)')
84     ax.set_ylabel('Y Position (sim units)')
85     ax.set_title('Sensor Layout')
86     ax.grid(True)
87     ax.legend()

```

```

88
89 # Embed into Tkinter
90 canvas = FigureCanvasTkAgg(fig, master=frame)
91 canvas.draw()
92 canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)
93
94 def get_simulated_sensor_positions():
95
96     schematic_width_mm = 57
97     schematic_height_mm = 62
98
99     sim_width = 10
100    sim_height = 10
101
102    # Scale factors
103    scale_x = sim_width / schematic_width_mm
104    scale_y = sim_height / schematic_height_mm
105
106    # offset to shift to origin
107    center_x_mm = schematic_width_mm / 2
108    center_y_mm = schematic_height_mm / 2
109
110    # Sensor positions from schematic (in mm)
111    sensor_mm_positions = [
112        (28.5, 32),    # Sensor 0
113        (36.5, 32),    # Sensor 1
114        (44, 38),      # Sensor 2
115        (44, 46)       # Sensor 3
116    ]
117
118    # Convert to simulation coordinates
119    sensor_sim_positions = []
120    for (x_mm, y_mm) in sensor_mm_positions:
121        x_sim = (x_mm - center_x_mm) * scale_x
122        y_sim = (y_mm - center_y_mm) * scale_y
123        print(f"[{x_sim}, {y_sim}]")
124        sensor_sim_positions.append((x_sim, y_sim))
125
126    return sensor_sim_positions
127
128 def get_readable_name(key):
129     return READABLE_NAMES.get(key, key.replace("_", " ").title())
130
131 def load_config_defaults():
132     with open(CONFIG_FILE, "r") as f:
133         return json.load(f)
134

```

```

135
136
137 def config_gui():
138     original_config = load_config_defaults()
139     entry_widgets = {}
140     config_obj = Config(data=original_config)
141
142     MIN_ITEMS_TO_GROUP = 4
143     grouped_sections = []
144     individual_sections = []
145
146     for section_name, section_data in original_config.items():
147         if isinstance(section_data, dict):
148             if len(section_data) <= MIN_ITEMS_TO_GROUP:
149                 grouped_sections.append(section_name)
150             else:
151                 individual_sections.append(section_name)
152
153     def combine_sections(container, section_names, start_row, start_col,
154                          num_columns):
155         """
156             Groups multiple small sections and packs them vertically into
157             grid cells.
158         """
159
160         row = start_row
161         col = start_col
162         max_sections_per_column = 5
163         grouped = 0
164
165         for i in range(0, len(section_names), max_sections_per_column):
166             frame_group = section_names[i:i + max_sections_per_column]
167             combined_frame = ttk.Frame(container)
168             combined_frame.grid(row=row, column=col, padx=15, pady=15,
169             sticky="nw")
170
171             for section_name in frame_group:
172                 section_data = original_config[section_name]
173
174                 bold_font = ("Segoe UI", 11, "bold")
175
176                 sec_frame = ttk.LabelFrame(
177                     combined_frame,
178                     text=get_readable_name(section_name),
179                     bootstyle="secondary",
180                     borderwidth=5,
181                     relief="ridge"
182                 )

```

```

179
180         sec_frame.configure(labelwidget=ttk.Label(sec_frame,
181     text=get_readable_name(section_name), font=bold_font))
182
183         sec_frame.pack(fill="x", expand=True, pady=(0, 10))
184         create_entries(sec_frame, section_name, section_data, 3)
185
186         col += 1
187         if col >= num_columns:
188             col = 0
189             row += 1
190
191
192
193     def create_entries(section_frame, section_name, section_data,
194 INTERNAL_NUM_COLUMNS):
195
196         row_index = 0
197         for key, value in section_data.items():
198             col = row_index % INTERNAL_NUM_COLUMNS
199             row = row_index // INTERNAL_NUM_COLUMNS
200
201             container = ttk.Frame(section_frame)
202             container.grid(row=row, column=col, sticky="ew", padx=10,
203 pady=10)
204
205
206             if isinstance(value, bool):
207                 var = tk.BooleanVar(value=value)
208                 cb = ttk.Checkbutton(container, variable=var)
209                 cb.pack(side="top", anchor="w")
210                 entry_widgets[(section_name, key)] = var
211             else:
212                 entry = ttk.Entry(container)
213                 entry.insert(0, str(value))
214                 entry.pack(side="top", fill="x", expand=True)
215                 entry_widgets[(section_name, key)] = entry
216
217             row_index += 1
218
219     def apply_and_close():
220         new_config = {}

```

```

221     for (section, key), widget in entry_widgets.items():
222         if section not in new_config:
223             new_config[section] = {}
224
225         if isinstance(widget, tk.BooleanVar):
226             new_config[section][key] = widget.get()
227         else:
228             text = widget.get()
229             try:
230                 parsed_value = ast.literal_eval(text)
231                 new_config[section][key] = parsed_value
232             except (ValueError, SyntaxError):
233                 new_config[section][key] = text
234
235     temp_config = Config(data=new_config)
236
237     messagebox.showinfo("Applied", "Configuration loaded into memory")
238
239     root.destroy()
240     run_with_config(temp_config)
241
242     root = ttk.Window(themename="darkly")
243     root.title("Edit Config for This Run")
244     root.geometry("1100x900")
245
246     style = ttk.Style()
247     # style.theme_use("clam")
248
249     default_font = font.nametofont("TkDefaultFont")
250     default_font.configure(size=11)
251
252     main_frame = ttk.Frame(root)
253     main_frame.pack(fill="both", expand=True)
254
255     canvas = tk.Canvas(main_frame, bd=0, highlightthickness=0)
256     scrollbar = ttk.Scrollbar(main_frame, orient="vertical", command=
257     canvas.yview)
258     scroll_frame = ttk.Frame(canvas)
259
260     scroll_frame.bind("<Configure>", lambda e: canvas.configure(
261     scrollregion=canvas.bbox("all")))
262     canvas.create_window((0, 0), window=scroll_frame, anchor="nw")
263     canvas.configure(yscrollcommand=scrollbar.set)
264
265     canvas.pack(side="left", fill="both", expand=True)
266     scrollbar.pack(side="right", fill="y")

```

```

265
266     def _on_mousewheel(event):
267         canvas.yview_scroll(int(-1 * (event.delta / 120)), "units")
268
269     canvas.bind_all("<MouseWheel>", _on_mousewheel)
270     canvas.bind_all("<Button-4>", lambda e: canvas.yview_scroll(-1, "units"))
271     canvas.bind_all("<Button-5>", lambda e: canvas.yview_scroll(1, "units"))
272
273     row = 0
274     col = 0
275     num_columns = 3
276
277     visualise_frame = ttk.LabelFrame(scroll_frame, text="Sensor Layout
278                                     Visualisation")
279     visualise_frame.grid(row=0, column=0, columnspan=1, padx=15, pady
280 =15, sticky="nsew")
281
282     col += 1
283     if col >= num_columns:
284         col = 0
285         row += 1
286
287     # Handle grouped small sections
288     row, col = combine_sections(scroll_frame, grouped_sections, row, col
289 , num_columns)
290
291     # Handle larger sections
292     for section_name in individual_sections:
293         bold_font = ("Segoe UI", 11, "bold")
294         sec_frame = ttk.LabelFrame(scroll_frame, text=get_readable_name(
295 section_name), bootstyle="secondary")
296         sec_frame.configure(labelwidget=ttk.Label(sec_frame, text=
297 get_readable_name(section_name), font=bold_font))
298
299         sec_frame.grid(row=row, column=col, padx=15, pady=15, sticky="nw
300 ")
301         create_entries(sec_frame, section_name, original_config[
302 section_name], 4)
303
304         col += 1
305         if col >= num_columns:
306             col = 0
307             row += 1
308
309     plot_area = ttk.Frame(visualise_frame)

```

```

303 plot_area.pack(fill="both", expand=True)
304
305 # Plot from current config initially
306 def plot_existing_config_layout():
307     # Read sensor positions from current config
308     sensor_keys = ["sensor_A", "sensor_B", "sensor_C", "sensor_D"]
309     positions = [config_obj.sensor_areas[k]["position"] for k in
310 sensor_keys]
311
312     # Flatten to 2D for plotting
313     sensor_positions = [(x, y) for x, y, _ in positions]
314     update_plot(plot_area, sensor_positions)
315
316     # Plot from schematic layout
317     def plot_simulated_layout_and_update_config():
318         sensor_positions = get_simulated_sensor_positions()
319         sensor_keys = ["sensor_A", "sensor_B", "sensor_C", "sensor_D"]
320         for key, pos in zip(sensor_keys, sensor_positions):
321             config_obj.sensor_areas[key]["position"] = [pos[0], pos[1],
322 0]
323
324         update_plot(plot_area, sensor_positions)
325
326
327     def update_plot(frame, sensor_positions):
328         for widget in frame.winfo_children():
329             widget.destroy()
330
331         sim_width = 10
332         sim_height = 10
333
334         fig = Figure(figsize=(5, 5), dpi=100)
335         ax = fig.add_subplot(111)
336         ax.set_aspect('equal')
337         ax.add_patch(patches.Rectangle(
338             (-sim_width / 2, -sim_height / 2),
339             sim_width, sim_height,
340             linewidth=1,
341             edgecolor='black',
342             facecolor='lightgrey',
343             linestyle='--',
344             label='Mounting Surface',
345         ))
346
347         for idx, (x, y) in enumerate(sensor_positions):
348             ax.plot(x, y, 'ro')
349             ax.text(x + 0.2, y + 0.2, f"A{idx}", fontsize=9)

```

```

348     ax.set_xlim(-sim_width / 2 - 1, sim_width / 2 + 1)
349     ax.set_ylim(-sim_height / 2 - 1, sim_height / 2 + 1)
350     ax.set_xlabel('X Position (sim units)')
351     ax.set_ylabel('Y Position (sim units)')
352     ax.set_title('Sensor Layout')
353     ax.grid(True)
354     ax.legend()
355
356     fig.patch.set_facecolor("#212529") # or match ttkbootstrap dark
357     bg
358     ax.set_facecolor("#2a2a2a") # slightly lighter for
359     contrast
360
361     ax.title.set_color("white")
362     ax.xaxis.label.set_color("white")
363     ax.yaxis.label.set_color("white")
364     ax.tick_params(axis='x', colors='white')
365     ax.tick_params(axis='y', colors='white')
366
367     ax.spines['bottom'].set_color('white')
368     ax.spines['left'].set_color('white')
369     ax.spines['top'].set_color('white')
370     ax.spines['right'].set_color('white')
371     ax.legend().get_frame().set_facecolor("#343a40")
372     ax.legend().get_frame().set_edgecolor("white")
373
374     canvas = FigureCanvasTkAgg(fig, master=frame)
375     canvas.draw()
376     canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)
377
378     # Initial plot
379     plot_existing_config_layout()
380
381     # Add buttons
382     btn_frame = ttk.Frame(visualise_frame)
383     btn_frame.pack(pady=5)
384
385     ttk.Button(btn_frame, text="Load Existing Layout", command=
386     plot_existing_config_layout, bootstyle="light").pack(side="left",
387     padx=5)
388     ttk.Button(btn_frame, text="Calculate Layout", command=
389     plot_simulated_layout_and_update_config, bootstyle="light").pack(side
390     ="left", padx=5)
391
392     for key in config_obj.sensor_areas:
393         print(f"{key}: {config_obj.sensor_areas[key]['position']}")

```

```

389
390     apply_button = ttk.Button(scroll_frame, text="Apply and Run",
391                             command=apply_and_close, bootstyle="light")
392     apply_button.grid(row=row, column=col, columnspan=1, pady=20)
393
394     root.mainloop()
395
396 def run_with_config(temp_config):
397     print("Custom config object created in memory")
398     print(f"Planes available: {list(temp_config.planes.keys())}")
399     print("Launching simulation with updated config...")
400     run_all_test(temp_config, temp_config.simulation["num_lines"])
401
402 if __name__ == "__main__":
403     config_gui()

```

H.11. requirements.txt

```

1 contourpy==1.3.1
2 cycler==0.12.1
3 fonttools==4.56.0
4 kaleido==0.2.1
5 kiwisolver==1.4.8
6 matplotlib==3.10.0
7 memory-profiler==0.61.0
8 narwhals==1.27.1
9 numpy==2.1.3
10 packaging==24.2
11 pandas==2.2.3
12 pillow==11.1.0
13 plotly==6.0.0
14 psutil==7.0.0
15 pyparsing==3.2.1
16 python-dateutil==2.9.0.post0
17 pytz==2025.2
18 scipy==1.15.2
19 six==1.17.0
20 tzdata==2025.2
21 ttkbootstrap~=1.12.0

```

H.12. test_arc_rotation.json

```

1 {
2     "planes": {
3         "source_plane": {
4             "position": [
5                 0,

```

```

6          0,
7          1
8      ],
9      "direction": [
10         0,
11         0,
12         -1
13 ],
14     "width": 10,
15     "length": 10
16 },
17   "sensor_plane": {
18     "position": [
19       0,
20       0,
21       0
22 ],
23     "direction": [
24       0,
25       0,
26       -1
27 ],
28     "width": 10,
29     "length": 10
30 },
31   "aperture_plane": {
32     "position": [
33       0,
34       0,
35       1
36 ],
37     "direction": [
38       0,
39       0,
40       1
41 ],
42     "width": 5,
43     "length": 5
44   }
45 },
46   "sensor_areas": {
47     "sensor_A": {
48       "title": "A0",
49       "position": [
50         0,
51         -3.25,
52         0

```

```

53     ] ,
54     "direction": [
55         0,
56         0,
57         1
58     ] ,
59     "width": 1.5 ,
60     "length": 1.5
61 },
62 "sensor_B": {
63     "title": "A1",
64     "position": [
65         0,
66         -1.25 ,
67         0
68     ] ,
69     "direction": [
70         0,
71         0,
72         1
73     ] ,
74     "width": 1.5 ,
75     "length": 1.5
76 },
77 "sensor_C": {
78     "title": "A2",
79     "position": [
80         -1,
81         2.25 ,
82         0
83     ] ,
84     "direction": [
85         0,
86         0,
87         1
88     ] ,
89     "width": 1.5 ,
90     "length": 1.5
91 },
92 "sensor_D": {
93     "title": "A3",
94     "position": [
95         1,
96         2.25 ,
97         0
98     ] ,
99     "direction": [

```

```

100          0,
101          0,
102          1
103      ],
104      "width": 1.5,
105      "length": 1.5
106  }
107 },
108 "aperture_areas": {
109   "Aperture_A": {
110     "title": "Aperture_A",
111     "position": [
112       0,
113       -2.25,
114       1
115     ],
116     "direction": [
117       0,
118       0,
119       1
120     ],
121     "width": 4,
122     "length": 2
123   },
124   "Aperture_B": {
125     "title": "Aperture_B",
126     "position": [
127       0,
128       2.25,
129       1
130     ],
131     "direction": [
132       0,
133       0,
134       1
135     ],
136     "width": 2,
137     "length": 4
138   }
139 },
140 "arc_movement": {
141   "execute_movements": true,
142   "initial_rotation": 90.0,
143   "rotation_axis": "y",
144   "radius": 10.0,
145   "horizontal_circles": false,
146   "horizontal_primary": 60,

```

```

147     "horizontal_secondary": 30,
148     "vertical_circles": false,
149     "vertical_primary": 10,
150     "vertical_secondary": [
151         0,
152         65,
153         130
154     ],
155     "rigid_arc": true,
156     "rigid_arc_step": 10,
157     "tilt_angles": [
158         0
159     ]
160 },
161 "simulation": {
162     "num_lines": 1000,
163     "num_runs": 1
164 },
165 "intersection": {
166     "max_distance": 100,
167     "strict_mode": true,
168     "tolerance": 0.01
169 },
170 "visualization": {
171     "show_output_parent": false,
172     "show_sensor_plane": true,
173     "show_source_plane": false,
174     "show_sensor_area": true,
175     "show_aperture_area": true,
176     "show_aperture_plane": false,
177     "color_sensor_plane": "red",
178     "color_source_plane": "yellow",
179     "color_aperture_plane": "green",
180     "color_sensor_area": "green",
181     "color_aperture_area": "black",
182     "show_axes": true,
183     "animated_plot": false,
184     "frame_rate": 500,
185     "hits_to_display": 2,
186     "misses_to_display": 2
187 },
188 "debugging": {
189     "enable_logging": true,
190     "logging_level": "INFO",
191     "data_csv_path": "data/results.csv"
192 },
193 "performance": {

```

```

194     "enable_memory_profiling": true,
195     "optimization_level": "basic"
196   },
197   "output": {
198     "save_static_png": false,
199     "save_animated_gif": false,
200     "Sim_title": "test_arc_rotation"
201   }
202 }
```

H.13. test_directly_below.json

```

1 {
2   "planes": {
3     "source_plane": {
4       "position": [
5         0,
6         0,
7         1
8       ],
9       "direction": [
10      0,
11      0,
12      -1
13     ],
14     "width": 10,
15     "length": 10
16   },
17   "sensor_plane": {
18     "position": [
19       0,
20       0,
21       0
22     ],
23     "direction": [
24       0,
25       0,
26       -1
27     ],
28     "width": 10,
29     "length": 10
30   },
31   "aperture_plane": {
32     "position": [
33       0,
34       0,
35       1
36     ]
37   }
38 }
```

```

36     ] ,
37     "direction": [
38         0,
39         0,
40         1
41     ] ,
42     "width": 5 ,
43     "length": 5
44   }
45 },
46   "sensor_areas": {
47     "sensor_A": {
48       "title": "AO",
49       "position": [
50         0,
51         0,
52         0
53     ] ,
54       "direction": [
55         0,
56         0,
57         1
58     ] ,
59       "width": 1.5 ,
60       "length": 1.5
61     }
62   },
63   "aperture_areas": {
64     "Full_Aperture": {
65       "title": "Full_Aperture",
66       "position": [
67         0,
68         0,
69         1
70     ] ,
71       "direction": [
72         0,
73         0,
74         1
75     ] ,
76       "width": 10 ,
77       "length": 10
78     }
79   },
80   "arc_movement": {
81     "execute_movements": true ,
82     "initial_rotation": 90.0 ,

```

```

83     "rotation_axis": "y",
84     "radius": 10.0,
85     "horizontal_circles": false,
86     "horizontal_primary": 60,
87     "horizontal_secondary": 30,
88     "vertical_circles": false,
89     "vertical_primary": 10,
90     "vertical_secondary": [
91         0,
92         65,
93         130
94     ],
95     "rigid_arc": true,
96     "rigid_arc_step": 10,
97     "tilt_angles": [
98         0
99     ]
100 },
101 "simulation": {
102     "num_lines": 1000,
103     "num_runs": 1
104 },
105 "intersection": {
106     "max_distance": 100,
107     "strict_mode": true,
108     "tolerance": 0.01
109 },
110 "visualization": {
111     "show_output_parent": false,
112     "show_sensor_plane": true,
113     "show_source_plane": false,
114     "show_sensor_area": true,
115     "show_aperture_area": true,
116     "show_aperture_plane": false,
117     "color_sensor_plane": "red",
118     "color_source_plane": "yellow",
119     "color_aperture_plane": "green",
120     "color_sensor_area": "green",
121     "color_aperture_area": "black",
122     "show_axes": true,
123     "animated_plot": false,
124     "frame_rate": 500,
125     "hits_to_display": 2,
126     "misses_to_display": 2
127 },
128 "debugging": {
129     "enable_logging": true,

```

```

130     "logging_level": "INFO",
131     "data_csv_path": "data/results.csv"
132   },
133   "performance": {
134     "enable_memory_profiling": true,
135     "optimization_level": "basic"
136   },
137   "output": {
138     "save_static_png": false,
139     "save_animated_gif": false,
140     "Sim_title": "test_directly_below"
141   }
142 }
```

H.14. test_no_intersection.json

```

1 {
2   "planes": {
3     "source_plane": {
4       "position": [
5         0,
6         0,
7         1
8       ],
9       "direction": [
10      1,
11      0,
12      0
13     ],
14     "width": 10,
15     "length": 10
16   },
17   "sensor_plane": {
18     "position": [
19       0,
20       0,
21       0
22     ],
23     "direction": [
24       0,
25       0,
26       -1
27     ],
28     "width": 10,
29     "length": 10
30   },
31   "aperture_plane": {
```

```

32     "position": [
33         0,
34         0,
35         1
36     ],
37     "direction": [
38         0,
39         0,
40         1
41     ],
42     "width": 5,
43     "length": 5
44 }
45 },
46 "sensor_areas": {
47     "sensor_A": {
48         "title": "A0",
49         "position": [
50             10,
51             10,
52             0
53         ],
54         "direction": [
55             0,
56             0,
57             1
58         ],
59         "width": 1.5,
60         "length": 1.5
61     }
62 },
63 "aperture_areas": {
64     "Full_Aperture": {
65         "title": "Full_Aperture",
66         "position": [
67             0,
68             0,
69             1
70         ],
71         "direction": [
72             0,
73             0,
74             1
75         ],
76         "width": 10,
77         "length": 10
78 }

```

```

79 },
80 "arc_movement": {
81   "execute_movements": true,
82   "initial_rotation": 90.0,
83   "rotation_axis": "y",
84   "radius": 10.0,
85   "horizontal_circles": false,
86   "horizontal_primary": 60,
87   "horizontal_secondary": 30,
88   "vertical_circles": false,
89   "vertical_primary": 10,
90   "vertical_secondary": [
91     0,
92     65,
93     130
94   ],
95   "rigid_arc": true,
96   "rigid_arc_step": 10,
97   "tilt_angles": [
98     0
99   ]
100 },
101 "simulation": {
102   "num_lines": 500,
103   "num_runs": 1
104 },
105 "intersection": {
106   "max_distance": 100,
107   "strict_mode": true,
108   "tolerance": 0.01
109 },
110 "visualization": {
111   "show_output_parent": false,
112   "show_sensor_plane": true,
113   "show_source_plane": false,
114   "show_sensor_area": true,
115   "show_aperture_area": true,
116   "show_aperture_plane": false,
117   "color_sensor_plane": "red",
118   "color_source_plane": "yellow",
119   "color_aperture_plane": "green",
120   "color_sensor_area": "green",
121   "color_aperture_area": "black",
122   "show_axes": true,
123   "animated_plot": false,
124   "frame_rate": 500,
125   "hits_to_display": 2,

```

```

126     "misses_to_display": 2
127 },
128 "debugging": {
129     "enable_logging": true,
130     "logging_level": "INFO",
131     "data_csv_path": "data/results.csv"
132 },
133 "performance": {
134     "enable_memory_profiling": true,
135     "optimization_level": "basic"
136 },
137 "output": {
138     "save_static_png": false,
139     "save_animated_gif": false,
140     "Sim_title": "test_no_intersection"
141 }
142 }
```

H.15. test_off_center.json

```

1 {
2     "planes": {
3         "source_plane": {
4             "position": [
5                 0,
6                 0,
7                 1
8             ],
9             "direction": [
10                0,
11                0,
12                -1
13             ],
14             "width": 10,
15             "length": 10
16         },
17         "sensor_plane": {
18             "position": [
19                 0,
20                 0,
21                 0
22             ],
23             "direction": [
24                 0,
25                 0,
26                 -1
27             ],

```

```
28     "width": 10,
29     "length": 10
30 },
31 "aperture_plane": {
32   "position": [
33     0,
34     0,
35     1
36   ],
37   "direction": [
38     0,
39     0,
40     1
41   ],
42   "width": 5,
43   "length": 5
44 }
45 },
46 "sensor_areas": {
47   "sensor_A": {
48     "title": "A0",
49     "position": [
50       0,
51       0,
52       0
53     ],
54     "direction": [
55       0,
56       0,
57       1
58     ],
59     "width": 1.5,
60     "length": 1.5
61 },
62   "sensor_B": {
63     "title": "A1",
64     "position": [
65       2,
66       0,
67       0
68     ],
69     "direction": [
70       0,
71       0,
72       1
73     ],
74     "width": 1.5,
```

```

75      "length": 1.5
76  },
77  "sensor_C": {
78    "title": "A2",
79    "position": [
80      -2,
81      0,
82      0
83    ],
84    "direction": [
85      0,
86      0,
87      1
88    ],
89    "width": 1.5,
90    "length": 1.5
91  }
92 },
93 "aperture_areas": {
94   "Full_Aperture": {
95     "title": "Full_Aperture",
96     "position": [
97       0,
98       0,
99       1
100    ],
101    "direction": [
102      0,
103      0,
104      1
105    ],
106    "width": 10,
107    "length": 10
108  }
109 },
110 "arc_movement": {
111   "execute_movements": true,
112   "initial_rotation": 90.0,
113   "rotation_axis": "y",
114   "radius": 10.0,
115   "horizontal_circles": false,
116   "horizontal_primary": 60,
117   "horizontal_secondary": 30,
118   "vertical_circles": false,
119   "vertical_primary": 10,
120   "vertical_secondary": [
121     0,

```

```

122     65,
123     130
124   ],
125   "rigid_arc": true,
126   "rigid_arc_step": 10,
127   "tilt_angles": [
128     0
129   ]
130 },
131 "simulation": {
132   "num_lines": 1000,
133   "num_runs": 1
134 },
135 "intersection": {
136   "max_distance": 100,
137   "strict_mode": true,
138   "tolerance": 0.01
139 },
140 "visualization": {
141   "show_output_parent": false,
142   "show_sensor_plane": true,
143   "show_source_plane": false,
144   "show_sensor_area": true,
145   "show_aperture_area": true,
146   "show_aperture_plane": false,
147   "color_sensor_plane": "red",
148   "color_source_plane": "yellow",
149   "color_aperture_plane": "green",
150   "color_sensor_area": "green",
151   "color_aperture_area": "black",
152   "show_axes": true,
153   "animated_plot": false,
154   "frame_rate": 500,
155   "hits_to_display": 2,
156   "misses_to_display": 2
157 },
158 "debugging": {
159   "enable_logging": true,
160   "logging_level": "INFO",
161   "data_csv_path": "data/results.csv"
162 },
163 "performance": {
164   "enable_memory_profiling": true,
165   "optimization_level": "basic"
166 },
167 "output": {
168   "save_static_png": false,

```

```

169     "save_animated_gif": false,
170     "Sim_title": "test_off_center"
171   }
172 }
```

H.16. test_with_aperture.json

```

1  {
2    "planes": {
3      "source_plane": {
4        "position": [
5          0,
6          0,
7          1
8        ],
9        "direction": [
10          0,
11          0,
12          -1
13      ],
14      "width": 10,
15      "length": 10
16    },
17    "sensor_plane": {
18      "position": [
19        0,
20        0,
21        0
22      ],
23      "direction": [
24        0,
25        0,
26        -1
27      ],
28      "width": 10,
29      "length": 10
30    },
31    "aperture_plane": {
32      "position": [
33        0,
34        0,
35        1
36      ],
37      "direction": [
38        0,
39        0,
40        1
41    }
42  }
43 }
```

```

41     ] ,
42     "width": 5 ,
43     "length": 5
44   }
45 },
46 "sensor_areas": {
47   "sensor_A": {
48     "title": "A0",
49     "position": [
50       0,
51       -2.25,
52       0
53     ],
54     "direction": [
55       0,
56       0,
57       1
58     ],
59     "width": 1.5 ,
60     "length": 1.5
61   }
62 },
63 "aperture_areas": {
64   "Aperture_A": {
65     "title": "Aperture_A",
66     "position": [
67       0,
68       -2.25,
69       1
70     ],
71     "direction": [
72       0,
73       0,
74       1
75     ],
76     "width": 4 ,
77     "length": 2
78   },
79   "Aperture_B": {
80     "title": "Aperture_B",
81     "position": [
82       0,
83       2.25,
84       1
85     ],
86     "direction": [
87       0,

```

```

88         0,
89         1
90     ],
91     "width": 2,
92     "length": 4
93   }
94 },
95 "arc_movement": {
96   "execute_movements": true,
97   "initial_rotation": 90.0,
98   "rotation_axis": "y",
99   "radius": 10.0,
100  "horizontal_circles": false,
101  "horizontal_primary": 60,
102  "horizontal_secondary": 30,
103  "vertical_circles": false,
104  "vertical_primary": 10,
105  "vertical_secondary": [
106    0,
107    65,
108    130
109  ],
110  "rigid_arc": true,
111  "rigid_arc_step": 10,
112  "tilt_angles": [
113    0
114  ]
115 },
116 "simulation": {
117   "num_lines": 1000,
118   "num_runs": 1
119 },
120   "intersection": {
121     "max_distance": 100,
122     "strict_mode": true,
123     "tolerance": 0.01
124 },
125   "visualization": {
126     "show_output_parent": false,
127     "show_sensor_plane": true,
128     "show_source_plane": false,
129     "show_sensor_area": true,
130     "show_aperture_area": true,
131     "show_aperture_plane": false,
132     "color_sensor_plane": "red",
133     "color_source_plane": "yellow",
134     "color_aperture_plane": "green"

```

```

135     "color_sensor_area": "green",
136     "color_aperture_area": "black",
137     "show_axes": true,
138     "animated_plot": false,
139     "frame_rate": 500,
140     "hits_to_display": 2,
141     "misses_to_display": 2
142   },
143   "debugging": {
144     "enable_logging": true,
145     "logging_level": "INFO",
146     "data_csv_path": "data/results.csv"
147   },
148   "performance": {
149     "enable_memory_profiling": true,
150     "optimization_level": "basic"
151   },
152   "output": {
153     "save_static_png": false,
154     "save_animated_gif": false,
155     "Sim_title": "test_with_aperture"
156   }
157 }
```

H.17. utils_io.py

```

1 import pandas as pd
2 import os
3
4 def load_csv(file_path, **kwargs):
5     if not os.path.exists(file_path):
6         raise FileNotFoundError(f"Missing {file_path}")
7     return pd.read_csv(file_path, **kwargs)
```

H.18. utils_metrics.py

```

1 import pandas as pd
2 import numpy as np
3
4
5 def compute_hit_percentage(df):
6     return df["hits"] / (df["hits"] + df["misses"]) * 100
7
8 def compute_cost_per_gain(runtime_series, hit_percentage_series):
9     runtime_gain = runtime_series.diff().fillna(0)
10    hit_gain = hit_percentage_series.diff().replace(0, np.nan)
11    return runtime_gain / hit_gain
```

H.19. utils_plot.py

```
1 import matplotlib.pyplot as plt
2
3 def plot_hit_rate(ax, x, y, label, marker="o"):
4     ax.plot(x, y, marker=marker, label=label)
5     ax.set_title("Hit Rate")
6     ax.set_xlabel("Position")
7     ax.set_ylabel("Hit Percentage")
8     ax.legend()
9     ax.grid(True)
```