



**Department of Electrical and Computer Engineering Technology (ECET)
School of Engineering, Technology, and Advanced Manufacturing (ETAM)**

EET 4950

Senior Design Project

Astraeus
Prototype Solar Support Rover

Submitted by:

Mark Figueroa & Pedro Cabrera

Supervised by:

Dr. Hall

July 18, 2025

Abstract

Technology to explore other planets is continuously evolving, driven by engineers' relentless curiosity about what humanity's potential will be beyond Earth. Mars stands as NASA's key focus for human exploration due to its potential to have once supported life and its significance in deepening our understanding of Earth's history, and perhaps even our future. NASA's initial Mars missions, including the Viking program, provided fundamental knowledge for understanding how to approach future man missions to the red planet. With plans to land on Mars by the 2030s [1], advancements in rover technology will play a crucial role in supporting humanity's efforts to explore and survive on Martian soil.

To support human missions beyond Earth, our goal is to design an autonomous rover specifically for ensuring the reliability and efficiency of solar energy infrastructure in extreme extraterrestrial environments. Dust accumulation on solar panels poses a significant threat to sustained power generation, potentially compromising mission success. Our rover autonomously navigates to designated solar panels based on crew commands, efficiently removing dust and debris to restore optimal energy absorption. Once cleaning is complete, the rover logs maintenance data, including timestamps and images, before returning to its docking station for recharging. By streamlining solar panel upkeep and minimizing manual intervention, our system enhances long-term exploration efforts and supports sustained human presence on Mars and beyond.

Acknowledgements

We would like to extend our gratitude to everyone who has contributed to the success of the Astraeus project. First and foremost, we would like to sincerely thank Dr. Hall, our advisor, for her invaluable guidance and mentorship throughout this project. Her support and insights have been instrumental in shaping our work.

We are grateful to Dr. Carbone for his expertise in planning our AI components and providing valuable insights into the project's scope, and to Dr. Britt for supplying Martian soil through the UCF Exolith Lab for our testing.

A big thank you to Will Goodman for his continued support with ordering components through Valencia College and supporting our ambitions for this project. Thank you to Thomas Dillen for providing us with access to the Innovation Lab for prototyping and testing.

Finally, we would like to extend our gratitude to the judges who have taken the time to evaluate our project. Their feedback and insights are invaluable in refining our work and pushing us to achieve excellence.

This project would not have been possible without the contributions, guidance, and encouragement of these individuals, and we are deeply thankful for their support.

Table of Contents

Abstract	i
Acknowledgements	ii
Table of Contents.....	iii
List of Tables	xi
Chapter 1 Introduction	1
1.1 Project Overview	2
1.2 Problem Definition.....	2
1.3 Project Objectives	3
1.4 Motivation	4
1.5 Project Requirements & Specifications	5
1.5.1 Engineering Requirements.....	5
1.5.2 Engineering Specifications	7
1.6 System Block Diagrams	10
1.6.1 Full System Architecture	10
1.6.2 Power Distribution	11
1.6.3 Electromechanical Subsystem.....	12
1.6.4 Vision & Sensor Integration	12
1.7 Limitations	13
1.7.1 Scale and Functionality.....	13
1.7.2 Component Selection	13
1.7.3 Autonomy & AI Capabilities	14
1.7.4 Panel Orientation Constraint.....	14
1.7.5 Testing Environment.....	14
1.7.6 Power and Runtime Constraints.....	14
1.7.7 Communication.....	14
1.8 Comparison of Existing Products.....	15

1.9 Report Structure.....	18
Chapter 2 Background Research	20
2.1 Rocker-Bogie Suspension System.....	21
2.1.1 3D Printing for Rocker Bogie Components	21
2.1.2 Implementing Rocker-Bogie	23
2.2 Martian Regolith Research	24
2.3 Drive System & Motor Control.....	25
2.3.1 Drive System.....	25
2.2.2 Motor Control	26
2.3 Robotic Arm & Cleaning Mechanism	27
2.3.1 Arm Control and Integration.....	28
2.3.2 Cleaning Testing	29
2.4 Sensors & Vision.....	31
2.4.1 Proximity Sensors	31
2.4.2 AI-Based Vision with HuskyLens	32
2.5 Control Platform and Software.....	33
2.5.1 Programming Languages and Software Architecture	34
2.5.2 Software Libraries and Dependencies.....	35
2.6 Navigations & Task Execution Algorithms	36
2.6.1 Main System Flow	36
2.6.2 Manual Navigation Logic	37
2.6.3 Autonomous Navigation Logic.....	39
2.6.4 Panel 1 Cleaning Subroutine.....	41
2.6.5 Panel 2 Cleaning Subroutine.....	43
2.6.6 Full Cleaning Routine	45
2.7 Power Budget	47

Chapter 3 Contributions	50
3.1 Main Chassis Contributions	51
3.1.1 Structural framework and layout.....	51
3.1.2 Structural Design Features	52
3.1.3 Assembly and Dimensional Refinements	55
3.2 Cleaning Arm Contributions	55
3.2.1 CAD Design & Mechanical Layout.....	56
3.2.1 Initial Servo Selection and Justification.....	57
3.2.3 Power Considerations	58
3.2.4 Robotic Arm CAD Design.....	59
3.2.5 Brush Cleaning Mechanism CAD Design	61
3.2.6 Arm & Electrical Integration	62
3.2.7 Servo Programing with Maestro Control Center	65
3.3 Rocker-Bogie Contributions	66
3.3.1 Rocker-Bogie Design.....	67
3.4 Software Development	70
3.4.1 Software Architecture Overview.....	70
3.4.2 Main Execution and Control Logic.....	71
3.4.3 Manual Control System	75
3.4.4 Autonomous Navigation and Tag Alignment	78
3.4.5 Motor and Movement Control	84
3.4.6 Sensor Integration and Obstacle Detection.....	87
3.4.7 External Subsystem Control.....	89
3.4.8 Event Logging and Dashboard Interface.....	90
3.4.9 Safety and Fail-Safe Mechanisms.....	96
3.4.10 Modular Design and Scalability.....	97
Chapter 4 Non-Technical Issues	99

4.1 Project Timeline	100
4.1.1 Proposal Phase (Spring 2025).....	100
4.1.2 Design Phase (Summer 2025).....	104
4.2 Budget.....	107
4.3 Environmental Aspects	112
4.4 Health and Safety Considerations	112
4.5 Ethical Aspects	113
4.6 Sustainability Considerations	113
 Chapter 5 Conclusion.....	 115
5.1 Summary and Conclusion	116
5.1.2 Robotic Arm Performance and Brush Results	116
5.1.2 Autonomous Capabilities Results	118
5.2 Suggestions for Future Work.....	119
5.2.1 Solar Panel Efficiency Monitoring.....	120
5.2.2 Self-Docking Charging Station.....	120
 APPENDICES	 123
Appendix A – Email Correspondence	124
Appendix B – VNH5019 Datasheet.....	125
Appendix C – Maestro Servo Controller User Guild.....	132
Appendix D – Solartech SPM030P-WP-F data sheet	137
Appendix E – GP2D120 Datasheet.....	139
Appendix F – Raspberry Pi 3 Model B	144
Appendix G – Maestro Code Script.....	147
Appendix H – Astraeus Source Code.....	154
Appendix H.1 – start_all.py	154
Appendix H.2 – main.py	156

Appendix H.3 – autonomy.py	159
Appendix H.4 – motors.py	171
Appendix H.5 – sharp_sensors.py	174
Appendix H.6 – visual_module.py	177
Appendix H.7 – maestro_module.py	179
Appendix H.8 – logger.py	181
Appendix H.9 – shared_state.py	182
Appendix H.10 – command_center.py	183
Appendix H.11 – failsafe_manual.py	184
Appendix H.12 – init_db.py	187
Appendix H.13 – app.py	188
Appendix H.14 – joystick.js	191
Appendix H.15 – home.html	193
Appendix H.16 – index.html	195
Appendix H.17 – autonomous.html	200
Appendix H.18 – log_table.html	203
Appendix H.19 – manual.html	205
Appendix H.20 – mode.html	209
Appendix I – IEEE Code of Ethics	212
Group Members	213

List of Figures

Figure 1. Full System Block Diagram	11
Figure 2. Power Distribution Block Diagram	11
Figure 3. Electromechanical Block Diagram	12
Figure 4. Vision & Sensor Module Block Diagram.....	13
Figure 5. Opportunity Rover	16
Figure 6. Curiosity Rover Radioisotope Thermoelectric Generator	16
Figure 7. SolarCleano F1	17
Figure 8. Rocker-bogie Suspension System Range of Contact	21
Figure 9. Overture PETG 1.75mm 3D Printer Filament.....	22
Figure 10. CAD model of the rocker-bogie suspension.....	23
Figure 11. Martian Regolith.....	24
Figure 12. Geartisan DC 12V 100RPM Gear Motor	26
Figure 13. VNH5019 Motor Driver Carrier.....	26
Figure 14. 25kg-DS3225 High-torque Digital Servo.....	27
Figure 15. ANNIMOS 45 kg High-torque Digital Servo.....	27
Figure 16. 55g High-torque Digital Servo	28
Figure 17. N20 75:1 12V Micro Gear Motor.....	28
Figure 18. Mini Maestro 12-Channel USB Servo Controller	29
Figure 19. Solartech SPM030P-WP-F Polycrystalline Panel	30
Figure 20. Sharp GP2D120 Infrared (IR) Proximity Sensors.....	31
Figure 21. ADS1115 16-bit analog-to-digital converter.....	32
Figure 22. HuskyLens AI Camera	32
Figure 23. Raspberry Pi 3 Model B	33
Figure 24. Main Flowchart.....	37
Figure 25. Navigation Process Flowchart	38
Figure 26. Alignment Process Flowchart.....	40
Figure 27. Panel 1 Cleaning Flowchart.....	42
Figure 28. Panel 2 Cleaning Flowchart.....	44
Figure 29. Clean All Flowchart.....	46
Figure 30. Botku 12 30Ah Lithium LiFePO4 Deep Cycle Battery	47

Figure 31. Corner Brackets on Print Plate	52
Figure 32. Main Chassis CAD Model.....	52
Figure 33. Rocker-Bogie Mounting Hubs.....	53
Figure 34. Base Rotation Servo Mount.....	53
Figure 35. Robotic Arm Baseplate & Lazy Susan Integration	54
Figure 36. Battery Tray Integration	54
Figure 37. Main Chassis Assembled.....	55
Figure 38. Initial Robotic Arm CAD Model (Proposed Design).....	56
Figure 39. Robotic Arm single-sheer joint.....	60
Figure 40. Revised Robotic Arm	60
Figure 41. ECOMAID Brush Compatible for iRobot Roomba	61
Figure 42. Bevel Gear Mechanism CAD Model.....	62
Figure 43. Wrist Rotation Mechanism CAD Model.....	62
Figure 44. 4-inch Lazy Susan	63
Figure 45. DC-DC Stepdown Regulator 6-40V into 1.2-36V 20A out	63
Figure 46. Maestro & VNH5019 Wiring.....	64
Figure 47. Arm & Brush Fully Assembled.....	64
Figure 48. Astraeus Wheel.....	67
Figure 49. Rocker-Bogie Assembly (Right)	68
Figure 50. Full Rocker Boggie Assembly.....	69
Figure 51. Shared dictionary setup in start_all.py for inter-process communication	72
Figure 52. Dual-process launch for Flask and control loop in start_all.py	72
Figure 53. Main control loop in main.py handling mode-based task delegation.....	73
Figure 54. Manual command and speed retrieval from shared memory in main.py	73
Figure 55. Task check and autonomous execution trigger in main.py	74
Figure 56. Command access functions in command_center.py using shared memory	74
Figure 57. Route in app.py to receive manual movement commands.	76
Figure 58. Speed update route in app.py for manual mode.	76
Figure 59. Local inclusion of nipplesjs in the HTML interface.	77
Figure 60. JavaScript function sending movement commands to Flask.....	77
Figure 61. Mode selection route in app.py.....	78

Figure 62. main.py triggers run_autonomy() when a task is detected.	79
Figure 63. Detection and buffering of April Tag data in visual_module.py.....	80
Figure 64. Smoothed average output of tag data to stabilize alignment.	81
Figure 65. Task sequence for "panel1" in autonomy.py, showing tag detection and alignment steps, with fallback to manual mode if either stage fails.	84
Figure 66. _set_single_motor() applies a direction and PWM value based on the requested speed.	85
Figure 67. Centralized drive command routing in motors.py for manual input handling.	86
Figure 68. Fine-grained motor control in autonomous tag tracking.	87
Figure 69. Voltage-to-distance conversion used to interpret raw Sharp sensor readings.	88
Figure 70. Exponential moving average (EMA) for smoothing noisy distance data.....	88
Figure 71. Emergency stop routine triggered when a nearby object is detected.	89
Figure 72. Function to trigger subroutine execution on the Pololu Maestro via serial command.	90
Figure 73. Home Tab here to show the starting point of the UI.	91
Figure 74. Select Operation Mode screen.....	92
Figure 75. Manual Mode Control Interface with Joystick after this paragraph.....	92
Figure 76. Autonomy Mode Selection Screen here.	93
Figure 77. Full Log Table View (All Levels)	94
Figure 78. Warnings Filtered View.....	94
Figure 79. Alerts Highlighted in Red.....	95
Figure 80. Load Older Log Button, no new messages were available at the time so no “New Message” Pop up.....	95
Figure 81. Key-mapping dictionary in failsafe_manual.py that defines manual control input using standard keyboard keys.	97
Figure 82. Panel Section 1 Cleaned	117
Figure 83. Panel Section 2 Cleaned	117

List of Tables

Table 1. Robotic Arm Engineering Requirements	6
Table 2. Navigation Engineering Requirements	7
Table 3. Drive System Specifications	8
Table 4. Cleaning Arm Specifications	9
Table 5. Power Module Specifications	10
Table 6. Product Comparison Table.....	18
Table 7. Mineral Composition of MGS-1	24
Table 8. Solar Panel Comparison	30
Table 9. List of Libraries Used.....	35
Table 10. Proposed Power Budget	48
Table 11. Comprehensive Power Budget.....	49
Table 12. Robotic Arm Servo Specifications.....	57
Table 13. Revised Robotic Arm Servo Specifications	57
Table 14. Spring Timeline.....	100
Table 15. Spring Gantt Chart (Proposal Phase)	102
Table 16. Senior Design (Proposed) Gantt Chart.....	103
Table 17. Spring Timeline.....	104
Table 18. Spring Gantt Chart	106
Table 19. Contributions Table.....	107
Table 20. Proposed Budget	108
Table 21. Final Comprehensive Budget.....	110

Chapter 1

Introduction

Summary

In this chapter, we introduce the Astraeus project, an autonomous prototype rover designed to clean solar panels on Mars using a 5-axis robotic arm. The motivation for this design stems from the critical need to maintain solar efficiency in extraterrestrial environments, where dust accumulation poses a significant threat to power generation. This chapter defines the problem, outlines the project's objectives, and presents the engineering specifications and block diagram of the system. It also compares Astraeus to existing solutions and describes the organization of the remainder of the report.

1.1 Project Overview

1.2 Problem Definition

1.3 Project Objectives

1.4 Motivation

1.5 Engineering Requirements & Specifications

1.6 Block Diagrams

1.7 Limitations

1.8 Comparison of Existing Products

1.9 Report Outline

1.1 Project Overview

Astraeus is an autonomous prototype rover developed to demonstrate the feasibility of robotic solar panel cleaning in conditions analogous to those found on the Martian surface. As solar energy remains one of the most practical and sustainable power sources for planetary missions, maintaining consistent panel efficiency is critical to ensuring mission success. However, on Mars, atmospheric dust storms and loose surface particles pose a continual threat to solar panel performance by reducing light absorption through fine dust accumulation.

The Astraeus project addresses this challenge by proposing a compact, AI-guided robotic system capable of autonomously identifying solar panels, navigating unstructured terrain, and executing cleaning operations using a precision-controlled robotic arm. The system is engineered to operate independently, without real-time human input, enabling reliable and repeatable maintenance in future off-world solar farms or mission-critical infrastructure.

Designed specifically for Earth-based testing, the prototype operates under realistic power, mobility, and sensing constraints. It utilizes MGS-1 Martian regolith simulant for dust-based testing and simulates extraterrestrial terrain challenges. The chassis features a rocker-bogie suspension system to support traversal over uneven ground. A Raspberry Pi 3 Model B serves as the main control platform, integrated with a HuskyLens AI camera for visual panel recognition and alignment.

All subsystems were developed with modularity and scalability in mind, enabling future upgrades and adaptation to more demanding mission environments. Astraeus serves not only as a proof-of-concept for solar panel cleaning robotics, but also as a foundation for the broader development of autonomous surface maintenance technologies essential for sustained human presence on Mars and beyond.

1.2 Problem Definition

As humanity prepares for long-term exploration and eventual settlement beyond Earth, the establishment of reliable energy infrastructure becomes essential for sustaining surface operations. On Mars, solar power is the most practical and scalable energy source due to its availability and ease of deployment. However, the Martian environment presents a major challenge: fine dust particles, carried by winds and storms in the planet's low-pressure atmosphere, which settle on solar panels and gradually degrade their efficiency.

Without an effective cleaning strategy, this accumulation can severely reduce power output over time, leading to energy shortages that compromise critical systems such as life support, communications, and scientific instruments. While manual cleaning by astronauts is theoretically possible, it is not sustainable. It would require extensive EVA time, present operational risks, and divert personnel from higher-priority mission objectives.

To address this issue, the Astraeus project was conceived as a robotic solution for autonomous solar panel cleaning on Mars. Astraeus is a mobile rover equipped with onboard navigation, panel detection capabilities, and a precision-controlled dual-brush cleaning mechanism. It can autonomously locate solar panels, align itself using onboard sensors and vision systems, and remove surface dust without human intervention.

By automating this critical maintenance task, Astraeus significantly improves the reliability and longevity of solar energy systems in extraterrestrial environments. This not only reduces the risk to mission personnel but also supports the scalability of solar infrastructure for future Martian habitats, research stations, and equipment deployments.

1.3 Project Objectives

The primary objective of Astraeus is to enhance the sustainability of long-duration Mars missions by maintaining the performance of solar energy infrastructure. Solar power is projected to be the primary energy source for off-world operations, and uninterrupted energy absorption is critical to sustaining life support systems, communication networks, and scientific instrumentation on the Martian surface.

Astraeus addresses the operational challenge of dust accumulation on solar panels, one of the primary limitations of previous solar-powered Mars missions. Designed as an autonomous rover, Astraeus actively monitors and cleans solar arrays without requiring astronaut intervention. This reduces risk and workload for human crews while improving energy system reliability and overall mission productivity.

The rover demonstrates autonomous cleaning functionality in an Earth-based test scenario using MGS-1 Martian regolith simulant and terrain conditions selected to resemble Martian surface challenges. Astraeus uses onboard vision systems to locate solar panels, aligns itself using a combination of sensor feedback and QR code positioning, and activates a 5-axis robotic arm to perform cleaning using dual counter-rotating soft brushes. Once cleaning is

complete, the rover autonomously returns to its original location. Throughout the operation, *Astraeus* functions independently, without the need for real-time external control or supplemental power.

Additionally, the project integrates a robust system logging process that collects diagnostic data during each cleaning cycle. This data informs future optimization of panel maintenance strategies and supports the development of scalable robotic infrastructure for future human settlement.

Astraeus is built with modularity and sustainability in mind. From its chassis and servo layout to its battery system and software architecture, the rover is designed to support iterative development and component-level upgrades. With a total system staying within budget, the project demonstrates that essential support systems for space infrastructure can be prototyped affordably, providing a foundation for future autonomous maintenance robotics on Mars.

1.4 Motivation

The success of planetary exploration missions depends heavily on the availability of reliable and sustainable power systems. Among these, solar energy has emerged as the preferred source for surface operations because of its accessibility, scalability, and simplicity of deployment. However, maintaining a consistent energy output remains a significant challenge on Mars, where fine dust particles regularly settle on solar panels. This buildup gradually reduces panel efficiency and threatens the continuity of mission operations.

This challenge has already affected real missions. NASA's *Opportunity* rover, which operated well beyond its expected lifespan, ultimately ceased functioning after a global dust storm covered its solar panels. Without a method to clean them autonomously, the rover lost power and was unable to recover. This event marked the end of a historic mission not because of mechanical failure but due to the absence of a maintenance system to sustain its energy supply.

As missions to Mars and other planetary bodies increase in complexity and duration, the need for autonomous maintenance capabilities becomes increasingly important. Dependence on solar energy will not be viable unless systems are in place to ensure continuous performance. Manual servicing is often impractical or impossible due to time constraints, safety concerns, and the limited availability of astronaut resources.

The Astraeus project was developed in response to this critical need. It is based on the idea that autonomous robotic systems must handle routine operational tasks to support scalable and sustainable exploration. By addressing the specific issue of solar panel dust accumulation, Astraeus serves as an early demonstration of robotic infrastructure that can support future extraterrestrial missions. Its development is a first step toward enabling continuous surface operations for research, habitation, and long-term exploration beyond Earth.

1.5 Project Requirements & Specifications

The engineering requirements and specifications presented here serve to narrow the project's scope and sharpen the focus of the Astraeus design efforts. By setting clear, measurable criteria, these requirements provide essential guidance and boundaries that keep the project targeted and manageable while ensuring alignment with its core objectives. This focused framework supports informed decisions in component selection and system design, leading to a well-defined and achievable engineering solution.

1.5.1 Engineering Requirements

The engineering requirements for Astraeus define the functional and performance expectations necessary for effective solar panel cleaning on Mars. These requirements ensure that the rover can efficiently navigate, clean solar panels, and log performance data while maintaining system reliability and energy efficiency. The requirements are categorized into two key subsystems: the robotic arm cleaning system and the navigation system. Each requirement is assigned a priority level, high for critical functions and medium for supporting capabilities, along with a corresponding verification method.

Robotic Arm (Cleaning System)

The robotic arm is responsible for executing the cleaning function of the Astraeus rover and was designed to operate autonomously with precision and control. Its performance is defined by specific functional and verification requirements to ensure effective dust removal and reliable data collection during testing. Table 1 below outlines the engineering requirements established for the robotic arm and its integrated cleaning system.

Table 1. Robotic Arm Engineering Requirements

Robotic Arm (Cleaning system)		
Level	Requirements (Astraeus Shall...)	Verification
High	Utilize a variable speed cleaning brush with precise control over speed and pressure to effectively clean surfaces.	Demonstrate the brush operating at steady speeds and document the cleaning effectiveness at each speed using controlled amounts of debris on the testing surface.
Medium	Document the cleaning of the solar panel by logging the data for users to analyze the performance of the rover.	The System must log all data in a structured format, accessible for further analysis.

Navigation Engineering Requirements

The navigation system enables Astraeus to operate autonomously by detecting and avoiding obstacles, identifying solar panel locations, and returning to its origin after completing a cleaning cycle. This subsystem integrates sensor data and visual recognition to support decision-making in real time. The engineering requirements for navigation were developed to ensure the rover can perform these tasks reliably under simulated mission conditions. Table 2 outlines the specific performance expectations and verification methods for the navigation system.

Table 2. Navigation Engineering Requirements

Navigation Engineering Requirements		
Level	Requirements (Astraeus Shall...)	Verification
High	Autonomously determine its own path to solar panels while avoiding obstacles.	Conduct navigation test where the rover reaches its destination without collision in test trials.
	Detect larger obstacles up to 40cm without running into them without human intervention.	Validate obstacle detection by testing against various obstacle sizes and distances.
	Recognize visual cues such as QR codes, distinct shapes, or colored markers to identify target zones.	Position various April Tag markers for the dedicated panel site and base position. Run various tests where Astraeus must correctly identify and navigate towards each unique marker using its on board vision system.
Medium	Return to the starting position without any human intervention.	Conduct various tests, verify Astraeus successfully navigates back to its start point after completing its cleaning task with no manual correction.

1.5.2 Engineering Specifications

The Astraeus project is designed to meet a precise set of engineering specifications that ensure optimal performance and reliability in its intended application. These specifications serve as the foundational requirements guiding the selection of components, system architecture, and overall design choices. The following table outlines the detailed specifications for key aspects such as mobility, power, sensing, and manipulation capabilities. Following this, a comprehensive overview of the selected parts will demonstrate how each component aligns with and fulfills these stringent requirements to create a cohesive and effective robotic system.

Table 3. Drive System Specifications

Drive System Specifications			
Module	Component	Engineering Specifications	Justification
Electro-Mechanical Module	Motors (x6)	<ul style="list-style-type: none"> - 12V DC - Peak Current: 15 A - Gearbox Output: ~100 RPM 	<ul style="list-style-type: none"> - High torque output to support rover mobility and terrain traversal. - Geared motors reduce speed for increased torque.
	Motor Drivers	<ul style="list-style-type: none"> - Rated for 12V motor operation. - Supports bidirectional speed control for 6 motors. 	<ul style="list-style-type: none"> - Enables independent control of left/right motor groups. - Required to safely and efficiently drive multiple motors under load.
Control Module	Microcontroller	<ul style="list-style-type: none"> - Minimum 12 PWM-capable pins - Supports UART, I²C for peripherals 	<ul style="list-style-type: none"> - Required for precise motor control and servo actuation. - Must support communication with sensors and AI camera.
	IR Analog Distance Sensor	<ul style="list-style-type: none"> - Voltage: 5V - Current Draw: ~30 mA - Range: up to 31 cm 	<ul style="list-style-type: none"> - Used for edge detection and alignment with solar panel. - Enables basic obstacle avoidance and correction.
	AI camera	<ul style="list-style-type: none"> - Supports onboard object tracking via AI - Compatible UART/I²C 	<ul style="list-style-type: none"> - Captures and analyzes solar panel position for autonomous alignment. - Enables visual feedback loop for positioning system.

Table 4. Cleaning Arm Specifications

Cleaning Arm			
Module	Component	Engineering Specifications	Justification
Electro-Mechanical Module	Servo	<ul style="list-style-type: none"> - Torque: 25 kg - Voltage Range: 4.8V – 7.4V - Stall Current: ~3.5 A 	<ul style="list-style-type: none"> - Strong enough for base rotation. - Uses PWM control. - IP67 rated.
	Servo	<ul style="list-style-type: none"> - Torque: 45 kg - Voltage Range: 6.0 V – 8.4 V - Stall Current: ~3.5 A 	<ul style="list-style-type: none"> - Strong enough for shoulder rotation. - Uses PWM control. - IP67 rated.
	Servo	<ul style="list-style-type: none"> - Torque: 45 kg - Voltage Range: 6.0 V – 8.4 V - Stall Current: ~3.5 A 	<ul style="list-style-type: none"> - Strong enough for elbow rotation. - Uses PWM control. - IP67 rated.
	Servo	<ul style="list-style-type: none"> - Torque: 25 kg - Voltage Range: 4.8V – 7.4V - Stall Current: ~3.5 A 	<ul style="list-style-type: none"> - Strong enough for wrist tilt. - Uses PWM control. - IP67 rated.
	Servo	<ul style="list-style-type: none"> - Torque: 55g - Voltage Range: 4.8V – 7.4V - Stall Current: ~3.5 A 	<ul style="list-style-type: none"> - Strong enough for wrist rotation. - Uses PWM control. - IP67 rated.
	Micro-Metal Gear Motor	<ul style="list-style-type: none"> - Gear Ratio: 75:1 - Voltage Range: 3V – 9V - Stall Current: ~1.6 A 	<ul style="list-style-type: none"> - Provide sufficient torque to rotate cleaning brush through dust/debris. - Compact size to fit within brush housing.
Control Module	Servo Controller	<ul style="list-style-type: none"> - 6 servo channels (12-bit resolution) - Logic voltage: 5V - Control via USB, UART (TTL) 	<ul style="list-style-type: none"> - Enables precise control of up to 6 servos. - USB and UART options allow easy integration with microcontrollers or PC. - Built-in scripting for autonomous sequences without a host controller.

Table 5. Power Module Specifications

Power Module Specifications			
Module	Component	Engineering Specifications	Justification
Power Module	Drive System Battery	12V 30Ah	- Provides sufficient current and capacity to operate the full drive system for at least 1 hour.
	Voltage regulator (Step Down)	12V to 6V regulator	- Powers subsystems requiring 6V.
	Voltage regulator (Step Down)	12v to 5V regulator	- Powers 5V logic devices like sensors, camera modules, and microcontroller.

1.6 System Block Diagrams

The following block diagrams illustrate the major subsystems of Astraeus and how they interconnect within the rover's control architecture. These visual tools clarify data and power flow, assist in modular design, and serve as technical references during integration and troubleshooting. Each diagram focuses on a different aspect of the system, including full system architecture, power distribution, electromechanical control, and computer vision.

1.6.1 Full System Architecture

The full system block diagram provides a high-level overview of all major subsystems and how they interface with the central control unit. At the core of the system is the Raspberry Pi, which handles communication, decision-making, and coordination across modules. Inputs from sensors, including the HuskyLens AI camera and limit switches, feed into the Raspberry Pi for processing. Outputs are sent to motor drivers, servo controllers, and actuators, enabling autonomous movement and task execution. The block diagram clearly separates logical and physical connections, showing how power and control signals are routed to different components.

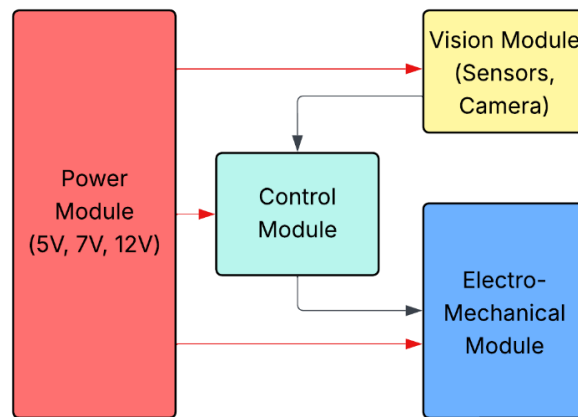


Figure 1. Full System Block Diagram

1.6.2 Power Distribution

The power block diagram focuses specifically on how energy is distributed throughout the rover. A 12V 30Ah lithium iron phosphate battery serves as the main power source. High-current loads such as DC motors are driven directly from this source through VNH5019 motor drivers. Buck converters are used to step down voltage for components that require lower operating levels, such as the Raspberry Pi (5V), servo power rail (7V), and ADCs or logic-level interfaces (3.3V or 5V). The diagram emphasizes safe current routing and system protection, including fuse points and rail separation.

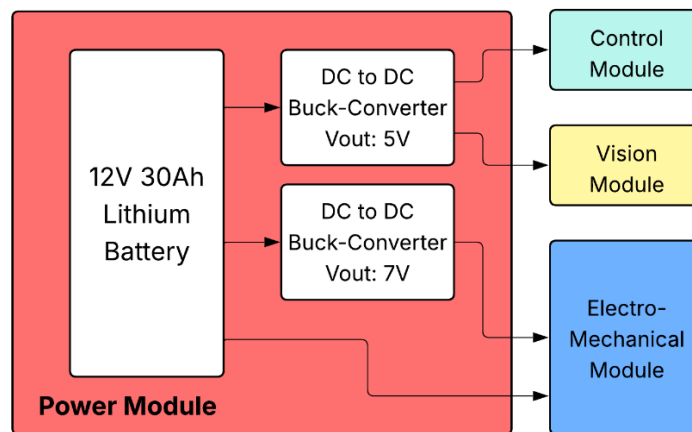


Figure 2. Power Distribution Block Diagram

1.6.3 Electromechanical Subsystem

This diagram isolates the electromechanical module of Astraeus, showing how the drive system and robotic arm are managed. The rocker-bogie suspension is powered by six DC motors grouped into left and right sets, each set controlled by a dedicated VNH5019 motor driver. The 5-axis robotic arm is controlled via a Maestro servo controller, which receives PWM signals from the Raspberry Pi and draws power from an independent 7V rail. The modular separation of drive and manipulation components supports parallel development and simplifies troubleshooting.

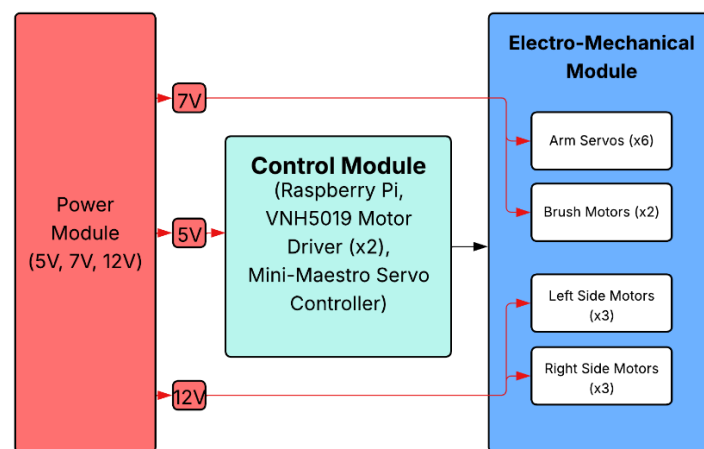


Figure 3. Electromechanical Block Diagram

1.6.4 Vision & Sensor Integration

The vision block diagram details how sensor inputs are integrated into the rover's control logic. The HuskyLens AI camera is the primary vision sensor, capable of recognizing objects, QR codes, and shapes. It communicates with the Raspberry Pi I2C, depending on the selected configuration. Additional digital and analog sensors, including Hall effect switches and limit switches, feed into an ADC or directly into GPIO pins on the Raspberry Pi. This diagram supports debugging of environmental perception and highlights how Astraeus navigates and responds to its surroundings.

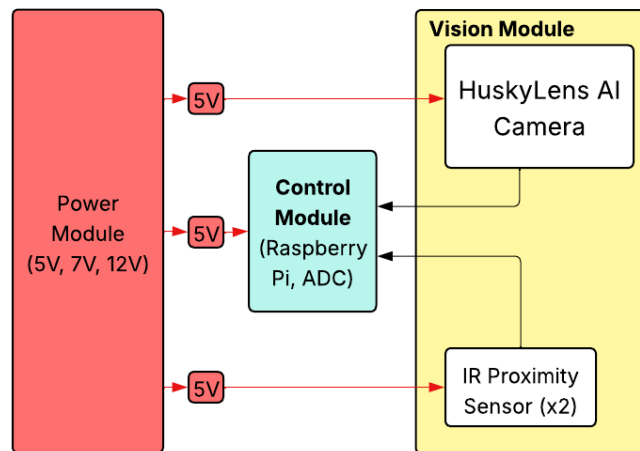


Figure 4. Vision & Sensor Module Block Diagram

1.7 Limitations

The Astraeus prototype was intentionally scoped to balance both ambition and feasibility. To be ensured the project remained achievable within the available time, budget, and resource constraints. The limitations outlined below were not merely obstacles, but deliberate design boundaries established to focus on core functionality, proof of concept, and our project timeline.

1.7.1 Scale and Functionality

Astraeus was designed as a scaled-down prototype to validate key engineering concepts including rocker-bogie suspension, modular power systems, and solar panel cleaning. It does not represent the full mechanical scale, ruggedness, or redundancy required for deployment on Mars. The prototype focused on demonstrating subsystem integration rather than simulating the exact environmental conditions of extraterrestrial operation.

1.7.2 Component Selection

To maintain affordability and accessibility, the team selected commercial off-the-shelf (COTS) components, including hobby-grade servos, consumer microcontrollers, and non-radiation-hardened sensors. While sufficient for a terrestrial prototype, these parts lack environmental endurance, fault tolerance, and precision of space-grade electronics.

1.7.3 Autonomy & AI Capabilities

Full AI-based navigation was outside the project’s scope. Instead, Astraeus uses QR code detection for identifying solar panel targets and Sharp IR distance sensors to align precisely 10 cm away from the panel before initiating cleaning. This control sequence provides reliable behavior in a controlled test setting but does not generalize to unstructured exploration, SLAM-based mapping, or adaptive routing in dynamic environments.

1.7.4 Panel Orientation Constraint

To ensure mechanical stability and simplify cleaning arm operation, Astraeus was limited to cleaning panels mounted at a fixed 45-degree angle. This orientation was chosen based on physical testing constraints and recommendations from a NASA solar array specialist, who affirmed that this is a reasonable starting point for early-stage prototyping. The specialist noted that future Martian solar farms may include panels with variable orientations or sun-tracking capabilities and suggested designing for cleaning adaptability in future versions [A].

(See Appendix A for the full email excerpt from NASA.)

1.7.5 Testing Environment

Terrain testing was performed on standard Earth surfaces with uneven obstacles. Although the team obtained 1 kg of MGS-1 regolith simulant, full environmental testing in UCF’s Martian soil pit was not completed due to a cost of \$1200, which exceeded the project budget. The cleaning brush was tested on MGS-1 applied to a test panel, but traction and suspension behavior on true regolith-like terrain remains unverified.

1.7.6 Power and Runtime Constraints

The estimated power consumption was based on the final comprehensive power budget, and the system was powered by a 12V, 30Ah LiFePO₄ battery. However, extended runtime tests beyond two hours were not performed. Testing focused on short-duration functional sequences. Battery degradation, thermal behavior, and peak surge response were not assessed under prolonged use.

1.7.7 Communication

Astraeus uses a local server configuration hosted on the onboard Raspberry Pi, which serves a custom web interface for system control and monitoring. This architecture allows for direct

interaction through any device connected to the same local network, either via a hotspot hosted by the Pi itself or an external Wi-Fi access point. While this method is efficient for short-range testing and demonstration, it limits the rover's operational range to areas with controlled wireless access.

This setup does not support long-range wireless communication, remote telemetry, or autonomous data uplink capabilities such as those required in actual planetary missions. For this reason, real-time communication and system diagnostics were restricted to line-of-sight operation within a lab or field test environment. Implementing true remote telemetry would require additional hardware such as LoRa, cellular, or satellite modules, which were beyond the scope of this project.

1.8 Comparison of Existing Products

Astraeus represents a significant departure from previous Mars rovers by focusing on infrastructure support rather than scientific exploration. Traditional Mars rovers, such as Opportunity (MER-B) and Curiosity, were primarily designed for planetary research and were equipped with scientific instruments to analyze Martian soil, rocks, and atmospheric conditions. While these missions provided invaluable scientific insights, they were not intended to support or maintain infrastructure critical to human survival. In contrast, Astraeus offers a practical, robotic solution to a real operational challenge: dust accumulation on solar panels, which has historically limited the viability of solar energy on Mars.

Mars rovers like Opportunity (MER-B) (Fig. 5) relied on solar panels for power, but ultimately suffered due to dust buildup, which reduced energy intake and led to mission failure [1]. Despite its robust design and autonomous capabilities, Opportunity had no built-in cleaning system, and the rover solely depended on unpredictable Martian wind events to clear its solar panels, which ultimately limited its operational lifespan. This demonstrated a major flaw in sustaining solar-powered operations on Mars.

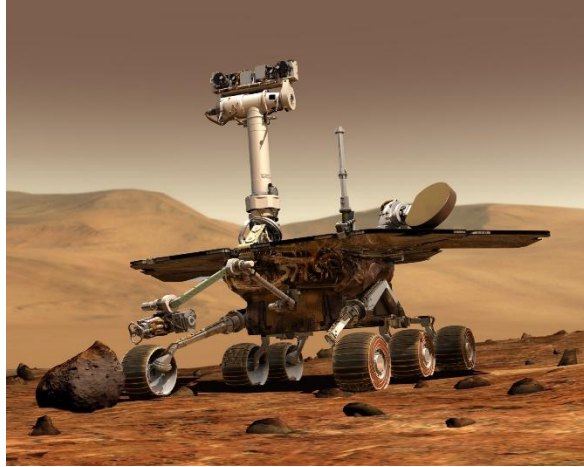


Figure 5. Opportunity Rover

When up against Curiosity Rover (Fig. 6), the comparison highlights a different philosophy. Curiosity avoided this issue altogether by switching to a Radioisotope Thermoelectric Generator (RTG) for power, eliminating the need for solar panels altogether.



Figure 6. Curiosity Rover Radioisotope Thermoelectric Generator

While effective, RTGs are not scalable or ideal for manned missions due to limited fuel availability, safety concerns, and their complexity. This shift away from solar was not because solar is unviable but because we lacked a way to maintain solar systems in Martian conditions. This is where Astraeus becomes essential. While it does not generate power itself, Astraeus is designed to maintain large-scale solar farms on Mars, which would be critical to powering habitats, research stations, and equipment in future human missions. Its dual rotating brush arms

with static charge assist actively remove dust from panel surfaces, solving the exact problem that ended solar-reliant missions like Opportunity. Astraeus also uses a rocker-bogie suspension system, a proven design used on Opportunity and Curiosity, for navigating rocky, uneven Martian terrain. However, unlike its multi-million-dollar predecessors, Astraeus is built at a small-scale prototype cost of just \$770, demonstrating that functional, effective solutions can be developed cost-efficiently and scaled up for future use.

In comparison with commercial solutions like the SolarCleano F1 (Fig. 7). Astraeus also holds a distinct advantage for specific use cases. The SolarCleano F1 is a high-end product designed for large-scale solar farms, costing between \$30,000 to \$50,000+ and typically requiring manual or remote control. Astraeus fills a unique niche by adapting space-proven mobility and navigation systems to create an autonomous Martian cleaning robot, specifically for enabling sustainable solar infrastructure.

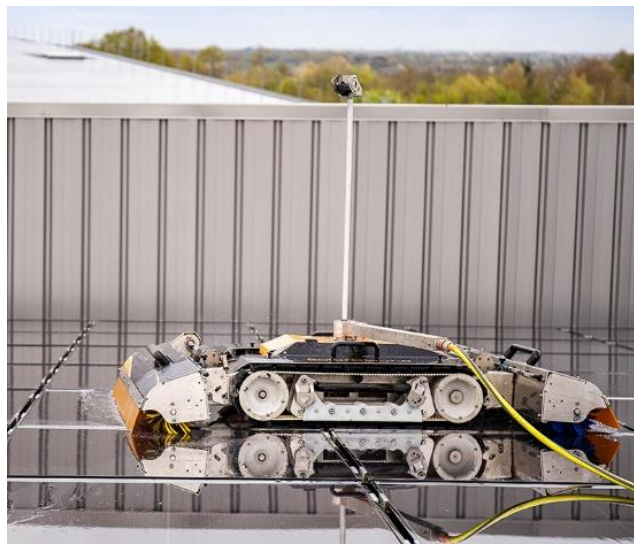


Figure 7. SolarCleano F1

Astraeus is a forward-thinking solution to one of the most pressing challenges facing future Martian exploration. By addressing the issue of dust accumulation on solar panels, Astraeus enables the long-term viability of solar infrastructure, which is essential for powering habitats, equipment, and life-support systems in manned missions. Its low cost, proven mobility system, and autonomous cleaning capability position it as a critical piece of future Mars mission success.

Table 6. Product Comparison Table

Astraeus Comparison Table					
System	Power Source	Mobility	Cost	Ability to Clean Solar Panels	Drive System
Astraeus	12V 30Ah Lithium-ion battery	Rocker-bogie suspension with obstacle avoidance	~\$770 (prototype)	Dual rotating brush arm with static charge assist	6 × 12V DC geared motors (skid steering)
Opportunity (MER-B)	Triple-junction solar panels (~140W init.)	Rocker-bogie suspension, autonomous navigation	~\$400 million (NASA)	Relied on Martian wind/dust devils	6 independently driven motors
Curiosity Rover	Radioisotope Thermoelectric Generator (RTG)	Rocker-bogie suspension with enhanced navigation	~\$2.5 billion (NASA)	Using nuclear power, no need for panel cleaning	6-wheel independent drive with steering motors
SolarCleano F1	Swappable battery pack (3–4 hr runtime)	Modular wheeled chassis, remote/manual control	~\$30K–\$50K+ (estimate)	Microfiber roller brush with adjustable pressure	Electric wheeled drive (human or remote)

1.9 Report Structure

This report is structured to comprehensively document the design, development, testing, and analysis of the Astraeus autonomous solar panel cleaning rover. The report is organized into five separate chapters, each addressing a critical stage of the engineering process, along with appendices containing supplemental materials, datasheets, code, and reference documentation.

Chapter 1 – Introduction: This chapter outlines the project’s purpose, motivation, engineering requirements, limitations, and a comparison of Astraeus with existing Mars

and terrestrial systems. It also provides system block diagrams and discusses the rationale for each major design decision.

Chapter 2 – Background Research: This section presents the technical and scientific research that informed the design of Astraeus, including studies on rocker-bogie suspension, Martian regolith properties, robotic arm mechanisms, vision systems, and the control platform. It also includes the power budget and algorithmic flowcharts that guided the system's logic.

Chapter 3 – Contributions: Chapter 3 details the individual contributions made to each subsystem of Astraeus, including the chassis design, cleaning arm, rocker-bogie integration, software development, and hardware-software interface. It breaks down each design and assembly process along with programming considerations and testing outcomes.

Chapter 4 – Non-Technical Issues: This chapter addresses the broader context of the project, including scheduling and planning, budgeting, environmental considerations, health and safety, ethics, and sustainability.

Chapter 5 – Conclusion: The final chapter summarizes project outcomes and discusses system performance, including autonomous cleaning success and robotic arm functionality. It also offers suggestions for future enhancements such as self-docking stations and panel efficiency monitoring.

Appendices: The appendices contain relevant datasheets, email correspondence, Maestro scripts, source code, and regulatory guidelines referenced throughout the report. These materials support transparency, reproducibility, and technical depth.

Chapter 2

Background Research

Summary

This chapter provides an in-depth discussion of the background research and technical components involved in the development of Astraeus. It begins with an explanation of the rover's mechanical platform, including the rocker-bogie suspension system and drive architecture. The chapter then examines the robotic arm and cleaning mechanism used to interact with the panel surface, followed by the sensors and AI-based vision system used for object detection and task alignment. Control strategies and software implementation are also covered, including the rationale for selecting Python and a Raspberry Pi as the main programming and control platform. Finally, a detailed power budget is presented, demonstrating that the selected 12V 30Ah battery provides sufficient energy to support the rover through multiple operational cycles. Together, these sections form a comprehensive overview of the design, logic, and feasibility behind the autonomous operation of Astraeus.

2.1 Rocker-Bogie Suspension System

2.2 Martain Regolith Research

2.3 Drive System & Motor Control

2.4 Robotic Arm & Cleaning Mechanism

2.5 Sensors & Vision

2.6 Control Platform and Programming Language

2.7 Navigations & Task Execution Algorithms

2.8 Power Budget

2.1 Rocker-Bogie Suspension System

The rocker-bogie suspension system is a foundational mechanism of Astraeus that was originally developed by NASA's Jet Propulsion Laboratory (JPL) for the Sojourner rover, which successfully landed on Mars as part of the Pathfinder mission in 1997. Since then, it has been used in every Mars surface mission involving rovers, including Spirit, Opportunity, Curiosity, and Perseverance. This system was selected for Astraeus because of its exceptional terrain adaptability, passive stability, and proven heritage in extraterrestrial environments.

Unlike conventional suspensions that use springs or shock absorbers, the rocker-bogie system distributes the rover's weight equally across six wheels, connected by rocker arms and bogie links. These components are joined by a passive differential that allows each side of the suspension to move independently. This design offers two very critical advantages. First, the pressure each independent wheel applies to the ground is balanced or equilibrated [5], which is especially important on soft or loose terrain like Martian regolith. Excessive pressure from any single wheel could cause it to sink into the surface, reducing traction and increasing power demand. Second, in rocky or uneven conditions, all six wheels remain in contact with the ground and under load. This maximizes traction and propels the vehicle forward, allowing it to climb over obstacles more effectively [6] (Fig. 8).

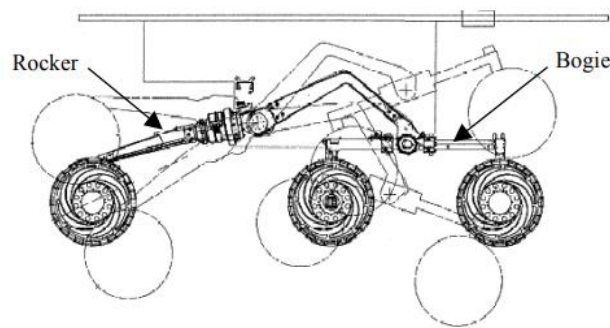


Figure 8. Rocker-bogie Suspension System Range of Contact

2.1.1 3D Printing for Rocker Bogie Components

3D printing plays a pivotal role in the fabrication of Astraeus' rocker-bogie suspension system, aligning with industry trends in aerospace and planetary exploration. Research into 3D printing in space has demonstrated its feasibility for producing structural components using high-performance materials such as titanium, carbon fiber-reinforced polymers (CFRP), PEEK, and ULTEM. These materials offer excellent mechanical strength, thermal stability, and radiation resistance, making them suitable for long-

term deployment in harsh extraterrestrial environments. Titanium has been explored for in-situ additive manufacturing due to its high strength-to-weight ratio and corrosion resistance, while CFRPs are favored for their durability and low mass in aerospace structures.

Although such advanced materials are ideal for actual space missions, the Astraeus prototype is designed for Earth-based testing and demonstration. As a result, PETG (polyethylene terephthalate glycol-modified) was selected for the fabrication of the rocker-bogie system and other structural elements. PETG offers an ideal compromise between mechanical performance, printability, and cost. Compared to more specialized aerospace polymers like PEEK or ULTEM, PETG can be printed with standard FDM 3D printers without requiring specialized high-temperature equipment. It provides better impact resistance than PLA and improved dimensional stability over ABS, which is critical for ensuring consistent geometry in mechanically loaded parts such as rocker arms, joint housings, and suspension brackets.



Figure 9. Overture PETG 1.75mm 3D Printer Filament

To ensure the durability of the rocker-bogie components under testing conditions, all PETG parts were printed using a reinforced configuration: 4 perimeter walls and 45% infill. This setting strikes a balance between strength and material usage, allowing the printed components to handle torsional and impact stresses experienced during navigation testing. By using PETG, customized components specific to the Astraeus rover's rocker-bogie geometry were able to be rapidly printed and revised throughout the project's development. This flexibility proved invaluable during prototyping and assembly, allowing parts to be revised and reprinted as the design evolved. Although PETG would not be suitable for direct space

deployment due to limitations in thermal and radiation tolerance, it serves effectively in this phase of the project by enabling efficient, low-cost development of functional prototypes.

2.1.2 Implementing Rocker-Bogie

For Astraeus, the rocker-bogie system was chosen to provide these same mechanical benefits in a smaller-scale Earth-based prototype. The rover's mobility platform was constructed using 1-inch PVC pipe for the suspension arms and 3D-printed PETG components for joints and wheel mounts. This lightweight, modular construction replicates a rocker-bogie system's mechanical behavior while remaining within fabrication and budget constraints.

The stability by using the rocker-bogie system is essential for Astraeus' ability to clean solar panels. Because the rover must approach and clean solar panels mounted at a 45-degree angle, the suspension ensures that it remains level and balanced during arm deployment and cleaning. All six wheels stay grounded even when the front of the rover rises slightly to reach the panel, minimizing vibration or drift that could interfere with precision cleaning. Below is the complete CAD model within Fusion 360 of the rocker-boogie implemented onto the Main Chassis (Fig. 10).

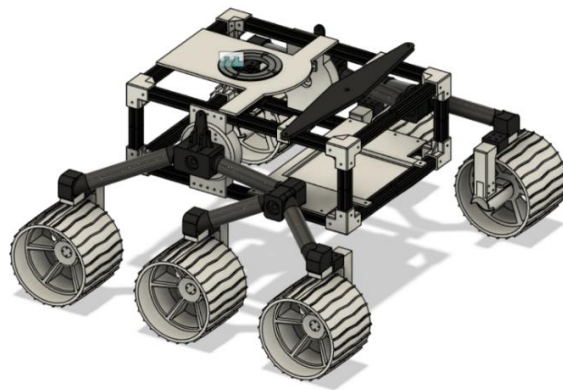


Figure 10. CAD model of the rocker-bogie suspension

This design incorporates the 1-inch PVC suspension arms, PETG 3D-printed joints, and firm structural layout necessary to maintain balance and articulation across uneven terrain. While this specific chapter focuses on the background and justification of the suspension system, the specific construction process with details on material selection, joint design, and chassis integration is discussed in greater detail within Chapter 4.

2.2 Martian Regolith Research

To ensure that the rocker-bogie suspension system would perform effectively under Martian terrain conditions, research and testing were conducted using MGS-1 (Fig. 11), a high-fidelity Martian regolith simulant developed by the Exolith Lab at the University of Central Florida. Martian regolith is known for its fine particulate size, angular grains, high abrasiveness, and electrostatic properties, all of which create mobility and durability challenges for planetary rovers.



Figure 11. Martian Regolith

The MGS-1 simulant closely replicates the geotechnical properties of Martian soil, including particle sizes between 0.02 mm and 1 mm, angular basalt grains, and a bulk density consistent with NASA site data. It is chemically formulated based on Mars Global Surveyor findings and includes oxides such as SiO_2 , Fe_2O_3 , MgO , and CaO . The complete mineral composition of the MGS-1 simulant is shown in Table 7.

Table 7. Mineral Composition of MGS-1

Mineral	Wt. %
Plagioclase	27.10%
Glass-rich basalt	22.90%
Pyroxene	20.30%
Olivine	13.70%
Mg-sulfate	4%
Ferrihydrite	3.50%
Hydrated silica	3%
Magnetite	1.90%
Anhydrite	1.70%
Fe-carbonate	1.40%
Hematite	0.50%

Although the Exolith Lab offers access to a full-scale regolith testing pit, the quoted cost for formal testing was approximately \$1,200, which exceeded the project's budget. Instead, the 1 kilogram of MGS-1 simulant was purchased through Space Resource Technology and used for localized testing of the brush cleaning mechanism. The simulant was spread across an acrylic solar panel surface to evaluate how effectively the rotating brush could remove particulate buildup under realistic Martian dust conditions.

It is also important to mention that though mobility evaluation in the Exolith Lab's regolith test pit was originally planned, the cost for formal access and supervision was out of the projects budget. As a result, the rocker-bogie system was instead tested on local terrain simulating uneven Martian-like conditions using natural Earth ground. While this does not replicate the mechanical drag or cohesion of Martian soil, it allowed the team to verify general suspension behavior and climbing ability within budgetary limits.

2.3 Drive System & Motor Control

This section details the selection, arrangement, and control of the drive motors, the integration of high-current motor drivers, and how the system can achieve reliable movement. The design emphasizes durability, torque efficiency, and control simplicity, ensuring that Astraeus can perform repeated traversal and obstacle negotiation tasks with precision and stability.

2.3.1 Drive System

The drive system of Astraeus consists of six Geartisan DC 12V motors (Fig. 12) [7] arranged in three pairs, each driving a wheel within the rocker-bogie suspension. Each motor operates at approximately 100 RPM under no load and can deliver the torque necessary to navigate rough terrain. The six motors are organized into two groups of three motors wired in parallel, with each group driving one side of the rover's rocker-bogie suspension. Wiring motors in parallel ensures uniform voltage across each motor while summing their current demands, providing synchronized torque and smooth power distribution across the wheels.



Figure 12. Geartisan DC 12V 100RPM Gear Motor

2.2.2 Motor Control

Control and power delivery to the motors are managed by two VNH5019 motor driver carriers, each handling one group of three motors. VNH5019 drivers are well-suited for this application, offering up to 12 A of continuous current per driver and built-in safety features such as thermal shutdown and overcurrent protection [B]. These drivers enable bidirectional control and speed modulation through PWM signals, which allows Astraeus to perform precise movements, including smooth acceleration, deceleration, and direction changes [13].

(See Appendix B for VNH5019 Datasheet)

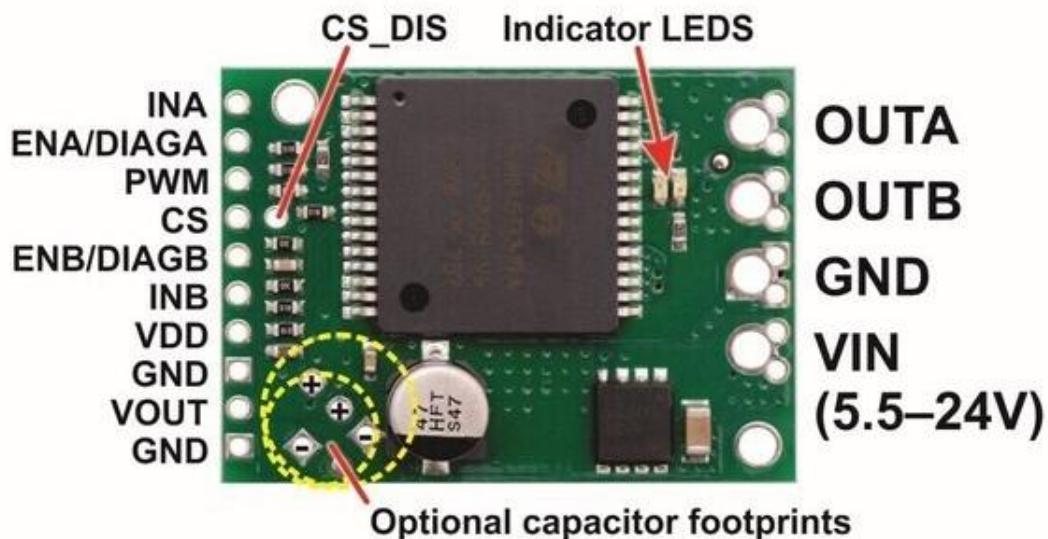


Figure 13. VNH5019 Motor Driver Carrier

2.3 Robotic Arm & Cleaning Mechanism

Astraeus has a custom-designed 5-axis robotic arm that enables the rover to be able to perform its panel-cleaning tasks. The arm is engineered to extend, position, and apply a dual spinning brush mechanism to remove Martian regolith from solar panel surfaces.

The arm consists of five axes: base rotation, shoulder lift, elbow movement, wrist pitch, and wrist yaw. These movements are powered by a combination of 25kg-DS3225 (Fig. 14) and ANNIMOS 45 kg high-torque digital servos (Fig. 17), selected to meet the torque requirements under load. A lightweight 55 g servo (Fig. 16) is dedicated to controlling the left and right tilt of the wrist on the cleaning brush.



Figure 14. 25kg-DS3225 High-torque Digital Servo



Figure 15. ANNIMOS 45 kg High-torque Digital Servo



Figure 16. 55g High-torque Digital Servo

The wrist-mounted cleaning tool features two N20 75:1 12V micro gear motors [9] (Fig. 17), which provide the rotational force needed to spin the brush effectively. These motors are lightweight yet powerful enough to handle dust removal without adding significant inertia to the arm's end effector. Their compact form factor allows for a balanced and efficient design suitable for dynamic arm movements.



Figure 17. N20 75:1 12V Micro Gear Motor

2.3.1 Arm Control and Integration

Servo control for the robotic arm is handled by a 12-channel Mini Maestro servo controller [10] (Fig.18), which communicates via USB with the Raspberry Pi. This setup allows precise, multi-axis control of all five servos using pre-programmed motion sequences or real-time commands. Maestro's built-in support for acceleration and speed parameters enables smooth motion transitions and safe, synchronized joint operation during panel cleaning tasks.

(See Appendix C for Maestro Servo Controller user guild.)



Figure 18. Mini Maestro 12-Channel USB Servo Controller

The dual N20 micro gear motors that spin the brush are powered by a dedicated VNH5019 motor driver, independent from the Raspberry Pi. This driver is configured for continuous operation during cleaning sequences and is activated through a separate control system, which may include manual switches or preset logic depending on test conditions. By isolating brush motor control from the main processor, the system avoids drawing excessive current through the Pi and improves fault tolerance and modularity. This set up ensures reliable control of both the robotic arm and the cleaning mechanism. By separating the high-current brush motor driver from the logic-level servo system, Astraeus maintains safe and consistent operation. The arm and brush work in tandem to complete autonomous cleaning cycles, demonstrating the rover's ability to perform maintenance tasks critically for long-duration planetary exploration.

2.3.2 Cleaning Testing

Astraeus is intended to function as a prototype for solar panel maintenance on Mars, where dust accumulation has historically degraded the performance of solar-powered missions. To demonstrate cleaning functionality in an Earth-based environment, a Solartech Power W-Series SPM030P-WP-F 30 W, 24 V polycrystalline solar panel [11] was used as the testing surface (Fig. 19). Although this panel does not match the engineering standards of space-grade solar arrays, it provides a practical and durable substrate for evaluating mechanical dust removal systems. This panel was generously provided by the team's project advisor, Dr. Hall, to support development and performance testing of the cleaning mechanism. (*See Appendix D for the Solartech SPM030P-WP-F data sheet.*)

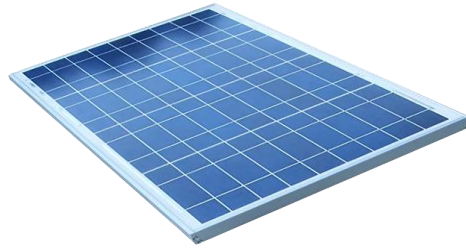


Figure 19. Solartech SPM030P-WP-F Polycrystalline Panel

The cleaning tests involved applying MGS-1 Martian regolith simulant to the surface of the panel and activating the robotic arm's brush mechanism to sweep across the array. Multiple cycles were run to evaluate dust clearance, contact behavior, and bristle alignment. These trials provided essential feedback for adjusting servo limits, brush torque, and arm positioning accuracy.

A comparison between the demonstration panel and the types used in Mars missions such as Spirit and Opportunity is shown in Table 8. While the flight-rated panels feature higher-efficiency triple-junction materials and are engineered to withstand extreme environmental conditions, the core issue of dust interference remains the same.

Table 8. Solar Panel Comparison

Feature (Specs)	Solartech Power W-Series SPM030P-WP-F	Mars Rover Solar Panels (e.g., Spirit & Opportunity)
Cell Type	Polycrystalline silicon	Triple-junction GaInP/GaAs/Ge
Power Output	30W	Approximately 140W (initial)
Voltage	24V	Variable
Efficiency	Approximately 14-15%	Higher Efficiency due to advanced materials
Cleaning System	None (Manually cleaned during testing)	None; relied on natural wind
Durability	Designed for terrestrial conditions	Engineering for Martian Environment

2.4 Sensors & Vision

Astraeus is equipped with a robust combination of proximity sensors and vision systems to support its autonomous navigation, obstacle avoidance, and target identification. These components work in tandem with each other to provide real-time environmental awareness, enabling the rover to navigate terrain safely and perform precise cleaning operations on solar panels.

2.4.1 Proximity Sensors

Obstacle detection is primarily handled by Sharp GP2D120 infrared (IR) proximity sensors [12] (Fig. 1\20), which provide reliable analog voltage outputs proportional to the distance of nearby objects. These sensors are mounted around the rover's chassis to detect obstacles in its path, allowing Astraeus to dynamically adjust its speed and direction to avoid collisions during traversal. (*See Appendix E for GP2D120 Datasheet*)

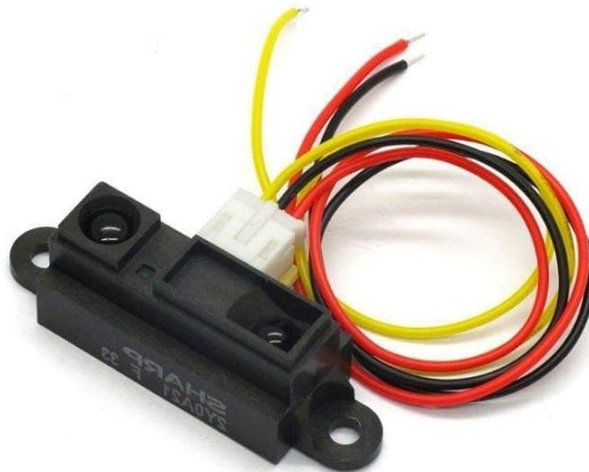


Figure 20. Sharp GP2D120 Infrared (IR) Proximity Sensors

Since the Raspberry Pi lacks native analog input capability, the IR sensors are interfaced through an external ADS1115 16-bit analog-to-digital converter (ADC) (Fig. 21) [13]. This high-resolution ADC accurately converts the analog voltage readings from the sensors into digital data that the Raspberry Pi can interpret [9]. This allows for precise distance calculations and smoother navigation behavior.

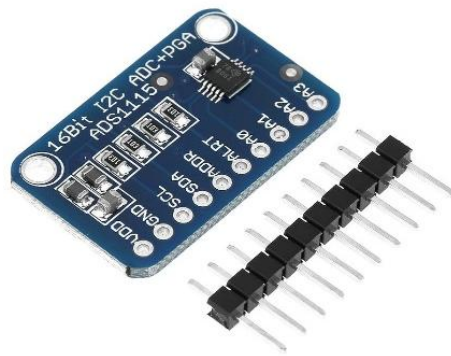


Figure 21. ADS1115 16-bit analog-to-digital converter

Because the IR sensors operate at 5V logic, and the Raspberry Pi's GPIO pins are only rated for 3.3V, a logic level shifter is used to safely interface the two components. This ensures that communication between the ADS1115 and the Raspberry Pi remains safe and reliable, without risking damage to the Pi's input circuitry.

2.4.2 AI-Based Vision with HuskyLens

Along with proximity sensing, Astraeus utilizes a HuskyLens AI camera (Fig. 22). This is a self-contained vision module capable of performing real-time object detection and tracking [14]. The camera uses built-in AI models to recognize visual cues enabling it to detect and follow towards the solar panel located at the solar panel as a target.

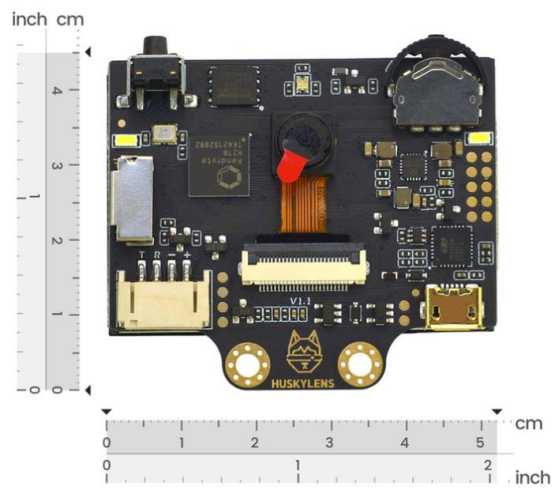


Figure 22. HuskyLens AI Camera

The Husky Lens communicates with the Raspberry Pi over UART, sending simplified tracking data without requiring intensive image processing on the Pi itself. This greatly reduces computational overhead while still enabling intelligent decision-making. During autonomous operation, the data from the HuskyLens is used alongside the IR sensor input to continuously refine the rover's navigation path and panel alignment.

2.5 Control Platform and Software

The control module of Astraeus is built around the Raspberry Pi 3 Model B, a Linux-based single-board computer that serves as the main platform for the entire system's hardware control. The Raspberry Pi handles all major subsystems including motor drivers, maestro servo controller, and sensors, using standard digital communication protocols such as UART, I2C, SPI, and general-purpose input and output [15].

(See Appendix F for Raspberry Pi 3 Model B Datasheet)

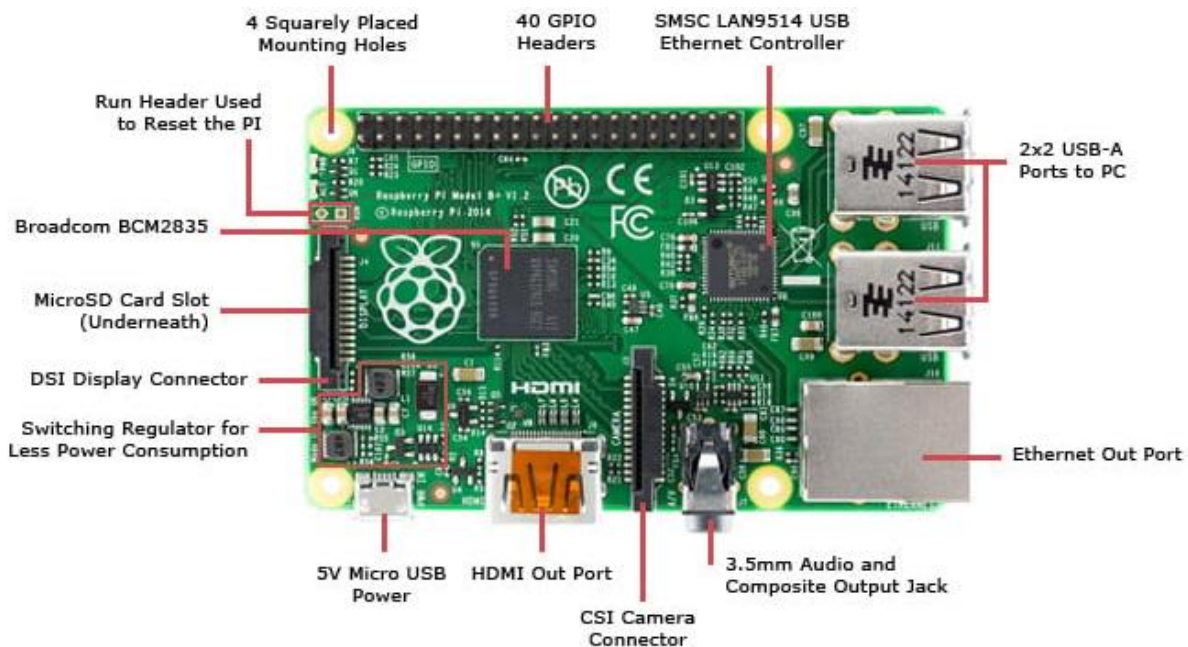


Figure 23. Raspberry Pi 3 Model B

Astraeus is programmed using Python, a versatile and widely supported language well-suited for hardware control and rapid development. Python allows for clear, modular code

structure and offers extensive libraries for communication and device integration. The project makes use of libraries for GPIO control, USB communication with the Mini Maestro servo controller, and I2C communication with the analog-to-digital converter that interfaces with the infrared distance sensors.

2.5.1 Programming Languages and Software Architecture

The software implementation for the Astraeus rover system utilizes a combination of programming languages, each selected for its specific strengths in handling embedded control, user interface design, and data processing. The control logic, sensor integration, and communication systems are primarily written in Python, while the front-end dashboard leverages standard web technologies to enable real-time interaction with the system.

Python

Python was the principal language used for developing the rover's backend control and logic systems. The Raspberry Pi 3 executed all control scripts written in Python, managing motor commands, sensor readings, and system behavior through various modules. Key functionalities included:

- Motor control and PWM regulation (motors.py)
- Autonomous navigation and tag alignment (autonomy.py)
- Sharp infrared distance sensing via ADS1115 ADC (sharp_sensor.py)
- Visual target tracking using the HuskyLens I2C module (visual_module.py)
- Subroutine triggering for Pololu Maestro controllers (maestro_module.py)
- Parallel execution of the control loop and Flask server (start_all.py)
- Logging of events, warnings, and alerts to a persistent database (logger.py)

Python's ease of integration, hardware compatibility, and multi-threading capabilities made it an effective tool for rapid development and reliable hardware control.

Web Technologies (HTML, CSS, JavaScript)

To provide a clean and interactive user interface, standard web technologies were used to build a dashboard accessible via any browser on the same local network. The interface included multiple pages for different control modes and system status viewing:

- **HTML** structured the web pages, including index.html, manual.html, and mode.html
- **CSS** was used for styling the user interface to ensure usability and responsiveness
- **JavaScript**, along with the nipplejs joystick library, enabled real-time manual control of the rover and dynamic updating of system logs via AJAX

This frontend was served through Flask, allowing for real-time two-way communication between the user and the rover.

SQLite

A lightweight SQLite database was integrated into the system to store all log entries. The logger.py module handled automatic insertion of timestamped messages categorized by severity (INFO, WARNING, ALERT). These entries were dynamically retrieved and displayed within the dashboard interface, enabling operators to monitor rover activity and status over time.

2.5.2 Software Libraries and Dependencies

To support the functionality of the Astraeus system, a wide range of software libraries and external packages were used. These libraries enabled hardware interfacing, I2C communication, sensor data handling, multi-process execution, and server hosting. Below is a table of the key libraries and their roles within the project:

Table 9. List of Libraries Used

Library	Purpose
Flask	Hosts the dashboard web server and handles HTTP requests for control and log viewing
gpiozero	Simplifies GPIO pin control for motor driver PWM and direction switching
RPi.GPIO	Used in low-level GPIO testing and setup, such as in blinka_test.py
smbus2	Enables I2C communication with peripherals like the ADS1115 ADC

adafruit-circuitpython-ads1x15	Interfaces with the ADS1115 module to read analog sensor data
multiprocessing	Allows concurrent execution of the control loop and Flask server
sqlite3	Manages event logging through a local database system
nipplejs (JavaScript)	Provides the virtual joystick interface for manual rover control
blinker	Supports signal handling between Flask components (optional utility)

2.6 Navigations & Task Execution Algorithms

Astraeus performs its autonomous tasks using a modular control architecture composed of independent yet interconnected software routines. These routines manage the rover's navigation, alignment, cleaning, and return sequences using sensor input and pre-programmed logic. To visualize and guide this system-level behavior, a series of flowcharts were developed during the proposal phase to represent the algorithms that define the rover's operational logic at each stage. These flowcharts are not only development tools but also serve as documentation for debugging, testing, and future system upgrades.

2.6.1 Main System Flow

The main flowchart (Fig. 24) serves as the overarching control structure for Astraeus, outlining the full sequence of operations from system initialization to shut down. Upon startup, the system enters a calibration stage to ensure all hardware components are set to a known and stable state. Following calibration, the vision system engages in detecting solar panels, preparing the rover for either autonomous or manual operation. If autonomous control is selected, the system transitions to a dedicated autonomy loop, while manual override allows human operators to directly command the rover. Upon task completion or intervention, the shutdown routine is executed to safely power down the system and reset components for future operations. This high-

level flow defines how control logic is distributed across subsystems and ensures safe, modular transitions throughout the mission.

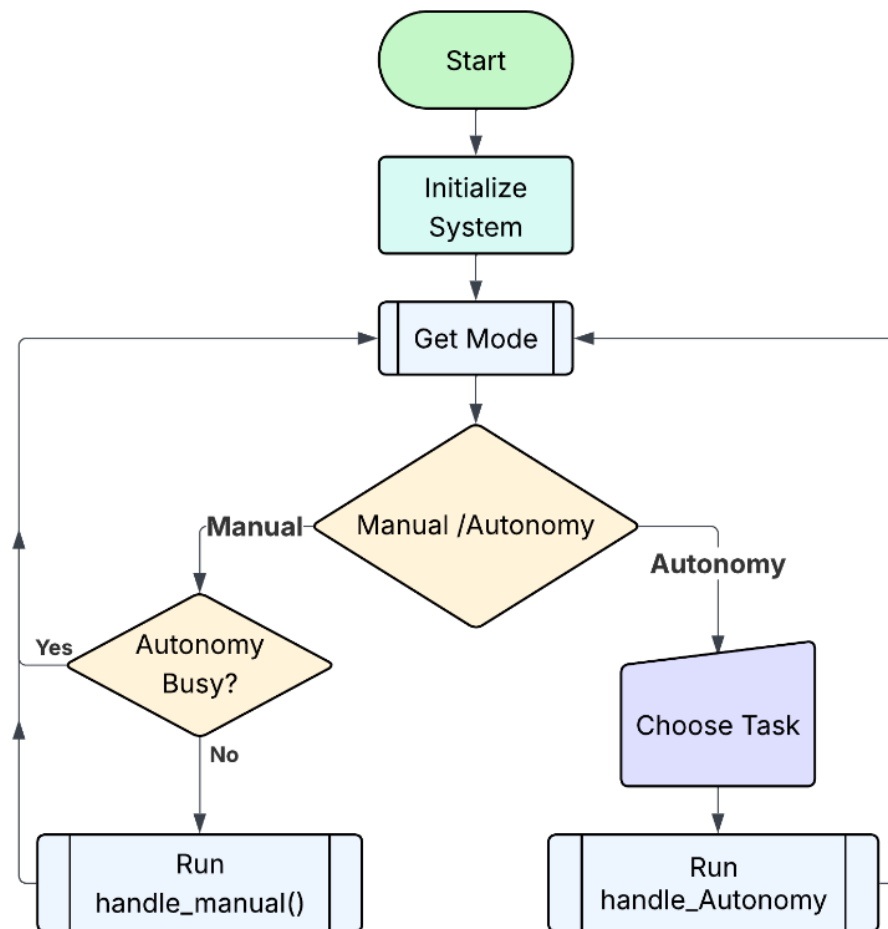


Figure 24. Main Flowchart

2.6.2 Manual Navigation Logic

The manual control flowchart outlines how Astraeus interprets direct user input to execute predefined commands. Upon receiving a command and associated speed value, the system evaluates the type of instruction. If the command corresponds to an alignment procedure (e.g., "align1" or "align2"), it initiates the appropriate sequence and checks for successful completion. If a failure is detected, the system logs an error and halts further execution.

If the command is a cleaning routine ("Seq1" or "Seq2"), the relevant cleaning sequence is triggered. For all other inputs, the rover sends a basic drive command controlling direction and speed. Once the instruction is executed, the command buffer is cleared, and the system is ready

for the next manual input. This routine allows a human operator to assume direct control over key rover functions for troubleshooting, calibration, or manual override operations. On the next page is the corresponding flow chart for the manual navigation logic (Fig. 25).

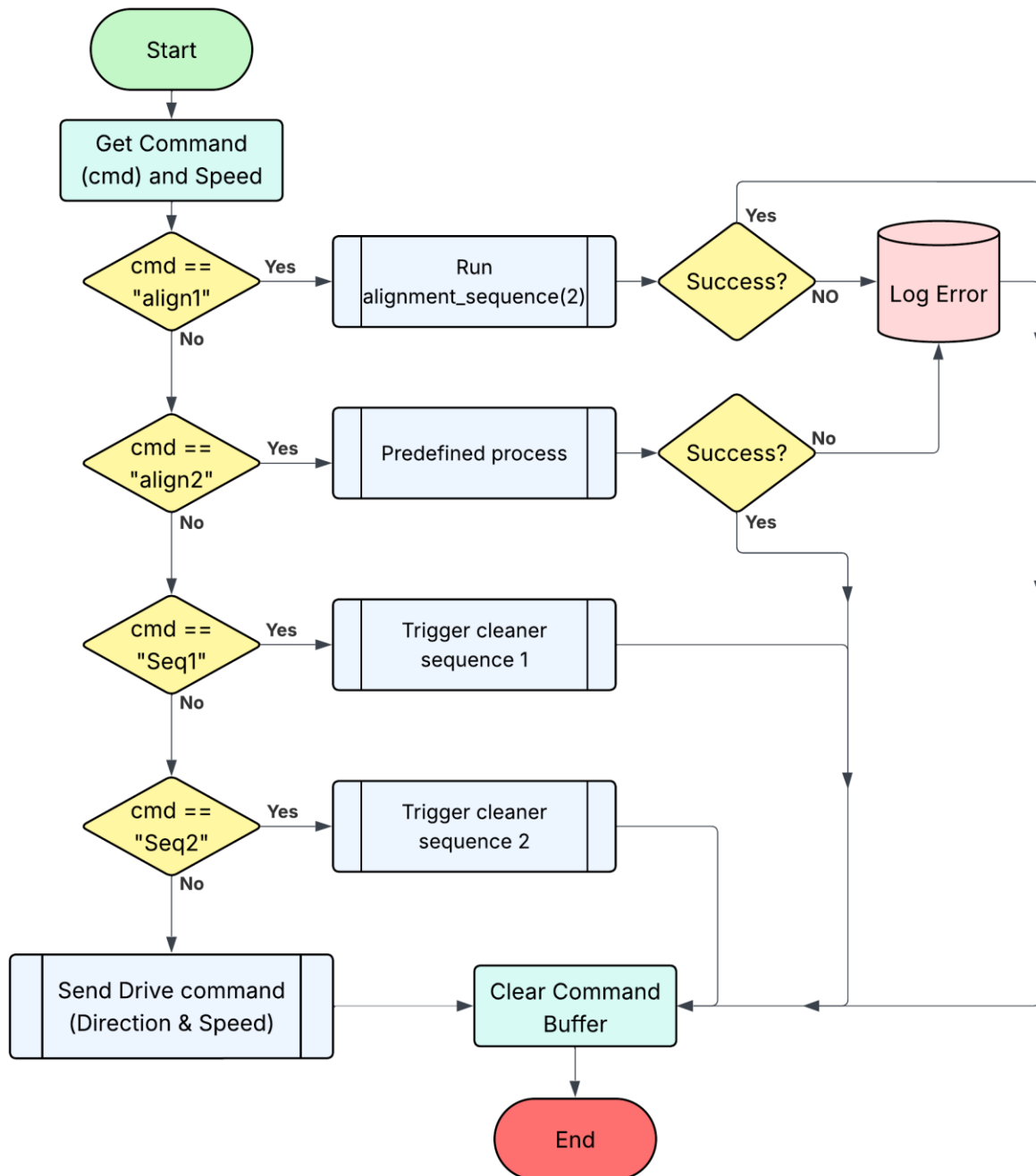


Figure 25. Navigation Process Flowchart

2.6.3 Autonomous Navigation Logic

The autonomy flowchart governs Astraeus' fully independent operation, enabling it to perform tasks without human intervention. The process initiates autonomous path planning, where the rover calculates an optimal route to the nearest solar panel using environmental inputs. As the rover navigates, it continuously scans for obstacles using onboard sensors and vision algorithms, dynamically adjusting its route as needed.

Upon arriving at the target, the system verifies the panel's presence and transitions to the cleaning phase. Once the cleaning task is complete, the rover initiates a return-to-base procedure, ensuring it navigates back to its starting point. This flow encapsulates Astraeus' decision-making capabilities, enabling it to handle variable terrain and perform tasks with high independence. The corresponding flowchart for autonomous navigation logic (Fig. 26) is on the next page.

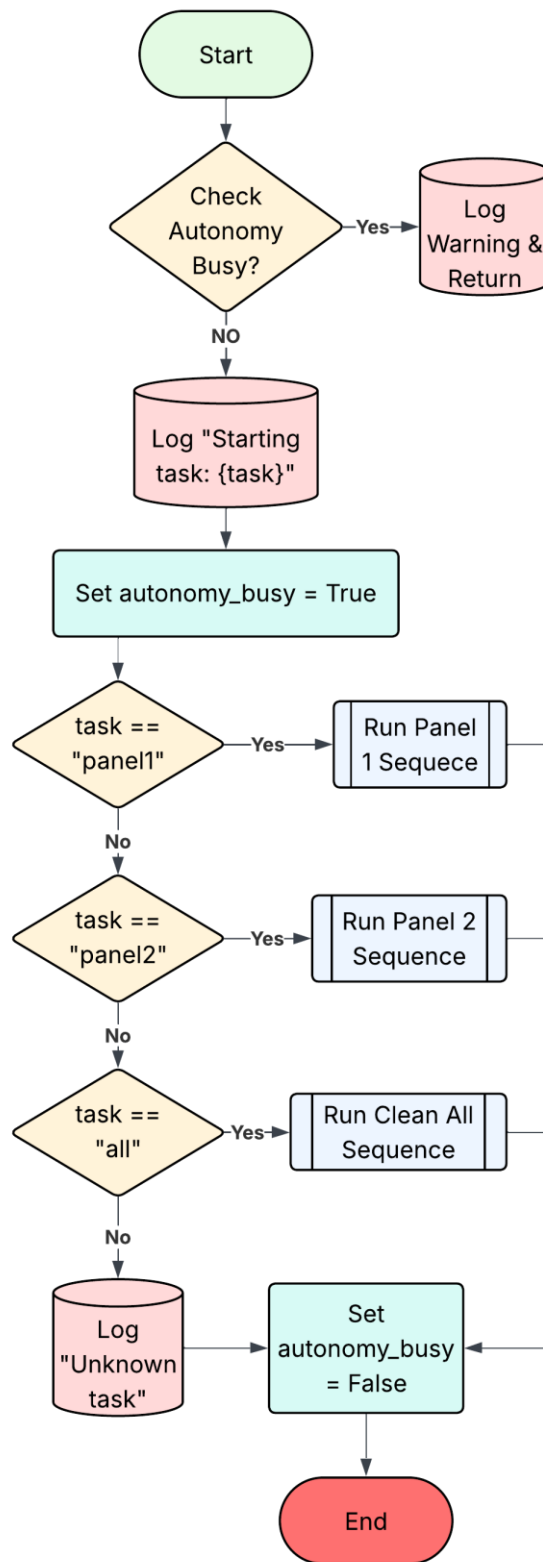


Figure 26. Alignment Process Flowchart

2.6.4 Panel 1 Cleaning Subroutine

The “Panel 1” flowchart (Fig. 27) defines the behavior of Astraeus when it is engaged in cleaning the first solar panel in a multi-panel configuration. The routine begins by initializing the cleaning arm and orienting the brush to align with the first edge of the panel. Once properly aligned, the arm performs a single cleaning pass across a specific section of the panel. After the sweep is completed, the system verifies the cleaning status of the segment and prepares to transition either to the next section or the next panel. This modular approach ensures that individual panels are addressed with precision and consistency, enabling repeatable and scalable cleaning operations. The following flow chart for the cleaning process of panel 1 is shown on the next page.

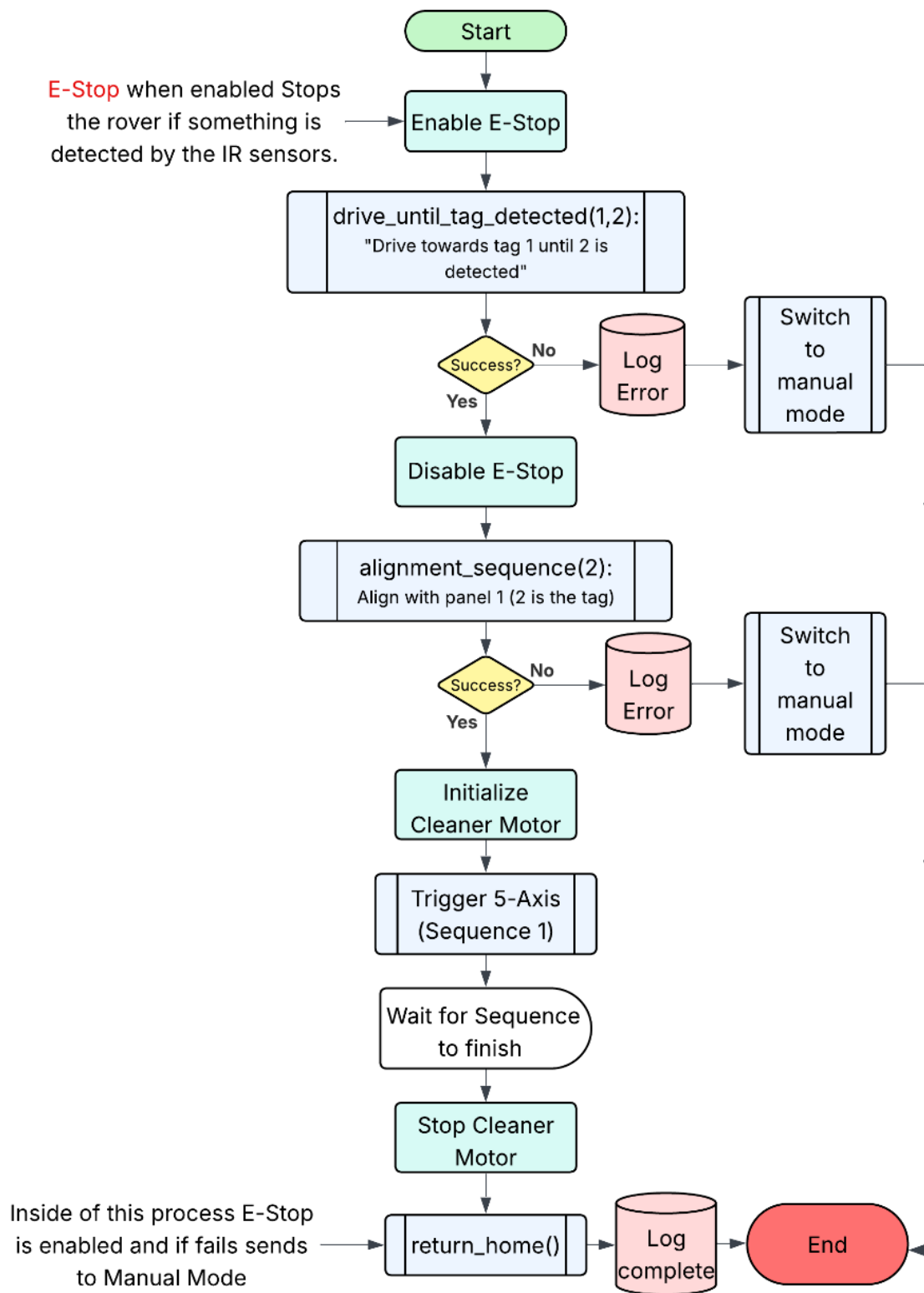


Figure 27. Panel 1 Cleaning Flowchart

2.6.5 Panel 2 Cleaning Subroutine

Like the logic in the Panel 1 sequence, the “Panel 2” flowchart (Fig. 28) manages the cleaning process for a subsequent solar panel. The rover repositions itself and adjusts the arm alignment to interface with the second panel. Once the panel is in position, the same cleaning sequence used in the first panel is executed, ensuring consistent methodology across different panel targets. After completing the task, the system logs the cleaning completion status, allowing the rover to track progress and make decisions for further tasks. This flow supports Astraeus’ ability to autonomously handle multiple panels during a single mission cycle.

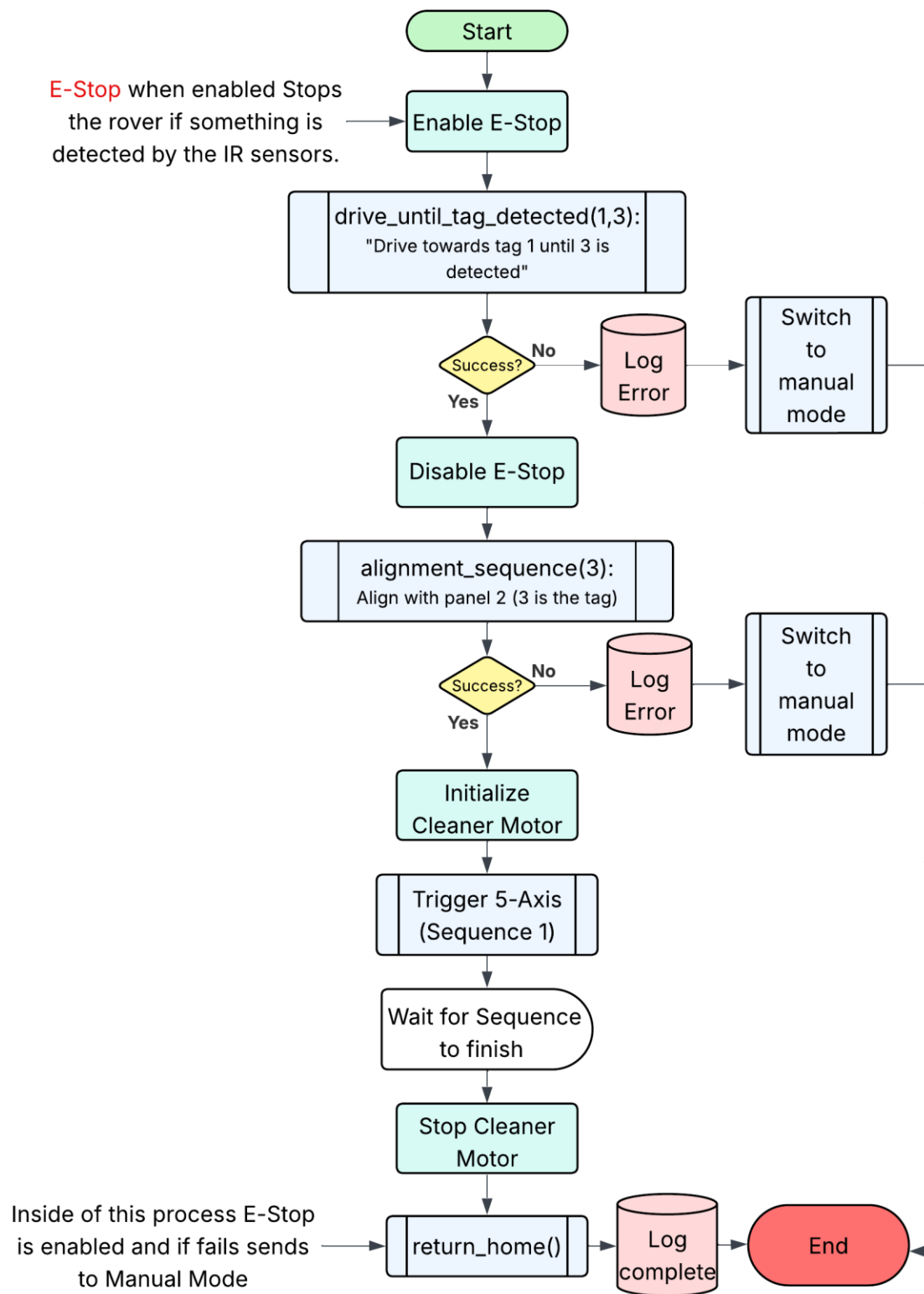


Figure 28. Panel 2 Cleaning Flowchart

2.6.6 Full Cleaning Routine

The “Clean All” flowchart (Fig. 29) provides the operational logic for executing a complete cleaning cycle of a detected solar panel. The sequence begins with initialization of the robotic arm and brush to a predefined starting position. Once the panel is confirmed and aligned, the panel is divided into discrete cleaning segments. The system then performs a looped sequence, where each segment is cleaned methodically. After each pass, the system checks if all segments have been addressed, and once cleaning is complete, the arm is retracted and stowed in its home position. This structured process ensures that each panel is thoroughly cleaned using a consistent and reliable pattern, optimizing coverage while minimizing mechanical strain.

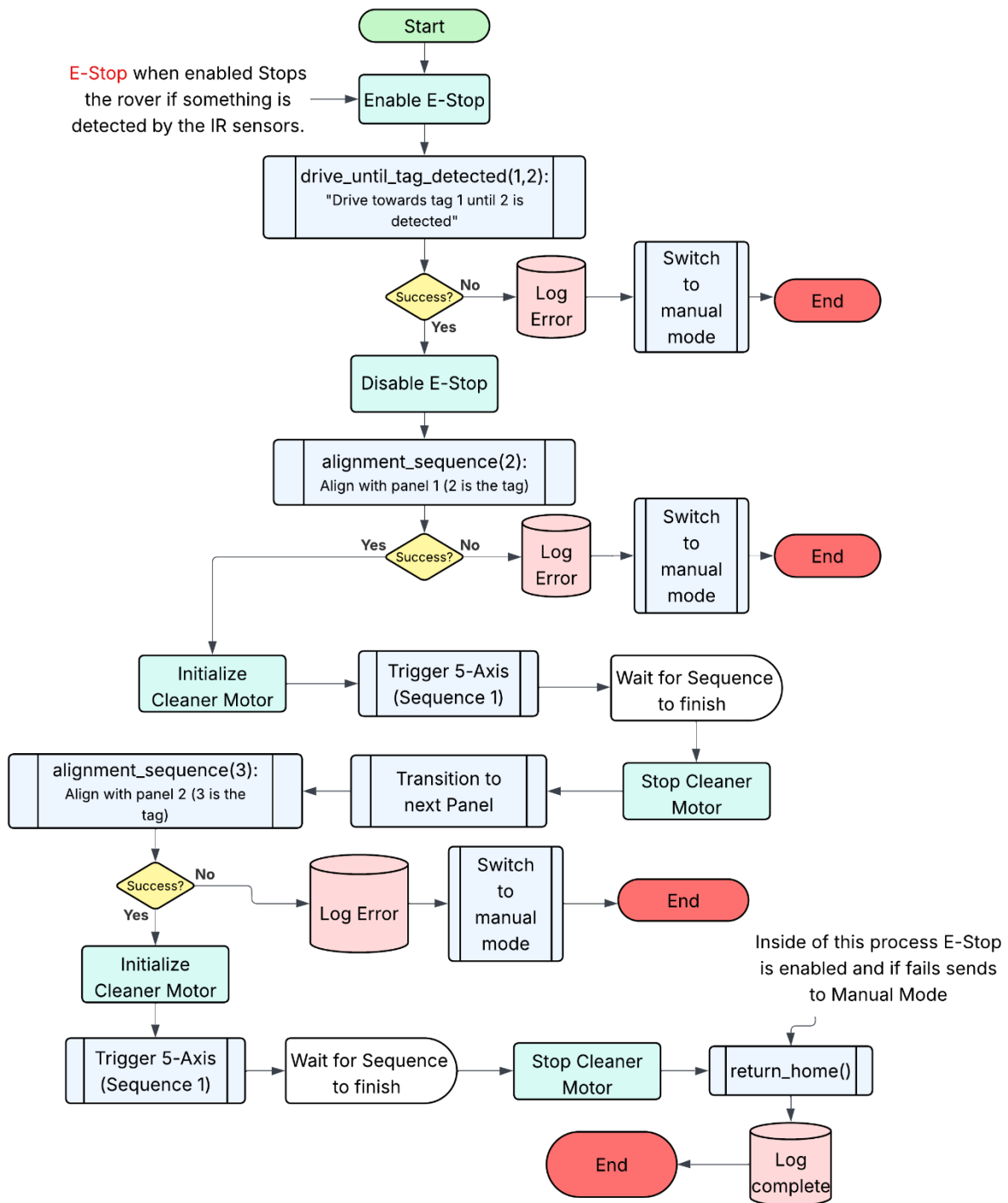


Figure 29. Clean All Flowchart

2.7 Power Budget

A comprehensive power budget was developed to verify that the selected power source could sustain all control, actuation, and sensor subsystems onboard Astraeus. The rover is powered by a 12V 30Ah LiFePO₄ deep cycle battery, which provides 360 watt-hours (Wh) of energy storage capacity (Fig. 30). This battery was chosen for its high energy density, flat discharge curve, rechargeability, and safety advantages over traditional lithium-ion chemistry.



Figure 30. Botku 12 30Ah Lithium LiFePO₄ Deep Cycle Battery

Before the system was fully integrated, a proposed power budget was drafted to guide hardware selection, energy management, and subsystem distribution. This budget was based on datasheet values and expected duty cycles for each module. Although adjustments were made during implementation, the proposed budget served as a baseline for confirming that the selected power source could sustain all rover operations. The proposal power budget is shown below in Table 10 and the Finalized budget is shown in Table 11.

Table 10. Proposed Power Budget

Drive System Power					
Component	Voltage (V)	Current (A)	Power (W)	Qty	Total Power (W)
Greartisan 12V 100RPM Motors	12	1.1	13.2	6	79.2
VNH5019 Motor Driver Carrier	5	1	5	2	10
Total (W)					89.2
Control System Power					
Component	Voltage (V)	Current (A)	Power (W)	Qty	Total Power (W)
Raspberry Pi 3B	5	3	15	1	15
ESP32-CAM	5	0.5	2.5	1	2.5
Maestro Servo Controller	6	1	6	1	6
VNH5019 Motor Driver Carrier	5	1	5	1	5
HuskyLens AI	5	0.32	1.6	1	1.6
ADS1115 16 Bit ADC PGA Converter	5	0.03	5.03	1	5.03
Sharp IR Analog Distance Sensor	5	0.03	5.03	4	20.12
Total (W)					55.25
Robotic Arm Power					
Component	Voltage (V)	Current (A)	Power (W)	Qty	Total Power (W)
25kg Servos (Base & Shoulder)	6	3.5	21	2	42
20kg Servo (Elbow)	6	2.5	15	1	15
55g Servos (Wrist & Tilt)	6	2.5	15	2	30
Pololu 6V 100RPM Brush Motors	6	2	12	2	24
Total (W)					111
System Total (W)					255.45
Power Module					
Component	Voltage (V)	Current (A)	Power (W)	Qty	Total Power (W)
LiFePO4 Deep Cycle Battery	12	30Ah	360 Wh	1	360
Total (W)					360

Table 11. Comprehensive Power Budget

Control Module Power					
Component	Voltage (V)	Current (A)	Power (W)	Qty	Total Power (W)
Raspberry Pi 3 Model B	5	3	15	1	15
VNH5019 Motor Driver Carrier	12	2.4	28.8	2	57.6
VNH5019 Motor Driver (Logic Supply)	3.3	0.04	0.132	2	0.264
VNH5019 Motor Driver Carrier	7	0.04	0.28	1	0.28
VNH5019 Motor Driver (Logic Supply)	3.3	0.04	0.132	1	0.132
Maestro Servo Controller	6	1	6	1	6
ADS1115 16 Bit ADC PGA Converter	5	0.03	5.03	1	5.03
Total (W)					77.63
Electro-Mechanical Module Power					
Component	Voltage (V)	Current (A)	Power (W)	Qty	Total Power (W)
Greartisan 12V 100RPM Motors	12	0.8	9.6	6	57.6
25kg Servos	7	2.5	17.5	2	35
45kg Servos	7	4.5	31.5	1	31.5
55g Servos (Wrist & Tilt)	7	0.4	2.8	1	2.8
N20 Micro Gear Motors	7	0.35	2.45	2	4.9
Total (W)					131.8
Vision Module Power					
Component	Voltage (V)	Current (A)	Power (W)	Qty	Total Power (W)
HuskyLens AI	5	0.32	1.6	1	1.6
Sharp IR Analog Distance Sensor	5	0.03	5.03	4	20.12
Total (W)					21.72
System Total (W)					231.15
Power Module					
Component	Voltage (V)	Current (A)	Power (W)	Qty	Total Power (W)
LiFePO4 Deep Cycle Battery	12	30Ah	360 Wh	1	360
Total (W)					360

Chapter 3

Contributions

Summary

This chapter details the contributions made by each team member to the design and development of Astraeus; a prototype solar support rover designed for future testing in simulated Martian environments. Specifically, it outlines the technical progress made in building the rover's major mechanical components and documents the decisions, modifications, and challenges faced throughout the design process.

- 3.1 Main Chassis Contributions**
- 3.2 Cleaning Arm Contributions**
- 3.3 Rocker-Bogie Contributions**
- 3.4 Software Development**

3.1 Main Chassis Contributions

This section outlines Mark's contributions to the design, fabrication, and refinement of the Astraeus main chassis. From early CAD modeling to physical assembly, the chassis underwent multiple design iterations to support subsystem integration, mechanical stability, and long-term modularity.

3.1.1 Structural framework and layout

The primary objective in designing the main chassis was to create a robust yet modular and adaptable frame capable of supporting all subsystems of Astraeus. The structural skeleton was built using 20 by 20-millimeter aluminum V-slot extrusions, chosen for their high strength-to-weight ratio and compatibility with standardized M4 T-nuts and bolts. These extrusions provided a versatile mounting platform for attaching mechanical subsystems, suspension components, and electronics.

The entire chassis was modeled using Autodesk Fusion 360. Creating a fully integrated CAD model enabled the team to visualize the complete system layout, plan subsystem integration, and identify potential mechanical or spatial conflicts prior to physical assembly. The model served as a reference throughout the build process, ensuring proper alignment and accurate placement of critical components such as the rocker-bogie suspension, battery tray, and arm mount.

To join the extrusions at structural intersections, custom corner brackets were designed and 3D printed using PETG filament, as shown in Figure 20. PETG was selected for its excellent mechanical strength, print reliability, and resistance to fatigue. The bracket geometry was optimized to reduce print time and improve surface quality while maintaining the durability needed to withstand repeated assembly, vibration, and operational stress. Their modular design allowed for quick reprinting and field replacement when design adjustments were required.

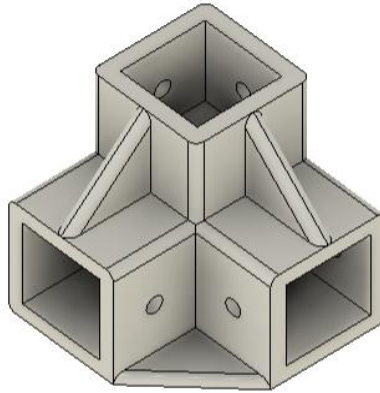


Figure 31. Corner Brackets on Print Plate

Once the bracket design was finalized, the complete frame was constructed in CAD (Figure 21). The extrusion layout was carefully planned to maintain structural symmetry, maximize component accessibility, and accommodate future expansion. The use of standardized hardware and open-slot rails ensured that all subsystems could be securely mounted while retaining the flexibility to reposition or replace them during later development phases.

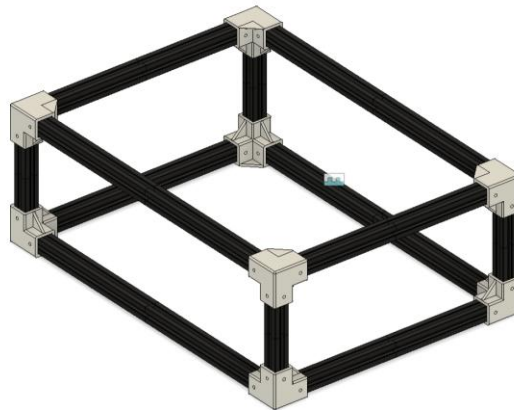


Figure 32. Main Chassis CAD Model

3.1.2 Structural Design Features

Three key structural features were integrated into the chassis design to support the mechanical integrity, modularity, and balance of the Astraeus rover during operation and testing. The rocker-bogie suspension hubs that mount on both sides of the chassis serve as the primary pivot points for the rocker arms. The rocker-bogie suspension hubs were mounted along both sides of the

chassis and served as the main pivot points for the rocker arms. These hubs were attached to the side rails using M4 T-nuts and bolts, allowing them to slide laterally during assembly for precise alignment. To reinforce the structure and maintain symmetry, a crossbar made from a cut section of 20 by 20-millimeter aluminum extrusion was installed to link the left and right hubs. This mechanical linkage ensured synchronized movement across the suspension and prevented uneven pivoting during traversal.

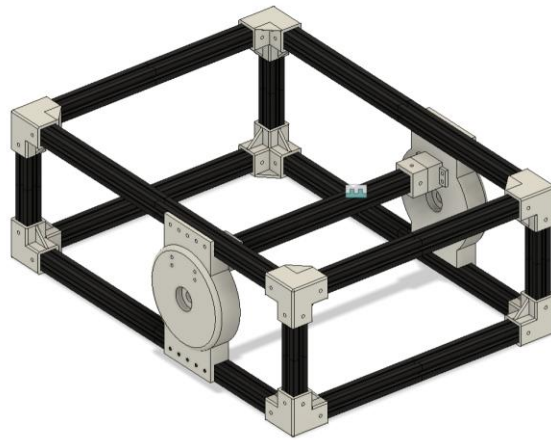


Figure 33. Rocker-Bogie Mounting Hubs

After finalizing the position of the suspension hubs, the crossbar extrusion was also utilized as a mounting structure for the rotational base of the robotic arm. This dual-purpose design allowed the servo responsible for base rotation to be anchored directly to the crossbar, providing a rigid and centrally located foundation. The use of this existing structural element reduced the need for additional material and helped integrate the robotic arm cleanly into the rover's overall layout.

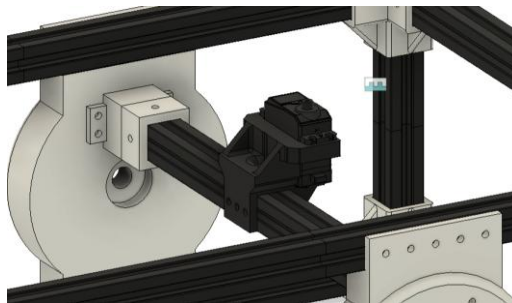


Figure 34. Base Rotation Servo Mount

To support rotation and reduce load on the base servo, a mounting plate was added to the top of the chassis. This plate spanned multiple extrusion rails and included a central cutout for the arm's main servo shaft. A four-inch lazy Susan bearing was incorporated beneath the plate to reduce rotational friction, distribute load evenly, and improve stability during arm motion. The widened geometry also allowed more mounting points to be used, which reduced flex and improved the rigidity of the overall assembly during brushing operations.

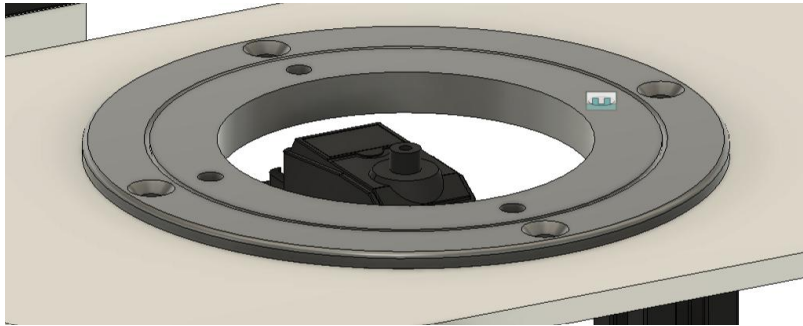


Figure 35. Robotic Arm Baseplate & Lazy Susan Integration

To balance the overall weight of the rover, the battery tray was installed along the internal rails of the frame. Mounted using M4 T-nuts and bolts, the tray could be shifted between the front and rear of the chassis to compensate for changes in the center of gravity as additional subsystems were added. This modular design made it possible to fine-tune the balance of Astraeus during testing without requiring permanent structural changes.

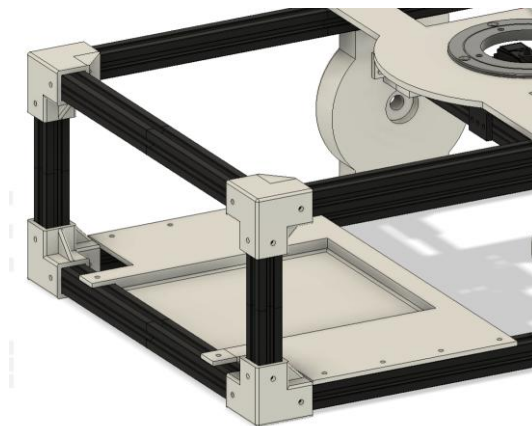


Figure 36. Battery Tray Integration

3.1.3 Assembly and Dimensional Refinements

During the initial assembly phase, several adjustments were made to improve structural stability, ensure mechanical alignment, and accommodate shifting mass distribution as new subsystems were integrated. Aluminum extrusions were trimmed and re-squared as needed to eliminate minor inconsistencies and improve frame symmetry.

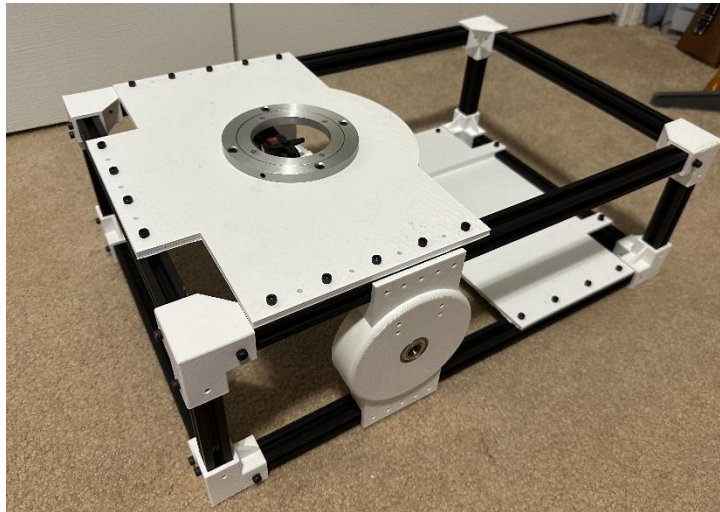


Figure 37. Main Chassis Assembled

Thinking ahead and compensating for the unknown mass distribution during early design stages, the battery tray was mounted using M4 T-nuts and bolts, allowing it to slide along the interior frame between the front and rear sections of the rover. This adjustment allowed for fine-tuning of the center of gravity and simplified balancing the chassis as heavier subsystems such as the arm and electronics were installed. Another enhancement made prior to the final assembly was widening the mounting plate for the base of the robotic arm. By extending the plate across more of the aluminum extrusion surface, additional mounting points could be secured, reducing torsional stress during arm actuation. This change improved mechanical stability during brushing sequences and minimized vibration.

3.2 Cleaning Arm Contributions

This section outlines Marks' contributions to the development process of the Astraeus robotic cleaning arm, from initial CAD modeling and servo selection to physical assembly, structural reinforcement, and servo programming using the Maestro Servo Control Center.

3.2.1 CAD Design & Mechanical Layout

The Astraeus robotic arm was conceptualized and modeled as a five-axis articulated arm, capable of cleaning solar panels autonomously using a preprogrammed sequence after the rover has aligned with the panel. The CAD model, created using Autodesk Fusion 360, detailed each point of articulation, including base rotation, shoulder and elbow articulation, wrist rotation, and a tilting brush end-effector. The arm was designed to have a reach of approximately 27.5 inches which is sufficient to clean a full 30W Solartech Power W-Series solar panel from a fixed chassis position. Figure 27 below represents this initial design as it was presented during the proposal phase.

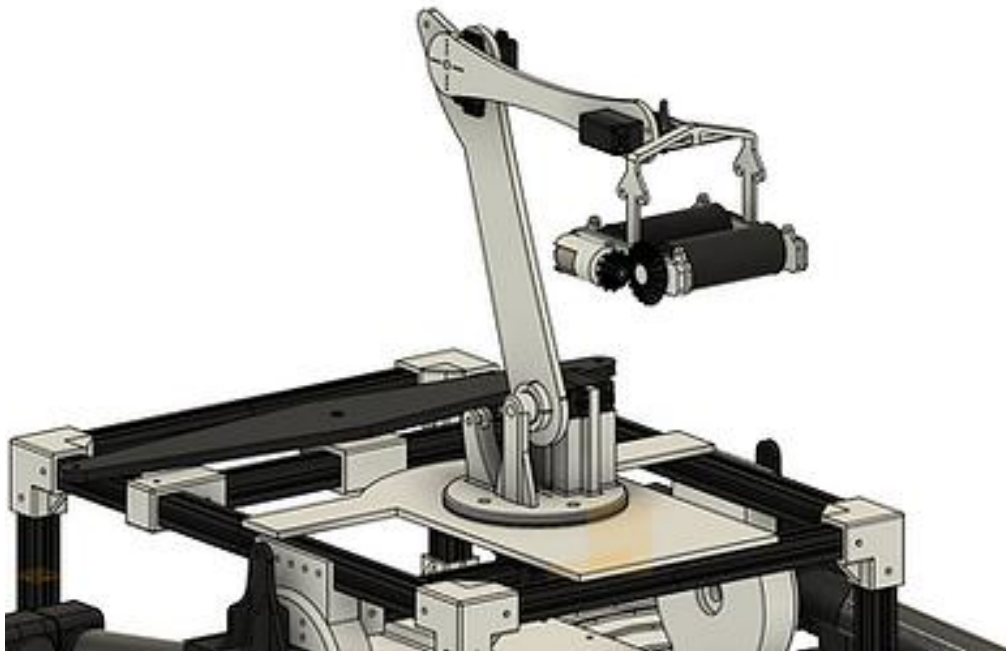


Figure 38. Initial Robotic Arm CAD Model (Proposed Design)

The model accounted for servo selection, wiring clearances, and structural stability. The base of the arm incorporated a high-torque servo mounted on a reinforced plate, supported by a 4-inch lazy Susan bearing to reduce axial load on the servo during arm movement. The model also included mounting brackets, joint alignments, and spatial integration with the main rover chassis. Because it was designed modularly, it made it easy for the design to undergo multiple revisions for optimizing servo loads and clearances prior to assembly.

3.2.1 Initial Servo Selection and Justification

Servo models were initially chosen based on their torque ratings and suitability for the load experienced at each joint. The first configuration used DS3225MG 25kg·cm servos for the base and shoulder joints, a DS3218MG 20kg·cm servo for the elbow, and MG996R servos for the dual-axis wrist mechanism. The specifications for this original proposed design during the previous semesters are provided in Table 12.

Table 12. Robotic Arm Servo Specifications

Servo Function	Model	Torque	Operating Voltage	Stall Current	Rotation Range
Base	DS3225MG	25kg·cm (6V)	4.8V - 7.4V	3.5A	180°
Shoulder	DS3225MG	25kg·cm (6V)	4.8V - 7.4V	3.5A	180°
Elbow	DS3218MG	20kg·cm (6V)	4.8V - 6.8V	2.5A	180°
Wrist (U/D)	MG996R	20kg·cm (6V)	4V - 6V	2.5A	180°
Wrist (L/R)	MG996R	55g Metal Gear Servo	4V - 6V	2.5A	180°

However, after physical testing and structural revisions, it became clear that higher-torque servos were required to ensure reliable operation. The finalized configuration included DS3235 servos at the base and shoulder, DS3225MGs at the elbow and wrist (U/D), and a smaller DS3225MG handling wrist tilt (L/R) due to its lightweight load. The revised servo specifications are summarized in Table 13.

Table 13. Revised Robotic Arm Servo Specifications

Servo Function	Model	Torque	Operating Voltage	Stall Current	Rotation Range
Base	DS3235	25kg	7V	~2.5 A	180°
Shoulder	DS3235	45kg	7V	~4.2 A	270°
Elbow	DS3225MG	45kg	7V	~4.2 A	270°
Wrist (U/D)	DS3225MG	25kg	7V	~2.5 A	180°
Wrist (L/R)	DS3225MG	55g	7V	~0.4–0.5 A	180°

3.2.3 Power Considerations

After finalizing the revised servo configuration for the robotic arm, it became critical to evaluate whether the increased electrical load from the new servos, particularly at the shoulder and elbow joints, would remain within the rover's available power budget. The higher torque ratings of these servos introduced greater current demand, which required a careful recalculation of the total power consumption to ensure compatibility with the existing electrical system.

Each servo was changed based on its mechanical performance requirements. After selection, the electrical characteristics were evaluated in relation to the rover's standardized power supply. The operating voltage for all servos was set to 7 volts, delivered by a dedicated high-current voltage regulator. According to manufacturer data, the estimated stall current values for each servo in the final configuration are as follows:

- Base (DS3235): ~2.5 A
- Shoulder (DS3235): ~4.2 A
- Elbow (DS3225MG): ~4.2 A
- Wrist (U/D) (DS3225MG): ~2.5 A
- Wrist (L/R) (DS3225MG): ~0.5 A

When all values are summed, the total estimated stall current is approximately 13.9 amperes. At the operating voltage of 7 volts, this results in a peak power demand of:

$$\text{Power} = \text{Voltage} * \text{Current} = 7V * 13.9A = 97.3 W$$

This value represents a worst-case scenario where all five servos are stalled simultaneously, which is highly unlikely during normal operation. In typical use, servos draw only a fraction of their stall current. Based on manufacturer guidelines and real-world conditions, the robotic arm is expected to operate between 30 to 50 percent of stall current. Using an average of 50 percent, the expected current draw becomes:

$$\text{Average Current} = 13.9A * 0.5 = 6.95 A$$

$$\text{Average Power} = \text{Voltage} * \text{Current} = 7V * 6.95A = 48.7 W$$

This average power consumption fits within the robotic arm's allocated share of the rover's overall power budget. It also provides a safety margin to accommodate variations in load during operation. The wiring, connectors, and voltage regulator selected for this system are rated above the expected peak current, ensuring both electrical reliability and thermal safety.

The updated power budget also confirms that the final servo configuration is not only mechanically capable but also electrically viable. With this validation complete, the arm was able to be mechanically assembled and integrated onto the arm for further testing.

3.2.4 Robotic Arm CAD Design

After confirming the revised servo configuration met both mechanical and electrical requirements, the project moved into the physical assembly stage. When developing the initial concept for the robotic arm, multiple CAD revisions were made throughout the design process to enhance mechanical stability and accommodate the functional requirements of the cleaning system. Early joint layouts and arm dimensions were based on preliminary torque calculations and assumptions, but once the parts were 3D printed and assembled, significant flex became apparent, especially around the bicep and elbow joints.

During initial physical testing, the arm exhibited noticeable wobbling and instability, particularly when fully extended or lifting the brush (Fig. 28). This behavior compromised the precision of movement and prompted a close evaluation of the mechanical design. It was determined that the source of the issue stemmed not only from insufficient torque in the original servos but also from the structural layout of the elbow joint.

The elbow was initially constructed using a single shear mounting configuration, where the pivot bolt passed through only one side of the joint bracket. This simplified assembly but concentrated stress on a single plane, which allowed for excessive rotational play. Under load, this created a mechanical weak point that reduced accuracy and raised concerns about long-term fatigue.

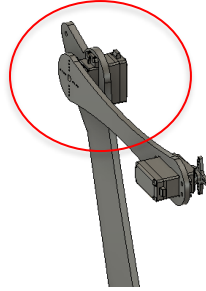


Figure 39. Robotic Arm single-shear joint

To address this issue, the elbow was redesigned using a double shear mounting configuration, where the pivot pin is sandwiched between two opposing plates (Fig. 29). This evenly distributes the applied load across both sides of the joint, dramatically improving structural stability and reducing flex. Additional CAD modifications included reinforcing the forearm with another double shear bracket and adding an outer plate between the shoulder and elbow segments to stiffen the arm as it moves.

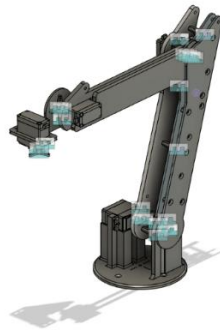


Figure 40. Revised Robotic Arm

These structural changes directly informed the decision to revise the servo specifications. The observed flex underload revealed that the originally selected servos did not provide sufficient holding torque. To resolve this, the shoulder and elbow servos were upgraded to DS3235 and DS3225MG models, which offered significantly higher torque output. With the stronger servos and revised mechanical joints, the arm's performance and stability improved substantially. Initial tests from programming the sequences confirmed that the reinforced design minimized structural wobble and maintained accuracy through the entire range of motion.

3.2.5 Brush Cleaning Mechanism CAD Design

After completing the arm revisions, focus was shifted to finalizing the design for the cleaning mechanism. The goal of this specific subsystem was to simulate removing Martian regolith from solar panels using a lightweight yet efficient dual rotating brush assembly.

All components for the cleaning mechanism were modeled in Fusion 360, including the structural housing and custom gear train. During the proposal phase, the option of sourcing custom-made brush assemblies from commercial vendors to meet the specific needs of this application were considered. However, due to the high cost of custom fabrication and low production volume, this option was ultimately deemed impractical within the project's budget. As a cost-effective alternative, nylon-Teflon hybrid brushes were selected for their ability to generate a static charge during rotation, which enhances their effectiveness in lifting fine particulate matter such as Martian dust simulant. These brushes, commonly found in robotic vacuum systems (Fig. 41), were adapted to fit the arm using 3D-printed holders designed in Fusion 360.



Figure 41. ECOMAID Brush Compatible for iRobot Roomba

To rotate each brush, N20 micro gear motors were chosen due to their compact size, low weight, and adequate torque output. Because the motor and brush shafts are oriented perpendicularly, a direct-drive configuration was not possible within the constrained form factor of the wrist assembly. To address this, a compact bevel gear system was designed in Fusion 360 to transfer motion from the motor shaft to the brush.

Each gear was designed to snap-fit onto the end of the brush shaft, eliminating the need for glue, fasteners, or brush modification. On the motor side, the mating bevel gear is friction-fit

to the N20 motor shaft, providing secure torque transfer without requiring permanent alterations. This allowed for fast assembly, simplified maintenance, and brush replacement if needed.

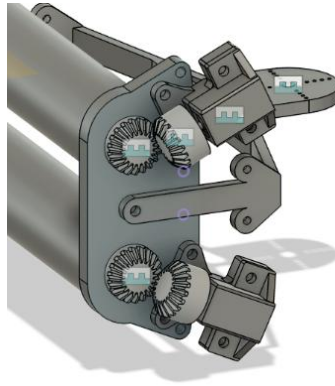


Figure 42. Bevel Gear Mechanism CAD Model

A servo-controlled tilt mechanism was also integrated at the wrist, enabling the entire brush assembly to pitch forward and backward to conform to varying panel angles during cleaning. This articulation ensured even contact pressure across the brush path and helped compensate for terrain or rover body tilt.

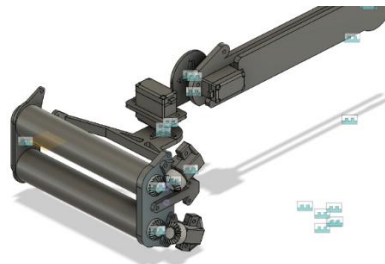


Figure 43. Wrist Rotation Mechanism CAD Model

The final brush module combined the rotating brush, friction- and snap-fit bevel gear system, N20 gear motor, and tilt servo, all mounted to a PETG-printed housing. Since both the robotic arm and brush mechanism were finalized and validated, they were ready to be fully integrated onto the rover chassis during final assembly and wiring.

3.2.6 Arm & Electrical Integration

Once the mechanical structure of the robotic arm and brush mechanism were finalized, the next stage involved integrating the assembly into the rover and completing the electrical wiring. The

arm was mounted onto a 3D printed plate on the rover's chassis using M4 screws and washers. A 4-inch lazy Susan bearing (Fig. 44) was installed beneath the base servo to reduce strain on the servo shaft while allowing smooth rotational movement.



Figure 44. 4-inch Lazy Susan

Initially, the servos were powered directly through the Maestro servo controller, but this configuration quickly proved inadequate. The wiring on the Maestro was not rated to handle the combined current load of all five high-torque servos, which caused power inconsistencies and unreliable operation. To resolve this, the servo power lines were rerouted to a dedicated power rail connected to a buck converter that steps down 12V to 7V and can handle up to 20 amps of continuous current (Fig. 45). This external power rail ensured stable and sufficient power delivery to each servo.



Figure 45. DC-DC Stepdown Regulator 6-40V into 1.2-36V 20A out

Each servo's control signal line was still connected to the Maestro controller, occupying channels 0 through 4 in the following order: base, shoulder, elbow, wrist (up/down), and wrist

(tilt). However, because the servos were no longer powered through the Maestro itself, shared ground was necessary to ensure signal integrity. A ground wire was added between the Maestro's power ground and the new external 7V power rail, allowing both systems to share a common reference. Once this ground connection was established, all servos began responding reliably to signal commands. The cleaning brush N20 micro gear motors were electrically wired in parallel to simplify routing and ensure synchronized rotation. These motors were powered independently of the servos and were connected to a dedicated VNH5019 motor driver (Fig 46).

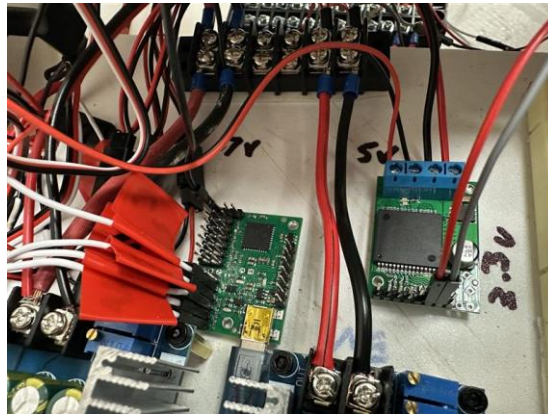


Figure 46. Maestro & VNH5019 Wiring

All wiring was organized using zip ties to secure loose lines and prevent interference with the arm's moving components. Careful attention was paid to routing near joints and along the chassis to avoid pinch points. While no protective sleeve was used, the clean routing and physical constraints ensured that cables remained secure during full articulation.

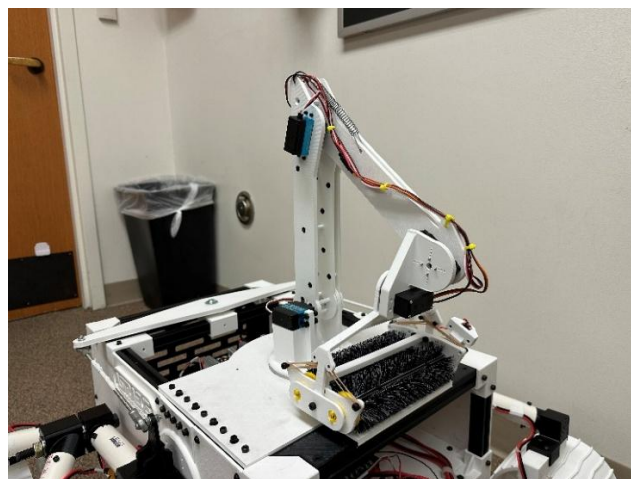


Figure 47. Arm & Brush Fully Assembled

With the wiring and mounting complete, the arm system was powered on and underwent control validation. Each servo responded correctly through the Maestro interface, and the N20 motors operated reliably under VNH5019 control. This marked the successful completion of mechanical and electrical integration for the Astraeus robotic arm system.

3.2.7 Servo Programing with Maestro Control Center

With servo power and signal wiring complete, servo control and motion sequencing were implemented using the Pololu Maestro Control Center. This Windows-based application communicates directly with the 12-channel Micro Maestro over USB and provides a graphical interface for configuration, manual control, and embedded scripting to enable autonomous operation of the robotic arm. Each joint of the robotic arm was assigned to a dedicated Maestro channel as follows:

Channel 0: Base rotation

Channel 1: Shoulder

Channel 2: Elbow

Channel 3: Wrist (up/down)

Channel 4: Wrist (left/right tilt)

The Status tab within the Maestro software was initially used to manually actuate each servo using on-screen sliders. This facilitated real-time validation of mechanical range, identification of center positions, and refinement of joint limits. In the Channel Settings tab, each servo was configured with a minimum pulse width of 3968, a maximum of 8000, and a neutral center of 6000 microseconds. Channel-specific speed and acceleration values were assigned based on the mechanical load of each joint. For instance, the shoulder joint, which supports the heaviest load, was assigned lower speed and acceleration values to reduce mechanical stress during motion.

To automate movement, the Sequence tab was used to construct motion routines from a series of servo position frames with defined durations. Each sequence comprises named frames that store exact servo positions and timing in milliseconds, allowing repeatable and smooth transitions. Key sequences developed for Astraeus include:

- Sequence 0: Resets the robotic arm to a stowed position for startup and shutdown procedures.
- Sequence 1: Performs a full brushing routine, including elbow extension, coordinated wrist adjustment, and sweeping motions.
- Sequence 1.1: Extends the basic brushing routine with additional directional shifts, allowing multi-pass coverage across a solar panel surface.

Each frame within a sequence is defined by target values for servo positions, speed, and acceleration. For example, the frame SweepDownL within Sequence 1 activates a combination of base rotation, elbow extension, and wrist articulation to simulate a downward brushing motion on the left side of the panel. Frames like Raise for Clearance provide lift motions to avoid collisions between brushing passes.

Once sequences were validated through live testing, they were exported to the Script tab using Pololu's stack-based scripting language. In the script, each frame was converted into a reusable subroutine (e.g., frame_0..11, frame_3_4), and high-level sequence logic was constructed using custom subroutines such as Sequence_0, Sequence_1, and Sequence_11. Script commands such as servo and delay control motion execution and timing. A key architectural advantage of this approach is that the complete motion logic is stored onboard the Maestro controller. The Raspberry Pi control system can initiate an entire sequence using a single serial command, significantly reducing software overhead and improving execution reliability during autonomous operation. For reference, the full annotated Maestro configuration file is included in Appendix [G], detailing all servo settings, sequence definitions, and embedded scripts used in the Astraeus robotic arm.

3.3 Rocker-Bogie Contributions

This section marks the beginning of Pedro's contributions to the Astraeus rover project.

3.3.1 Rocker-Bogie Design

The rocker-bogie suspension system was fully designed from scratch in Fusion 360, including all structural joints, brackets, and wheel components. Every part of this subsystem was created to strict dimensional tolerances to support reliable movement, modular assembly, and real-world fabrication using PVC tubing and 3D-printed brackets.

The design process began with the creation of a single wheel, which was fully modeled with a ribbed exterior for terrain grip and a custom internal hub to match the motor shaft. Once finalized, this wheel was duplicated and mirrored across the chassis to form the remaining five wheels. Similarly, the entire rocker and bogie geometry was first designed on one side of the rover, then mirrored to create the opposite side. This approach ensured symmetry while reducing repetitive CAD work and limiting dimensional mismatch.

Three primary rocker joints were designed:

- The **chassis-to-rocker joint**, which allows each rocker arm to pivot relative to the main frame.
- The **rocker-to-bogie joint**, enabling the bogie segment to rotate independently and allow rear wheel articulation.
- The **sway bar interface joint**, connecting the rocker arms to the central differential bar and allowing balanced terrain compensation between both sides.



Figure 48. Astraeus Wheel

These joints were modeled individually and designed to accept standard PVC tubing, which serves as the load-bearing link between segments. Each joint was developed to ensure tight press fits or proper mounting holes, depending on the connection type.

At the end of each rocker arm, a modular three-part assembly was created to support the motor and wheel. This included:

- The **motor bracket**, which secured the drive motor directly to the wheel mount with appropriate offsets and flange supports.
- The **L-bracket**, which linked the motor bracket to the end of the rocker arm while maintaining correct alignment.
- The **end joint**, which completed the connection between the rocker and bogie structure, formed the transition to the next wheel support.



Figure 49. Rocker-Bogie Assembly (Right)

These components were designed to interlock and remain interchangeable. This modularity was especially important during physical testing, as it allowed damaged or experimental parts to be swapped without redesigning the entire system.

The most technically challenging part of the design involved angled tube connections between the rocker and bogie arms. These parts needed to be connected at both outward and downward angles, which cannot be accomplished with basic extrusions. To resolve this, a multi-step modeling approach was used:

- A construction line was created between the start and end points of each angled connection.
- A sweep extrusion was performed along that line to form the connecting pipe.
- Each end was trimmed flat using a 90-degree cut, ensuring proper mounting faces.

- Because the angled cut created elliptical ends, new planes were placed at each face, and a 1-inch diameter circle was redrawn and extruded to restore perfect circular geometry.

This method was repeated for each connecting tube, ensuring all joints maintained structural integrity and geometric precision.

The **sway bar** was also modeled from scratch and connects both rocker arms through a differential linkage. It was designed to rotate passively, allowing the rover to adjust to uneven terrain while keeping the main chassis relatively level. The sway bar terminates in custom mounting heads that interface with the rocker joints on each side.

Every structural segment of the rocker-bogie system was matched to standard PVC tube sizes for easy fabrication. The CAD models ensured each piece aligned correctly with physical cut lengths and motor dimensions.

This section of the building was critical to both the rover's functionality and overall presentation. Any dimensional inaccuracy would compromise weight distribution, cause instability, or interfere with motor alignment. The success of this design directly impacted on the rover's ability to operate effectively and withstand real-world conditions.

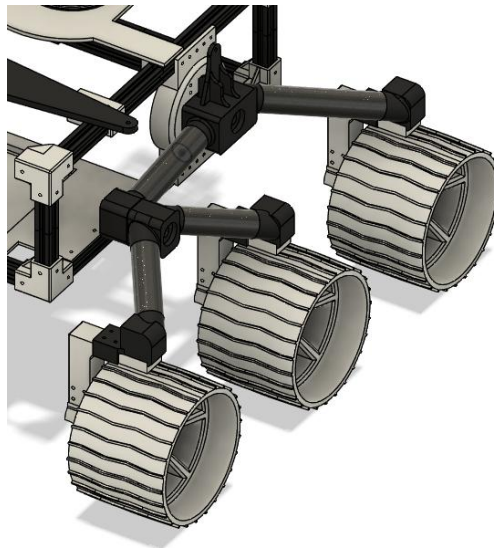


Figure 50. Full Rocker Bogie Assembly

3.4 Software Development

This section outlines Pedro's contributions to the software development of the Astraeus rover. The software system was engineered to enable both manual and autonomous operation, with integrated sensor feedback, precise motor control, and dynamic task execution. Key functionalities include real-time web-based interaction, modular control over subsystem behaviors, and coordination with external components such as vision modules and actuator controllers. Every part of the codebase, from low-level GPIO handling to high-level user interface logic, was developed in-house with a focus on robustness, clarity, and future expandability. The resulting architecture allows for seamless transitions between control modes, efficient sensor data processing, and simplified debugging across individual modules.

3.4.1 Software Architecture Overview

The software system developed for the Astraeus rover was designed with modularity and layered control in mind. Each component of the software is encapsulated in its own module, allowing for individual development, testing, and debugging without compromising the integrity of the overall system. The architecture is split into three primary layers: the control logic layer, the hardware interface layer, and the user interface layer.

At the core of the system lies the control logic, which governs the flow of commands and decision-making during both manual and autonomous operation. The entry point for this logic is handled by `start_all.py`, which is responsible for launching concurrent processes, specifically the main control loop (`main.py`) and the Flask-based web server (`app.py`). These processes communicate through shared buffer modules (such as `command_center.py`), ensuring real-time responsiveness across both user and sensor-driven events.

Manual and autonomous modes are managed through separate but coordinated pathways. Manual commands originate from the web interface and are transmitted via HTTP POST requests to `app.py`. These are then stored in memory and continuously polled by the control loop in `main.py`, which interprets and executes them accordingly using the movement functions defined in `motors.py`.

Autonomous functionality is implemented as a modular set of routines inside `autonomy.py`. These routines operate in stages, each representing a distinct autonomous behavior, such as tag following, alignment, cleaning, or repositioning. Transitions between

stages are handled through internal logic based on visual input, sensor feedback, or timed sequences. The control loop evaluates the current mode of operation and delegates control to either the autonomous behavior set or the manual command interpreter.

Hardware interfacing is abstracted into independent modules. Motor control is handled through `motors.py`, which provides clean, reusable functions for directional movement, speed control, and braking. Sensor input is managed by `sharp_sensor.py`, which converts analog distance data into usable values using the ADS1115 ADC over I2C. External systems such as the Pololu Maestro are triggered through `maestro_module.py`, enabling the activation of cleaning routines or other subassemblies without interfering with the rover's primary control flow.

The user interface layer is powered by a Flask server defined in `app.py`. This server renders several HTML pages (`home.html`, `manual.html`, `mode.html`, etc.) and provides endpoints for sending commands, adjusting speed, and retrieving system logs. Interaction is supported by JavaScript, with `joystick.js` translating user input from a virtual joystick into movement commands. These commands are sent to the backend and processed in near real time.

A local SQLite database serves as the event logging system. The `logger.py` module writes time-stamped logs categorized by level (INFO, WARNING, ALERT), and the dashboard displays them through dynamically rendered templates like `index.html` and `log_table.html`. This logging system operates independently of the main control flow, ensuring that diagnostics and event tracking remain functional even during high-load operations.

The entire codebase is structured to allow new modules or hardware to be added with minimal disruption. Each functional block is designed to operate independently, with communication handled through shared memory, modular imports, and clean interface boundaries. This architecture makes the Astraeus software system not only functional and reliable, but also maintainable and scalable for future development.

3.4.2 Main Execution and Control Logic

The core runtime of the Astraeus rover is governed by two primary scripts: `start_all.py` and `main.py`. These files coordinate how the system boots, how control is shared between manual and autonomous modes, and how commands are interpreted and executed in real-time.

start_all.py – System Bootstrapper

The start_all.py script serves as the system's bootstrapper. It uses Python's multiprocessing module to simultaneously launch two independent processes: the Flask-based user interface and the rover's main control loop.

```
32         # Initialize shared memory once here
33         manager = Manager()
34         shared = manager.dict({
35             "command": None,
36             "speed": 40,
37             "mode": "idle",
38             "task": None
39         })
```

Figure 51. Shared dictionary setup in start_all.py for inter-process communication

This shared dictionary is initialized once and passed to both processes. It holds the current drive command, speed setting, operation mode (manual or autonomous), and any selected task. This architecture ensures synchronized communication between the web interface (app.py) and the main control logic (main.py), even though they run independently.

The processes are launched as follows:

```
43         p1 = Process(target=run_flask, args=(shared,))
44         p2 = Process(target=run_main, args=(shared,))
45
46         p1.start()
47         p2.start()
```

Figure 52. Dual-process launch for Flask and control loop in start_all.py

This structure ensures that both the control system and the web server are live from the moment the program starts. If either process terminates, the other will continue to run unless explicitly terminated.

main.py – Control Decision Loop

The main.py file contains the central control loop that continuously checks the shared state to determine the rover's operational mode and execute corresponding logic. The entry point is the main_loop() function.

```
85     while True:
86         mode = get_mode()
87
88         if mode != last_mode:
89             print(f"[MAIN] Mode switched to: {mode}")
90             last_mode = mode
91
92         if mode == "manual":
93             if not autonomy_busy:
94                 handle_manual()
95             else:
96                 print("[MAIN] Ignored manual input - autonomy is active.")
97
98         elif mode == "autonomous":
99             handle_autonomy()
100
101         time.sleep(0.2)
```

Figure 53. Main control loop in main.py handling mode-based task delegation

Every 200 milliseconds, the loop polls the current mode (manual or autonomous) and performs the appropriate action. This mode is set by the Flask interface and accessed through shared memory via command_center.py.

Manual Mode Handling

In manual mode, the handle_manual() function reads the current command and speed from the shared dictionary:

```
19     def handle_manual():
20         cmd = get_command()
21         speed = get_speed()
```

Figure 54. Manual command and speed retrieval from shared memory in main.py

Depending on the command received—such as movement instructions (forward, left, stop), alignment requests (align1, align2), or cleaning sequences (seq1, seq2), the appropriate

control routine is triggered. Once executed, the command is cleared to prevent it from being repeated.

Autonomous Mode Handling

In autonomous mode, the `handle_autonomy()` function checks whether the system is already running a task. If idle and a new task is detected in the shared memory, it is passed into the `run_autonomy()` function.

```
73     task = get_task()
74     if task and task != "None":
75         print(f"[AUTO] Starting task: {task}")
76         run_autonomy(task)
77         clear_task()
```

Figure 55. Task check and autonomous execution trigger in `main.py`

This structure ensures that only one task is executed at a time, and that manual overrides are ignored while autonomous functions are active.

Shared Memory and Command Routing

The `command_center.py` module abstracts access to the shared dictionary, allowing all scripts to interact with shared state in a clean and controlled manner.

```
3     def set_command(cmd):
4         get_shared()["command"] = cmd
5
6     def get_command():
7         return get_shared().get("command", None)
```

Figure 56. Command access functions in `command_center.py` using shared memory

This modular setup makes it easy to route and isolate data between components, whether it's setting a drive command from the web interface or reading a selected cleaning task from within the control loop.

Coordination and Isolation

The multiprocessing architecture ensures fault isolation between the interface and control logic. The Flask server can be restarted independently of the control system, and commands are buffered in shared memory to prevent missed instructions. This separation also simplifies debugging, as issues in one process do not halt the entire system.

This dual-process structure forms the foundation of the Astraeus software system. It ensures responsiveness to user input, real-time execution of control routines, and a clean separation between logic and interface, all while remaining lightweight and flexible enough to run on a Raspberry Pi 3.

3.4.3 Manual Control System

The manual control system provides a real-time interface for users to directly operate the Astraeus rover using a web dashboard. This system was built with one key design principle in mind: full containment. No external hosting services, scripts, or cloud-based dependencies are required to operate the system. Everything is served locally from the onboard Flask server, including the HTML frontend, CSS styling, JavaScript logic, and virtual joystick controls. This architecture ensures the rover can function reliably even in isolated environments without internet access.

Flask Server Architecture

The Flask server defined in `app.py` is responsible for serving the manual control interface and handling incoming HTTP POST requests. It exposes several key endpoints that allow the user to send movement commands, adjust motor speed, switch modes, and access other control functions.

One of the core endpoints is `/manual/command`, which receives movement instructions from the web interface and places them into shared memory:

```

67     @app.route("/manual/command", methods=["POST"])
68     def manual_command():
69         data = request.get_json()
70         cmd = data.get("command")
71         set_command(cmd)
72         print(f"[Manual CMD] Sent to buffer: {cmd}")
73         return jsonify({"status": "ok"}), 200

```

Figure 57. Route in app.py to receive manual movement commands.

This command is then picked up by the main loop in main.py when the system is in manual mode, allowing real-time execution of user input. The use of shared memory ensures that command state remains synchronized between the Flask process and the control loop process.

Speed Adjustment Endpoint

In addition to directional movement, the operator can adjust motor speed using a slider interface. This input is handled via the /manual/speed route:

```

75     @app.route("/manual/speed", methods=["POST"])
76     def set_speed_handler():
77         data = request.get_json()
78         speed = int(data.get("speed", 40))
79         set_speed(speed)
80         print(f"[Speed] Updated in buffer: {speed}%")
81         return jsonify({"status": "updated", "speed": speed}), 200

```

Figure 58. Speed update route in app.py for manual mode.

This allows the user to tailor responsiveness of the rover to match surface conditions or task sensitivity. The updated speed value is stored in shared memory and applied to all future movement commands until it is changed again.

Local Hosting of Joystick Library

A major goal in the development of this system was to ensure full local functionality, even without an internet connection. As part of that goal, the joystick interface was implemented using

the open-source nipplejs library, but instead of loading it from an external CDN, the full library was downloaded and stored locally in the project's static directory. This guarantees that the virtual joystick will load and function correctly regardless of internet access.

The script is included in the HTML as follows:

```
7      <script src="{{ url_for('static', filename='js/nipplejs.min.js') }}"></script>
```

Figure 59. Local inclusion of nipplejs in the HTML interface.

This ensures that the joystick logic is served directly from the Raspberry Pi running the Flask server, further reinforcing the autonomy and self-reliance of the rover system.

Joystick Input Handling

The virtual joystick is defined and configured in joystick.js, which continuously monitors input direction and magnitude. It calculates a command string such as "forward", "backward", "left", or "right" and sends it to the backend through a fetch request. This system ensures smooth transitions and responsiveness, rather than relying on discrete button presses.

```
111      function sendCommand(cmd) {
112          fetch('/manual/command', {
113              method: 'POST',
114              headers: { 'Content-Type': 'application/json' },
115              body: JSON.stringify({ command: cmd })
116          });
117      }
```

Figure 60. JavaScript function sending movement commands to Flask.

While the joystick is being held, the script continuously sends commands to the backend every few milliseconds. When the joystick is released, a "stop" command is sent, halting movement immediately.

This behavior ensures responsive manual control while also allowing the user to make precise adjustments. Any delay or packet loss is minimized by the fact that all communication occurs on the local network and within a single device in most use cases.

Mode Switching

Mode control is handled via another route, `/set_mode`, which allows the operator to toggle between manual and autonomous modes. The selected mode is written to shared memory and governs how the control loop in `main.py` behaves.

```
83     @app.route("/set_mode", methods=["POST"])
84     def set_mode_route():
85         data = request.get_json()
86         mode = data.get("mode")
87         if mode in ["manual", "autonomous"]:
88             set_mode(mode)
89             print(f"[Mode] Updated to {mode}")
90             return jsonify({"status": "ok", "mode": mode}), 200
91         return jsonify({"status": "error", "message": "Invalid mode"}), 400
```

Figure 61. Mode selection route in `app.py`.

When manual mode is selected, the main loop immediately begins polling for joystick commands and speed updates, while pausing any autonomous tasks or tag-following logic.

Real-Time Responsiveness and Loop Timing

Due to the lightweight design and use of local memory, the time between joystick input and motor movement is minimal. The control loop runs on a 200 ms delay, and the joystick sends continuous updates while active. This enables smooth, responsive handling, even during fine control tasks such as alignment or obstacle navigation in manual mode.

This architecture ensures that the manual control system is not only functional and responsive but also completely self-sufficient. All frontend and backend elements are hosted internally, making the system portable, reliable, and robust in offline or field environments.

3.4.4 Autonomous Navigation and Tag Alignment

The autonomous behavior of the Astraeus rover is coordinated through two core modules: `autonomy.py` and `visual_module.py`. These scripts work together to manage visual tag tracking, alignment, and stage transitions within a cleaning sequence. Tags are used solely for tracking and alignment, not for task identification. The system executes one task at a time and uses visual

feedback to precisely position itself before engaging cleaning actions or repositioning maneuvers.

Task Detection and Execution

Autonomous mode is activated by selecting a task from the web interface and switching the mode to "autonomous". When this happens, main.py checks the shared memory for a queued task and launches the control flow by calling run_autonomy(task) from autonomy.py.

```
74         task = get_task()
75         if task and task != "None":
76             print(f"[AUTO] Starting task: {task}")
77             run_autonomy(task)
78             clear_task()
```

Figure 62. main.py triggers run_autonomy() when a task is detected.

Each task represents a complete sequence such as aligning to a panel, cleaning it, and repositioning. Tag IDs (e.g., Tag 2 for Panel 1, Tag 3 for Panel 2) are associated with alignment only, not for identifying which task to perform.

AprilTag Detection Using HuskyLens

The rover uses a HuskyLens camera for detecting AprilTags via I2C. The visual_module.py module interfaces with the HuskyLens using the HuskyLensLibrary. The function get_tag_data(tag_id) requests all visible tags, scans for a specific ID, and returns its position if found.

```

19  ✓ def get_tag_data(tag_id):
20      """
21      Returns smoothed data for a given tag_id.
22      Output: dict with x, y, width, height or None if tag not found.
23      """
24      try:
25          huskylens.requestAll()
26          blocks = huskylens.blocks()
27          if not isinstance(blocks, list):
28              blocks = [blocks]
29
30          for block in blocks:
31              if hasattr(block, 'ID') and block.ID == tag_id:
32                  data = {'x': block.x, 'y': block.y, 'w': block.width, 'h': block.height}
33                  _buffers[tag_id].append(data)
34                  return _get_smoothed(tag_id)
35      except Exception as e:
36          print(f"[visual_module] Error: {e}")
37      return None

```

Figure 63. Detection and buffering of April Tag data in visual_module.py.

This positional data is stored in a tag-specific rolling buffer and smoothed before being used to determine adjustments.

Tag Data Smoothing

To ensure steady and jitter-free feedback, visual_module.py uses a simple averaging function over the last five readings for each tag. This minimizes unwanted corrections from noisy data.

To calculate the smoothed value of each coordinate (e.g., x , y , width, or height), a simple moving average is applied over the last n readings. The formula is defined as:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

Where:

- \bar{x} is the smoothed (averaged) value,
- x_i is the i^{th} recorded value,
- n is the total number of recent readings (in this case, $n = 5$).


```

39  ✓ def _get_smoothed(tag_id):
40      """
41      Returns the average of x, y, w, h from the buffer for the given tag.
42      """
43      buf = _buffers[tag_id]
44      if not buf:
45          return None
46      avg = {
47          'x': round(sum(d['x'] for d in buf) / len(buf), 2),
48          'y': round(sum(d['y'] for d in buf) / len(buf), 2),
49          'w': round(sum(d['w'] for d in buf) / len(buf), 2),
50          'h': round(sum(d['h'] for d in buf) / len(buf), 2)
51      }
52      return avg

```

Figure 64. Smoothed average output of tag data to stabilize alignment.

Approaching the Solar Panel – drive_until_tag_detected()

The autonomous navigation process begins with the rover executing the `drive_until_tag_detected()` function. The primary objective of this function is to bring the rover within a usable alignment range of a solar panel, where AprilTag detection is stable, and proximity is close enough to begin fine adjustments. This function acts as the entry point to the full autonomous task pipeline.

At the start of the routine, the rover rotates slowly in place until it detects Tag 1, which serves as the approach marker for solar panel tracking. Once Tag 1 is found and its width exceeds a small threshold (indicating it's visible but still distant), the rover begins driving forward. As it moves, it continually checks for the presence of the target panel tag (either Tag 2 or Tag 3 depending on the selected task).

The rover prioritizes transitioning from Tag 1 to the actual panel tag as soon as it is seen. Once the target panel tag becomes visible and passes a defined minimum width (indicating the rover is within reasonable tracking range), forward motion stops and alignment can begin.

This process includes several built-in fail-safes:

- If the target tag is not seen within 12 seconds, the rover stops and logs a timeout event.
- If no tag is visible for 6 seconds during approach, the rover exits and returns False.
- If the rover's IR sensors report that it is already too close to the panel, the function will not proceed further to avoid collision.

The loop continuously evaluates these visual and distance conditions to determine if it's appropriate to transition into alignment.

This function plays a critical role in the autonomy logic:

- It serves as a search and approach phase, ensuring the panel tag is located from a wide angle.
- It guarantees the rover is in an optimal starting position for alignment, facing the tag head-on and within both visual and IR range.

Once `drive_until_tag_detected()` exits successfully, it returns `True`, signaling that the rover is now ready for precise alignment using the `alignment_sequence()` function. The full implementation of this function can be found in Appendix H.3.

Alignment Process – `alignment_sequence(tag_id)`

Once the rover is in close proximity to the solar panel, the `alignment_sequence(tag_id)` function is called to perform precise orientation and positioning. This function is responsible for centering the rover directly in front of the panel using both visual and IR sensor feedback. The alignment logic operates in a continuous loop, reading smoothed AprilTag data from the camera (including the tag's x-position and width), along with distance values from both the left and right IR sensors. These inputs are analyzed together to determine if the rover is centered horizontally, close enough to the panel, or needs to back up and retry.

The alignment strategy is based on a staged controller. If the tag is visible but offset from the horizontal center of the frame (162 pixels), the rover turns in place using asymmetric motor speeds to bring it back on course. When the tag is centered and the rover is not yet too close, it slowly advances forward. As the rover gets closer, the tag width increases, and if it exceeds a set threshold, or if the IR sensors indicate the rover is less than ~17 cm away—the system will reverse slightly and retry the alignment. The alignment is only considered successful when the rover holds the correct position (tag centered, distance matched, and tag stable) for at least two continuous seconds.

To protect against failure, the routine allows up to three alignment attempts. If the tag disappears or the rover is misaligned repeatedly, it will back out and try again. After three failed attempts, the function exits and logs a failure condition. On success, the rover stops all

movement and returns control to the main task sequence, now correctly positioned to begin cleaning.

This alignment stage is essential to ensure that the cleaning hardware can operate reliably. Without accurate alignment, the brush or arm mechanisms may miss the panel entirely or apply uneven pressure. By combining real-time camera feedback with distance sensing and logic for retries, `alignment_sequence()` ensures that the rover only proceeds when it's in a confirmed optimal position. The full implementation of this function can be found in Appendix H.3.

Cleaning and Return-to-Home Behavior

After a successful alignment, the rover transitions to the cleaning phase. At this point, it is correctly positioned in front of the solar panel, and the autonomous logic calls a specific cleaning sequence using `trigger_sequence()` and `run_cleaner()`. These functions interact with the Maestro servo controller via `maestro_module.py`, sending predefined commands that initiate motion of the cleaning mechanism—typically a brush or arm. The sequence includes a brief delay, then activates the cleaning motor at a set speed for a fixed duration (40 seconds in the current configuration). Once complete, the system stops the cleaning motor and proceeds to the next stage.

After cleaning, the rover must reposition itself. For tasks involving only one panel (such as "panel1" or "panel2"), the rover calls the `return_home()` function. This function initiates a 180-degree turn using timed motor commands, then begins searching for Tag 4, which represents the home base. Once Tag 4 is detected, the rover switches to the `follow_tag(tag_id=4)` routine to track and approach the tag. This function uses real-time tag data to dynamically adjust motor speeds, keeping the rover aligned with the tag's center while driving forward.

The tag-following continues until the tag width exceeds a predefined threshold, indicating that the rover is close enough to stop. Once within range, the rover stops all movement, ending the autonomous task.

This structured sequence, cleaning followed by return-to-home, ensures the rover resets to a known state after each operation. It also improves usability by positioning the rover in an accessible location for recharge, maintenance, or manual control. The logic also avoids dead

ends by using tag-based homing instead of hardcoded paths, making the return phase more robust and environment-aware.

The implementation of these post-alignment behaviors can be found in Appendix H.3.

```
362         if task == "panel1":
363             _e_stop_enabled = True
364             if not drive_until_tag_detected(1, 2, timeout=20):
365                 fail_and_switch_to_manual("Lost sight of both Tag 1 and Tag 2 during approach")
366                 return
367             _e_stop_enabled = False
368
369             if not alignment_sequence(2):
370                 fail_and_switch_to_manual("Alignment with Tag 2 failed")
371                 return
372
373             log_event("Sequence 1 initiated. On standby for 40 seconds.")
374             trigger_sequence(1)
375             time.sleep(2)
376             motors.run_cleaner(100)
377             time.sleep(43)
378             motors.stop_cleaner()
379             time.sleep(4)
380
381             log_event("Returning home...")
382             return_home()
```

Figure 65. Task sequence for "panel1" in autonomy.py, showing tag detection and alignment steps, with fallback to manual mode if either stage fails.

This modular design allows each stage to be tested independently and updated without affecting the others.

3.4.5 Motor and Movement Control

The movement system of the Astraeus rover is built around a simple but effective abstraction of PWM-based motor control. Each of the drive motors is wired to a Pololu VNH5019 motor driver, which receives directional and speed commands via the Raspberry Pi's GPIO pins. The motors.py module contains all logic necessary to control these motors, starting from individual motor movement up to coordinated drivetrain commands.

Core Motor Control

At the heart of the system is the `_set_single_motor()` function. This function accepts a motor object, its corresponding PWM output device, and a speed value between -100 and 100. Positive values make the motor spin forward, negative values spin it in reverse, and a value of 0 stops the motor entirely. The PWM signal is scaled based on the absolute value of the speed input, and the direction logic is handled through the `gpiozero` library's `.forward()` and `.backward()` methods.

```
33  ✓ def _set_single_motor(motor, pwm_device, speed, label=""):
34      speed = max(-100, min(100, speed))
35      duty = abs(speed) / 100.0
36      direction = "Stopped"
37
38      if speed > 0:
39          motor.forward()
40          pwm_device.value = duty
41          direction = "Forward"
42      elif speed < 0:
43          motor.backward()
44          pwm_device.value = duty
45          direction = "Reverse"
46      else:
47          motor.stop()
48          pwm_device.value = 0.0
49
50      if label:
51          print(f"[MOTOR] {label} => Speed: {abs(speed)}% | Direction: {direction}")
```

Figure 66. `_set_single_motor()` applies a direction and PWM value based on the requested speed.

This modular structure allows for highly reusable motor control logic. Whether the command is for a single wheel, both drive motors, or the cleaning motor, they all pass through this same function.

Manual Mode Movement

In manual mode, movement commands are received from the web interface via joystick input. These commands are passed to the `send_drive_command(cmd, speed)` function in `motors.py`, which interprets them and calls `set_motor_speed(left, right)` with appropriate values. The

available string-based commands include “forward,” “backward,” “left,” “right,” as well as diagonal and curved options like “forward_left” or “backward_right.”

```
57  def send_drive_command(cmd, speed):
58      if cmd == "forward":
59          set_motor_speed(speed, speed)
60      elif cmd == "backward":
61          set_motor_speed(-speed, -speed)
62      elif cmd == "left":
63          set_motor_speed(-speed, speed)
64      elif cmd == "right":
65          set_motor_speed(speed, -speed)
66      elif cmd == "forward_left":
67          set_motor_speed(speed // 2, speed)
68      elif cmd == "forward_right":
69          set_motor_speed(speed, speed // 2)
70      elif cmd == "backward_left":
71          set_motor_speed(-(speed // 2), -speed)
72      elif cmd == "backward_right":
73          set_motor_speed(-speed, -(speed // 2))
74      elif cmd == "stop":
75          stop_motors()
76      else:
77          print(f"[WARN] Unknown command: {cmd}")
```

Figure 67. Centralized drive command routing in motors.py for manual input handling.

This abstraction ensures that all manual controls rely on the same low-level speed functions, keeping motor behavior predictable and easy to debug.

Autonomous Mode Usage

In autonomous mode, `send_drive_command()` is not used. Instead, functions like `set_motor_speed(left, right)` are called directly to achieve precise, frame-by-frame adjustments. For example, while aligning to a tag, the system calculates the offset between the tag’s position and the center of the frame and converts that into an adjustment factor. The adjusted speeds are then passed directly to `set_motor_speed()`:

```

70         if primary_data:
71             last_seen_time = time.time() #reset loss timer
72             x = primary_data['x']
73             error = x - CENTER_X
74             correction = min(1.0, abs(error) / MAX_DEVIATION)
75             adjustment = int(correction * base_speed * steer_aggression)
76
77             if error > 0:
78                 left = base_speed - adjustment
79                 right = base_speed + adjustment
80             else:
81                 left = base_speed + adjustment
82                 right = base_speed - adjustment
83
84             motors.set_motor_speed(left, right)

```

Figure 68. Fine-grained motor control in autonomous tag tracking.

This approach gives the autonomy system precise control over rover movement, while still reusing the exact same motor logic as manual mode. It allows for gradual turns, course corrections, and safe braking behavior, without any need to redefine how motors are controlled.

3.4.6 Sensor Integration and Obstacle Detection

To enable short-range obstacle detection, the Astraeus rover utilizes two Sharp analog infrared distance sensors, one positioned on the left and the other on the right front edge of the chassis. These analog signals are read using the ADS1115 16-bit ADC over the I²C bus, with the integration logic handled entirely in `sharp_sensor.py`. The system continuously monitors these sensors to prevent collisions and supports both visual guidance and physical proximity safeguards during autonomous operation.

The core of the conversion process involves reading the raw voltage from each sensor and converting that to distance using a calibrated mathematical formula. This formula accounts for the non-linear response of the Sharp IR sensor and outputs a usable approximation in centimeters. Voltages below a certain threshold are considered “out of range,” returning infinity to signal no nearby obstacle.

```

33     def voltage_to_distance(voltage):
34         if voltage <= 0.1:
35             return float('inf') #Out of range
36         return round(27.86 / (voltage - 0.1), 2)

```

Figure 69. Voltage-to-distance conversion used to interpret raw Sharp sensor readings.

To stabilize the data and reduce the effect of transient fluctuations, the module employs a two-tiered smoothing approach. First, each new distance reading is added to a buffer (deque) of the last 10 values, from which a running average is computed. Then, this average is passed through an exponential moving average (EMA) filter, which gives more weight to recent readings while still retaining the influence of past values. The result is a smoothed, real-time distance output that updates predictably and responsively.

```

49     left_ema = left_dist if left_ema is None else alpha * left_dist + (1 - alpha) * left_ema
50     right_ema = right_dist if right_ema is None else alpha * right_dist + (1 - alpha) * right_ema

```

Figure 70. Exponential moving average (EMA) for smoothing noisy distance data.

These final, filtered readings are made available via the `get_smoothed_distances()` function, which is frequently called within `autonomy.py`, especially during tag alignment routines. The IR sensors provide a physical check that complements visual detection, ensuring alignment doesn't proceed unless the rover is at a safe and effective cleaning distance.

For obstacle detection, the function `is_obstacle_detected()` compares the smoothed IR values to a defined threshold (typically 40 cm). If either sensor detects an object within that range, an obstacle is flagged. This mechanism is further expanded in the `emergency_stop_check()` function, which acts as a safety supervisor. When an obstacle is confirmed and no recent cooldown is active, this function stops the motors, logs a warning, and waits five seconds before reevaluating the environment. If the obstacle is still present after 20 seconds, the system logs an alert and remains halted.


```

60  ✓ def emergency_stop_check(threshold_cm=40.0, cooldown=7):
61      global _last_triggered, _obstacle_start_time
62
63      now = time.time()
64      if now - _last_triggered < cooldown:
65          return
66
67      if is_obstacle_detected(threshold_cm):
68          print("Obstacle detected! Pausing movement...")
69          log_warning("Obstacle detected! Pausing movement...")
70          motors.stop_motors()
71
72          if _obstacle_start_time is None:
73              _obstacle_start_time = now
74          elif now - _obstacle_start_time > 20:
75              print("Obstacle still present after 1 minute! Logging alert...")
76              log_alert("Obstacle still present after 1 minute! Logging alert...")
77
78          time.sleep(5)
79          _last_triggered = time.time()
80      else:
81          _obstacle_start_time = None # Reset timer if path is clear

```

Figure 71. Emergency stop routine triggered when a nearby object is detected.

These safeguards are particularly important during autonomous routines like `alignment_sequence()`, where the rover moves closer to a solar panel using visual cues. If visual alignment appears successful but the IR sensors disagree, the system will prioritize safety and back off to avoid impact. This layered logic makes the rover more resilient in unpredictable environments and enhances its ability to operate autonomously without requiring constant oversight.

The complete implementation of this sensor integration and safety logic can be found in Appendix H.5.

3.4.7 External Subsystem Control

The Astraeus rover communicates with an external Pololu Maestro servo controller to operate its mechanical subsystems, such as activating a cleaning brush or lifting mechanism. To simplify this communication, a custom Python module named `maestro_module.py` was created. This

module serves as a lightweight interface between the Raspberry Pi and the Maestro via USB serial.

At the core of the module is the `trigger_sequence(subroutine)` function, which sends a predefined 4-byte serial command following Pololu's official Serial Script Protocol. Each subroutine corresponds to a scripted behavior already programmed into the Maestro using its configuration software.

```
35     command = [0xAA, DEVICE_NUMBER, 0x27, subroutine]
36     try:
37         with serial.Serial(PORT, BAUDRATE, timeout=1) as maestro:
38             maestro.write(bytearray(command))
39             print(f"[Maestro] Triggered subroutine {subroutine}")
40     except Exception as e:
41         print(f"[Maestro] Error triggering subroutine {subroutine}: {e}")
```

Figure 72. Function to trigger subroutine execution on the Pololu Maestro via serial command.

This modular setup allows autonomous routines, such as panel cleaning, to call external hardware sequences simply by invoking `trigger_sequence()` with the appropriate subroutine number. This keeps the Python-side logic clean and avoids embedding low-level serial commands throughout the main control flow. The complete implementation of this module can be found in Appendix H.7.

3.4.8 Event Logging and Dashboard Interface

The Astraeus rover features a fully self-contained web dashboard, hosted locally on the Raspberry Pi using Flask. This dashboard was designed as both a control and monitoring interface, offering real-time feedback and full interaction without needing internet access. All necessary files, including HTML, CSS, JavaScript, and logging logic, are stored and served from the Raspberry Pi itself. This design ensures the interface remains functional in remote locations and field deployments.

Upon visiting the dashboard, users first land on the Home tab, which serves as the central landing point. From this screen, users are presented with system status information and navigational options. They can choose to enter either the Mode Selection tab or the Data Logging tab, depending on whether they wish to operate the rover or review system activity.

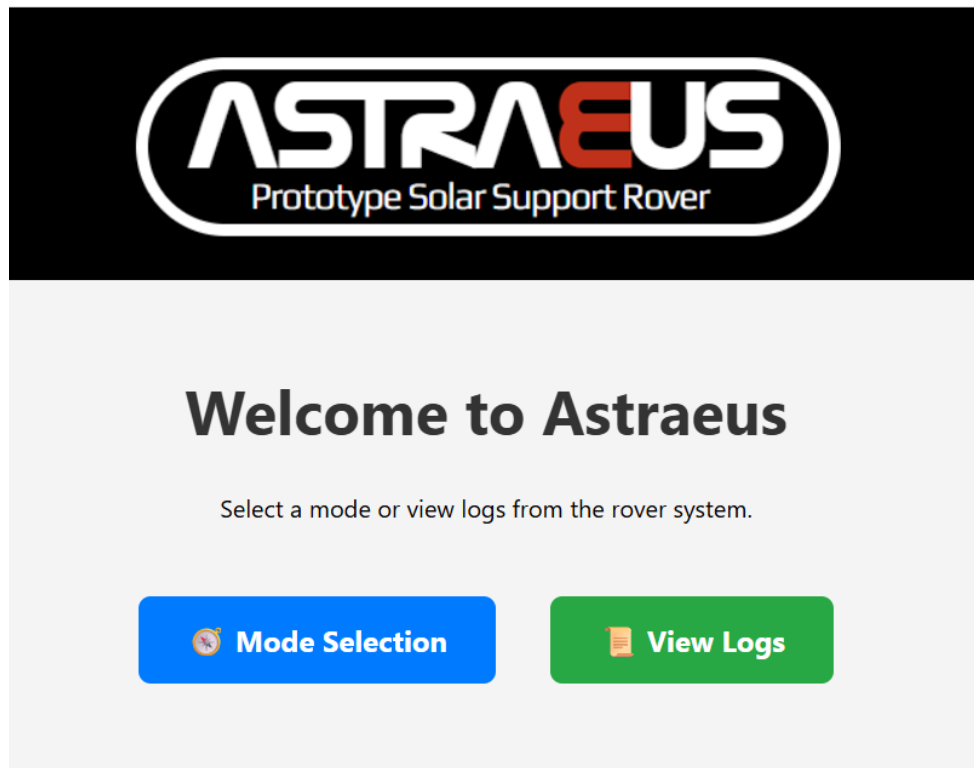


Figure 73. Home Tab here to show the starting point of the UI.

Navigating to the Mode Selection tab, users can switch between manual and autonomous operation. In manual mode, the interface provides a full control panel featuring a locally hosted virtual joystick powered by the nipplejs library. By downloading and storing this library locally, the entire UI remains fully offline and self-reliant. The joystick input is sent to the Flask backend and used to issue real-time drive commands to the motors. This design ensures tight responsiveness and eliminates external dependencies.

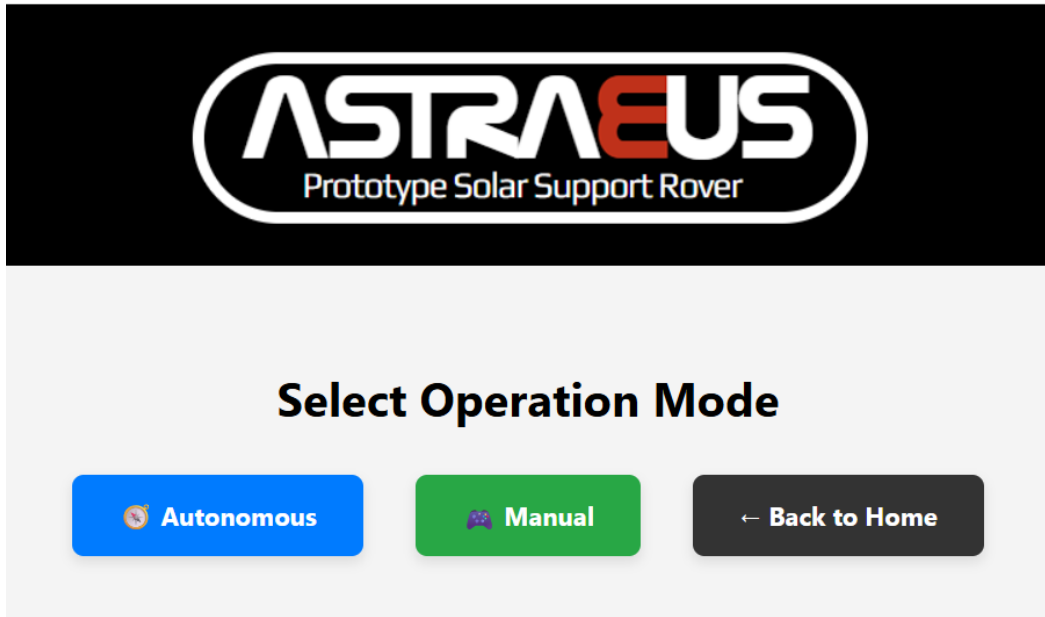


Figure 74. Select Operation Mode screen.

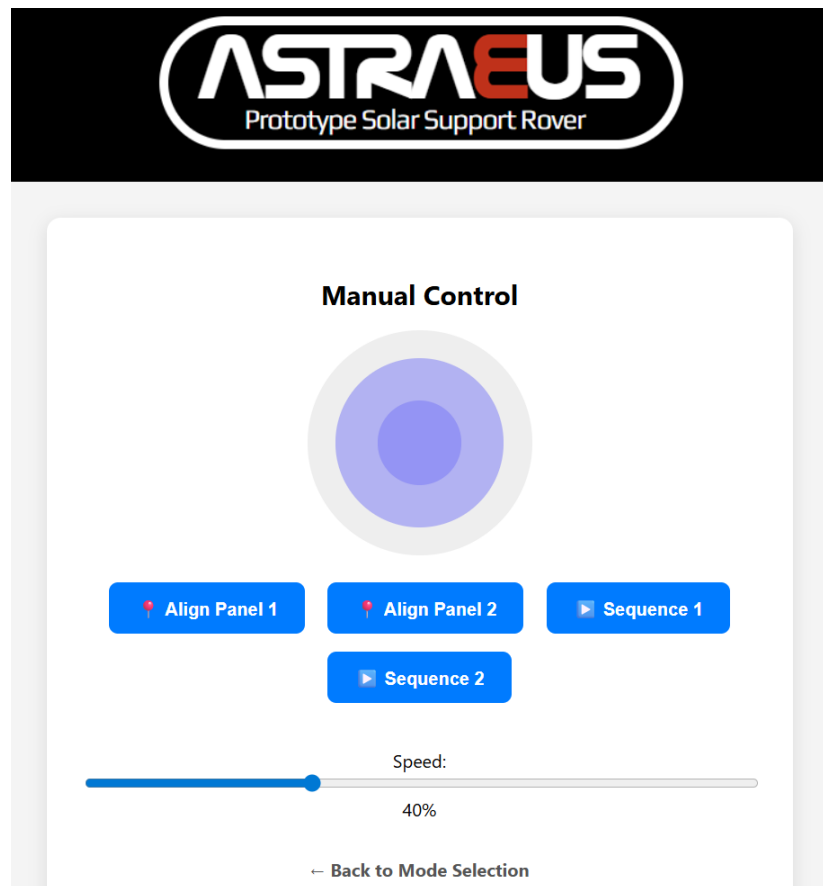


Figure 75. Manual Mode Control Interface with Joystick after this paragraph.

When users select Autonomous mode, they are prompted to choose from three available task options: clean Panel 1, clean Panel 2, or clean both panels sequentially. Once a selection is made, the rover executes the appropriate sequence autonomously using a combination of vision, sensors, and mechanical subsystems. The interface also indicates the active task and tracks progress, helping the user understand what the system is currently executing.

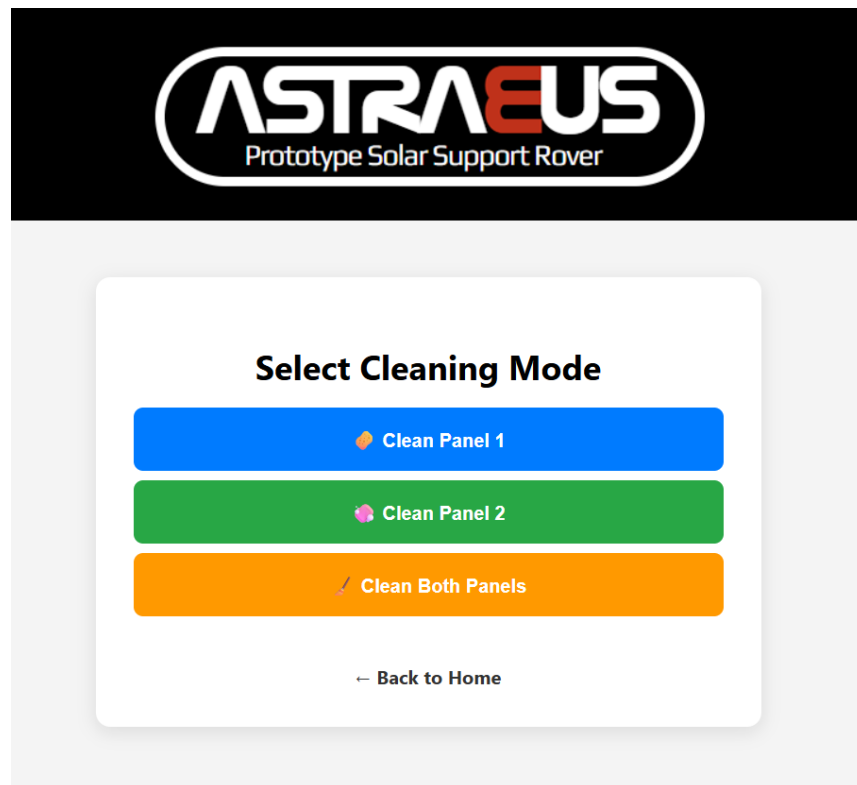
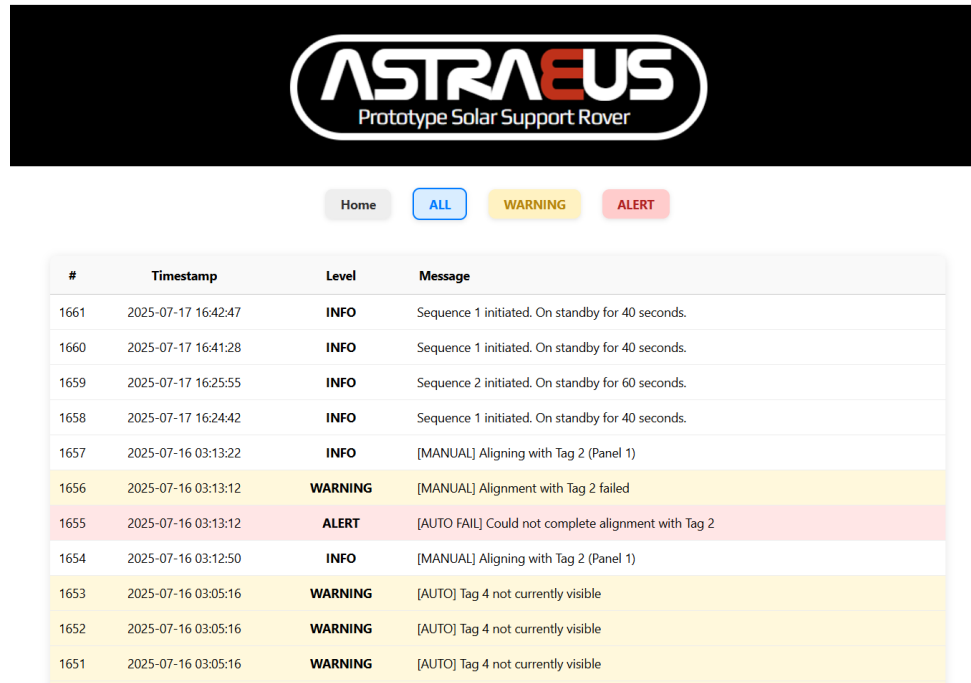


Figure 76. Autonomy Mode Selection Screen here.

Alternatively, users can access the Data Logging tab, which offers real-time insight into the rover's operations. The system logs all notable events, such as tag detection, alignment status, obstacle warnings, and cleaning activation, into an SQLite database. These logs are categorized into INFO, WARNING, and ALERT levels and are displayed in an organized table. Users can filter by severity or view all entries, and the page updates automatically without requiring a manual refresh.

The logging interface also includes a floating “new messages” notification. If the user is scrolled up in the log history and new messages are received, this banner appears at the bottom

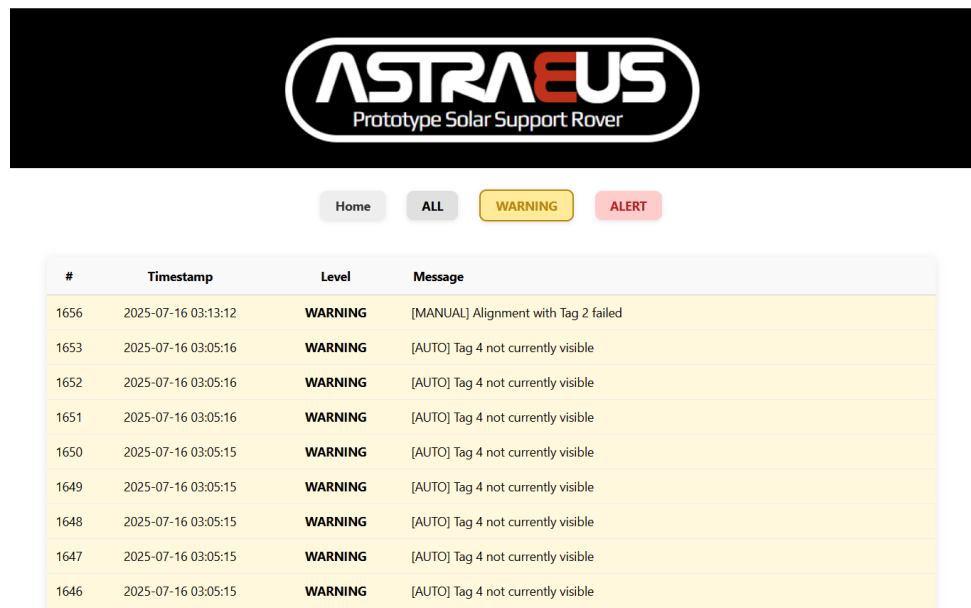
of the screen to ensure the user is immediately informed of recent system activity. This feature keeps the operator aware of important updates without disrupting their current view.



The screenshot displays the ASTRAEUS interface with a black header bar containing the logo. Below the header is a navigation bar with four buttons: 'Home' (grey), 'ALL' (blue), 'WARNING' (yellow), and 'ALERT' (red). The 'ALL' button is selected. The main area shows a table of log entries with four columns: '#', 'Timestamp', 'Level', and 'Message'. The table contains 11 rows of data, with rows 1656, 1655, and 1651 highlighted in yellow, and row 1655 highlighted in red.

#	Timestamp	Level	Message
1661	2025-07-17 16:42:47	INFO	Sequence 1 initiated. On standby for 40 seconds.
1660	2025-07-17 16:41:28	INFO	Sequence 1 initiated. On standby for 40 seconds.
1659	2025-07-17 16:25:55	INFO	Sequence 2 initiated. On standby for 60 seconds.
1658	2025-07-17 16:24:42	INFO	Sequence 1 initiated. On standby for 40 seconds.
1657	2025-07-16 03:13:22	INFO	[MANUAL] Aligning with Tag 2 (Panel 1)
1656	2025-07-16 03:13:12	WARNING	[MANUAL] Alignment with Tag 2 failed
1655	2025-07-16 03:13:12	ALERT	[AUTO FAIL] Could not complete alignment with Tag 2
1654	2025-07-16 03:12:50	INFO	[MANUAL] Aligning with Tag 2 (Panel 1)
1653	2025-07-16 03:05:16	WARNING	[AUTO] Tag 4 not currently visible
1652	2025-07-16 03:05:16	WARNING	[AUTO] Tag 4 not currently visible
1651	2025-07-16 03:05:16	WARNING	[AUTO] Tag 4 not currently visible

Figure 77. Full Log Table View (All Levels)



The screenshot displays the ASTRAEUS interface with the same header and navigation bar as Figure 77. In this view, the 'WARNING' button (yellow) is selected. The log table now only displays 10 rows, all of which are warnings. The rows are highlighted in yellow.

#	Timestamp	Level	Message
1656	2025-07-16 03:13:12	WARNING	[MANUAL] Alignment with Tag 2 failed
1653	2025-07-16 03:05:16	WARNING	[AUTO] Tag 4 not currently visible
1652	2025-07-16 03:05:16	WARNING	[AUTO] Tag 4 not currently visible
1651	2025-07-16 03:05:16	WARNING	[AUTO] Tag 4 not currently visible
1650	2025-07-16 03:05:15	WARNING	[AUTO] Tag 4 not currently visible
1649	2025-07-16 03:05:15	WARNING	[AUTO] Tag 4 not currently visible
1648	2025-07-16 03:05:15	WARNING	[AUTO] Tag 4 not currently visible
1647	2025-07-16 03:05:15	WARNING	[AUTO] Tag 4 not currently visible
1646	2025-07-16 03:05:15	WARNING	[AUTO] Tag 4 not currently visible

Figure 78. Warnings Filtered View

#	Timestamp	Level	Message
1655	2025-07-16 03:13:12	ALERT	[AUTO FAIL] Could not complete alignment with Tag 2
1407	2025-07-16 02:59:53	ALERT	[AUTO FAIL] Alignment with Tag 2 failed - switching to manual mode.
1406	2025-07-16 02:59:53	ALERT	[AUTO FAIL] Cannot align due to close proximity.
1349	2025-07-15 22:46:31	ALERT	[AUTO FAIL] Tag 2 lost too many times during alignment.
1313	2025-07-15 21:36:39	ALERT	[AUTO FAIL] Tag 2 lost too many times during alignment.
1310	2025-07-15 21:35:49	ALERT	[AUTO FAIL] Tag 2 lost too many times during alignment.
1307	2025-07-15 21:32:05	ALERT	[AUTO FAIL] Lost sight of both Tag 1 and Tag 2 during approach - switching to manual mode.
1306	2025-07-15 21:32:05	ALERT	[ALERT] Tag 2 lost: Timed out after 20s without seeing any tag
1242	2025-07-15 21:19:42	ALERT	[AUTO FAIL] Tag 2 lost too many times during alignment.

Figure 79. Alerts Highlighted in Red

#	Timestamp	Level	Message
1562	2025-07-16 03:05:02	INFO	[AUTO] Tag 4 visible X=124.0 W=31.2

Load Older Logs

Figure 80. Load Older Log Button, no new messages were available at the time so no “New Message” Pop up.

On the backend, the logging system is managed by `logger.py`, which defines modular functions for logging events from any part of the codebase. These functions—`log_info()`, `log_warning()`, and `log_alert()`, record entries with timestamps and severity levels into the SQLite database (`log.db`). Logging occurs throughout the autonomy pipeline, sensor monitoring, safety routines, and even manual actions, making it easy to trace what the system did and when.

Altogether, the dashboard and logging interface offer a robust and intuitive way to interact with the rover. Whether driving manually, launching an autonomous routine, or reviewing past events, the user is given full transparency and control in one integrated system. The implementation details of the Flask server and the event logging system can be found in Appendix H.8.

3.4.9 Safety and Fail-Safe Mechanisms

Safety was a core consideration throughout the development of the Astraeus rover's software. Given its autonomous capabilities and interaction with physical hardware, a layered system of fail-safes was designed to prevent collisions, halt unsafe operations, and recover gracefully from software or communication faults.

The primary safety mechanism is the obstacle detection and emergency stop system, implemented through the `sharp_sensor.py` module. If either of the IR distance sensors detects an object within a predefined threshold, the `emergency_stop_check()` function halts all movement, logs a warning, and enforces a brief cooldown period. If the obstruction persists, the system logs an alert and remains in a safe, stopped state. This feature ensures that even if a visual tag is being followed, the rover will never collide with an unexpected object due to poor lighting or misalignment.

Complementing this are timeout and retry systems scattered across key control modules. For example, in `autonomy.py`, if a tag is lost for too long or if alignment fails after three attempts, the system exits the task early and logs the failure. Similarly, the return-to-home sequence has built-in tag reacquisition attempts and timeout logic, preventing infinite loops or stalling behavior in the event of lost visual input.

Another important safeguard is the cooldown timers on critical subsystems, like cleaning motors. These prevent the hardware from being reactivated too quickly after a cleaning sequence finishes, protecting both the motors and the battery from overuse or overheating.

Despite these software safeguards, the system also includes a fully independent hardwired recovery mode via `failsafe_manual.py`. This standalone script bypasses the web interface entirely and provides direct terminal-based keyboard control of the rover. Once launched, it enables full manual driving using WASD-style key input and also provides one-key access to critical operations such as running cleaning sequences or initiating tag alignment routines.


```
13  ✓ key_to_command = {  
14      'w': 'forward',  
15      'a': 'left',  
16      's': 'backward',  
17      'd': 'right',  
18      'q': 'forward_left',  
19      'e': 'forward_right',  
20      'z': 'backward_left',  
21      'x': 'backward_right'
```

Figure 81. Key-mapping dictionary in `failsafe_manual.py` that defines manual control input using standard keyboard keys.

Using the terminal-based `curses` interface, the user receives printed feedback for each movement, action, or alignment attempt. If a crash occurs in the Flask web UI or if the Pi loses network connectivity, this script can still be run directly through SSH or a monitor and keyboard connection. It gives the operator the ability to safely navigate *Astraeus* out of an unsafe position or complete critical tasks when the main dashboard becomes unresponsive.

By combining software-based detection, timeout and retry logic, and a fully isolated manual fallback script, the *Astraeus* control system is able to operate safely in autonomous mode while still offering full manual override in emergency conditions. These features ensure that the system remains responsive, resilient, and recoverable under a wide range of failure scenarios. The full failsafe control script can be found in Appendix H.11.

3.4.10 Modular Design and Scalability

The software architecture of the *Astraeus* rover was intentionally designed with modularity as a core principle, enabling isolated development, flexible debugging, and scalable integration for future hardware and features. Each major system component, such as motors, sensors, autonomy, visual tracking, logging, and web control, was separated into its own dedicated Python module. This allowed for parallel development of each subsystem and ensured that any changes or upgrades to one component would not interfere with the functionality of others.

For example, motor control logic resides entirely in `motors.py`, which exposes clear functions like `set_motor_speed()` and `stop_motors()` that can be reused across both manual and autonomous routines. Similarly, `sharp_sensor.py` handles all aspects of analog IR input and smoothing, making it easy to swap in different sensor models or adjust filtering without

modifying core control logic. Vision-based tag detection is abstracted into `visual_module.py`, providing a clean interface for retrieving smoothed AprilTag data regardless of the underlying detection algorithm.

The separation of UI logic into `app.py` also means that interface changes or feature additions, like new control buttons or tabs, can be made without affecting any of the movement or sensor logic. Meanwhile, the event logging system (`logger.py`) operates independently, capturing events from anywhere in the codebase without requiring deep integration.

This clean separation of responsibilities makes the codebase highly extensible. If new hardware components are introduced, such as a GPS module, LIDAR, or additional actuators, they can be integrated by simply creating a new module or expanding existing ones. Because of the modular command structure, these additions would be able to hook into `main.py` or `autonomy.py` with minimal risk of breaking existing functionality. Similarly, new autonomy behaviors or task types can be added as additional functions without requiring a major overhaul of the control structure.

The design also lends itself well to future software growth. Enhancements such as adding telemetry logging, ROS integration, or cloud-based dashboards could be implemented as entirely new modules or services that interact with the existing framework. This forward-thinking structure ensures that *Astraeus* is not only reliable in its current state, but also adaptable to future use cases, competition rules, or field conditions.

Overall, the modular approach has made the codebase easier to manage, easier to test, and far more scalable, ensuring that *Astraeus* can continue to evolve without needing to be rewritten from the ground up.

Chapter 4

Non-Technical Issues

Summary

In this chapter, we will discuss the project timeline, proposed and actual budget, and the economic factors associated with the development of Astraeus. We will also address important non-technical issues such as environmental impact, health and safety concerns, ethical responsibilities, and sustainability. These considerations are vital to the success and integrity of any engineering project, and this chapter outlines how they were addressed during the design, construction, and testing phases of Astraeus.

4.1 Project Timeline

4.2 Budget

4.3 Health & Safety Considerations

4.4 Ethical Considerations

4.5 Environmental & Sustainability Considerations

4.6 Sustainability Considerations

4.1 Project Timeline

The Development of Astraeus was guided by clearly defined schedules broken into two phases: Proposal Phase (Spring 2025) and Design Phase (Summer 2025). Each phase was structured to ensure the group met the weekly objectives through research, development, and testing.

4.1.1 Proposal Phase (Spring 2025)

This phase focused highly on conceptual development of the project, team responsibility and coordination, and CAD rendering. The spring semester weekly timeline is displayed in Table 14,

Table 14. Spring Timeline

Week	Objective
Week 1: <i>1/6/25 - 1/13/25</i>	<ul style="list-style-type: none">- Review syllabus and project expectations- Determine future meeting schedule- Draft 2–3 project ideas with flowcharts and descriptions
Week 2: <i>1/13/25 - 1/20/25</i>	<ul style="list-style-type: none">- Discuss project ideas and website layout- Explore SEAR applications beyond space- Research semi-autonomous rover capabilities
Week 3: <i>1/20/25 - 1/27/25</i>	<ul style="list-style-type: none">- Finalize project concept- Assign roles and start organizing requirements- Begin report draft and project naming- Contact Dr. Carbone and UCF Exolith Lab
Week 4: <i>1/27/25 - 2/3/25</i>	<ul style="list-style-type: none">- Define mobility, autonomy, and communication constraints- Revise report to reflect updated project scope- Research brush system with static charge potential
Week 5: <i>2/3/25 - 2/10/25</i>	<ul style="list-style-type: none">- Review report Chapters 1–4- Discuss component selection and power budget- Finalize solar panel for testing- Transfer documentation to report
Week 6: <i>2/10/25 - 2/17/25</i>	<ul style="list-style-type: none">- Review Chapter 1 introduction and Chapter 2 integration- Align hardware/software functions with project goals

	<ul style="list-style-type: none"> - Finalize budget draft and project timeline - Continue website updates
<p>Week 7:</p> <p>2/17/25 - 2/24/25</p>	<ul style="list-style-type: none"> - Refine Chapters 3 (budget/timeline) and 4 (future expansions) - Plan CAD responsibilities for chassis and rocker-bogie - Contact UCF for Martian soil - Begin CAD for chassis
<p>Week 8:</p> <p>2/24/25 - 3/3/25</p>	<ul style="list-style-type: none"> - Discuss Martian soil acquisition with Dr. Britt - Confirm testing plan at UCF Exolith Lab - Resize main frame CAD and refine block diagrams
<p>Week 9:</p> <p>3/3/25 - 3/10/25</p>	<ul style="list-style-type: none"> - Confirm contact with Parks Easter (via Dr. Britt) - Assign design responsibilities - Continue drafting Chapters 1 & 2 - Refine system flowcharts
<p>Week 10:</p> <p>3/10/25 - 3/17/25</p>	<ul style="list-style-type: none"> - Focus on steady progress before spring break - Continue CAD development (arm, brush, frame) - Maintain consistent communication during the break
<p>Week 11:</p> <p>3/17/25 - 3/24/25</p>	<ul style="list-style-type: none"> - Maintain project momentum during break - Continue CAD work on main assemblies - Draft Chapters 3 & 4 of report - Modify and finalize flowcharts
<p>Week 12:</p> <p>3/24/25 - 3/31/25</p>	<ul style="list-style-type: none"> - Register for Lunabotics and confirm volunteer status - Finalize arm and drive system CAD - Begin brush mechanism CAD - Finalize report details and website review
<p>Week 13:</p> <p>3/31/25 - 4/7/25</p>	<ul style="list-style-type: none"> - Review revised flowcharts and presentation slides - Finalize formatting of report and website - Continue printing chassis components for final assembly
<p>Week 14:</p> <p>4/07/25 - 4/14/25</p>	<ul style="list-style-type: none"> - Practice and finalize report presentation - Assemble printed chassis components

The Corresponding Gantt chart on the next page is for spring displayed in Table 15. The expected design-phase Gantt chart created prior to the start of implementation is shown in Table 16 and served as a planning reference to guide weekly goals and deliverables following Senior Proposal.

Table 15. Spring Gantt Chart (Proposal Phase)

Senior Proposal Timeline Spring 2025																
Month	January				February				March				April			
Week	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15	Week 16
Task/Date	9-Jan	15-Jan	22-Jan	29-Jan	5-Feb	12-Feb	19-Feb	26-Feb	4-Mar	11-Mar	18-Mar	25-Mar	1-Apr	8-Apr	04/11 (Due Date)	04/18 (Presentation)
Define Project Goals & Scope																
Research Components & Feasibility																
Assign Team Roles & Responsibilities																
Choose Core Components																
Create System Block Diagram																
Test Motor & Servo Setup																
Draft Proposal Outline																
Prototype Small Drive System																
Start AI Camera Testing																
Refine Proposal & Get Feedback																
Develop Project Website																
Finalize Proposal Report																
Prepare Presentation																
Practice & Refine Presentation																

Table 16. Senior Design (Proposed) Gannt Chart

Senior Design Tentative Timeline Summer 2025 May 5th - July 28th												
Month	May				June				July			
Week	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12
Task/Date	05/05	5/12	5/19	5/26	06/02	6/9	6/16	6/23	6/30	07/07	7/14	7/21
Finish Assembling Astraeus												
Program Obstacle Avoidance												
Program AI Object Detection												
Program Path to Solar Panel												
Program Alignment Sequence												
Program Return to Base												
Create Database for logging												
Field Testing and Debugging												
Write Report												
Create Presentation												
Rehearse Presentation												

4.1.2 Design Phase (Summer 2025)

The Summer semester timeline is depicted in Table 17. The design semester focused on finalizing the software, assembling the finalized hardware, and troubleshooting. Table 18 depicts the timeline represented as Gantt chart. Following is the contributions table representing each group members' contribution in Table 19.

Table 17. Spring Timeline

Week	Objective
Week 15: 5/06/25 - 5/13/25	<ul style="list-style-type: none">- Begin final mechanical assembly- Review design-phase deliverables- Review roles and timeline ownership
Week 16: 5/13/25 - 5/20/25	<ul style="list-style-type: none">- Finalize mechanical structure- Power system wiring check- Start report Chapters 1–2
Week 17: 5/20/25 - 5/27/25	<ul style="list-style-type: none">- Begin obstacle avoidance programming- Verify Pi-to-driver communication- Continue documentation
Week 18: 5/27/25 - 6/3/25	<ul style="list-style-type: none">- Continue obstacle avoidance logic- Begin integrating AI object detection (HuskyLens)- Internal wiring validation
Week 19: 6/3/25 - 6/10/25	<ul style="list-style-type: none">- Refine object detection setup- Start path-to-panel logic- Continue report draft (Ch. 3–4)
Week 20: 6/10/25 - 6/17/25	<ul style="list-style-type: none">- Test AI object detection in real time- Tune path-planning algorithm- Flowchart and diagram updates
Week 21: 6/17/25 - 6/24/25	<ul style="list-style-type: none">- Begin alignment sequence programming- Start database setup for logging- Internal navigation test in mock setup

Week 22: 6/24/25 - 7/1/25	<ul style="list-style-type: none"> - Refine alignment and return-to-base logic - Field test partial navigation flow - Update testing procedures
Week 23: 7/1/25 - 7/8/25	<ul style="list-style-type: none"> - Repair and reinforce left-side rocker - Rewire rover for modularity with labeled connectors - Replaced Raspberry Pi and verified drivers - Order new motor drivers - Connect and configure Maestro controller - Finalize navigation/system flowcharts
Week 24: 3/10/25 - 3/17/25	<ul style="list-style-type: none"> - Complete system test of 5-axis arm - Confirm readiness for field testing - Finalize power budget with motor driver specs - Complete rough drafts of Chapters 1–5 - Refine navigation verification procedures
Week 25: 7/15/25 - 7/22/25	<ul style="list-style-type: none"> - Begin programming final autonomous logic - Start compiling results and debug logs - Continue field testing individual functions - Refine final report formatting
Week 26: 7/22/25 - 7/25/25	<ul style="list-style-type: none"> - Finalize presentation - Rehearse presentation

The weekly progress initially expected during the proposed design-phase timeline were closely followed with only minor setbacks, ensuring that major milestones such as system integration, field testing, and report completion were met on time. The Corresponding Gantt chart for the summer is displayed in Table 18.

Table 18. Spring Gantt Chart

Senior Design Timeline														
Month	May				June				July					
Week	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11		Week 12	
Task/Date	05/06	5/13	5/20	5/27	06/03	6/10	6/17	6/24	7/1	07/08	7/15	7/18	7/22	7/25
Finish Assembling Astraeus												Report & Website Due		Presentation Date
Program Obstacle Avoidance														
Program AI Object Detection														
Program Path to Solar Panel														
Program Alignment Sequence														
Program Return to Base														
Create Database for logging														
Field Testing and Debugging														
Write Report														
Create Presentation														
Rehearse Presentation														

The development of Project Astraeus required coordinated efforts across multiple technical domains, including mechanical design, electrical integration, and software development. To ensure efficiency and clear accountability, team responsibilities were divided based on each member's strengths and contributions. The table below outlines the specific roles and tasks completed by each team member throughout the course of the project, as well as tasks that were completed collaboratively. This distribution highlights the individual contributions that made the successful integration and functionality of the system possible.

Table 19. Contributions Table

Team Member	Responsibility
Mark	Design and implemented all electrical components and circuits
	Design CAD for main chassis to house components
	Design CAD for 5-axis cleaning arm
	3D-print and assemble the main frame and cleaning arm
	Finalize mechanical integration and fit-up
	Integrate electrical components into mechanical structure
	Code 5-axis cleaning arm movement sequences
Pedro	Design CAD for rocker-bogie suspension
	3D-print and assemble the components for rocker-bogie
	Code main control software and autonomous algorithm
	Create the Database and Web User Interface
	Integrate into Raspberry Pi, HuskyLens, and sensors
	Establish LAN communication for data transfer
Group	Write Report
	Refine System components
	Trouble-shoot system integrations
	Create and Practice Presentation

4.2 Budget

The financial plan for the Astraeus project includes two separate charts: one outlining the projected costs and another detailing the actual expenses tracked throughout the development cycle. This distinction provides transparency between the planned budget and the final comprehensive budget.

All purchases are categorized based on whether they were personally funded or provided by Valencia College, ensuring clear attribution of financial responsibility. Tables 20 and 21 present the proposed and final comprehensive budgets, respectively, and highlight where adjustments were made due to component replacements, testing needs, or integration challenges.

Table 20. Proposed Budget

ASTRAEUS Bill of Materials						
Line Item	Part #	Name	Qty	Price (USD)	Total	Supplier
1	N/A	V Slot Aluminum Extrusion	1	\$59.99	\$59.99	Purchased/Provided by Valencia College
2	N/A	OVERTURE PETG Filament 1.75mm	4	\$15.99	\$63.96	Purchased/Provided by Valencia College
3	N/A	HuskyLens AI Camera	1	\$59.90	\$59.90	Purchased/Provided by Valencia College
4	LM2596S	DC-DC Voltage Regulator Power Module	1	\$12.49	\$12.49	Purchased/Provided by Valencia College
5	12V30Ah	12V 30Ah Lithium LiFePO4 Battery	1	\$62.99	\$62.99	Purchased/Provided by Valencia College
6	N/A	Greartisan DC 12V 10RPM Gear Motor	6	\$14.99	\$89.94	Purchased/Provided by Valencia College
7	VNH5019	Motor Driver Carrier	3	\$29.95	\$89.85	Purchased/Provided by Valencia College
8	GP2Y0A21YK0F	Sharp IR Analog Distance Sensor (10-80cm)	4	\$8.59	\$34.36	Purchased/Provided by Valencia College
9	1576	99:1 Metal Gearmotor 25Dx54L mm HP 6V	2	\$28.95	\$57.90	Purchased/Provided by Valencia College

10	ADS1115	16-bit 4 Channel I2C ADC PGA Converter	1	\$15.99	\$15.99	Purchased
11	N/A	4 Channels I2C Logic Level Converter Bi-Directional 3.3V- 5V Shifter	1	\$7.49	\$7.49	Purchased
12	1352	Mini Maestro 12-Channel Servo Controller	1	\$32.95	\$32.95	Purchased/Provided by Valencia College
13	DS3225MG	25KG Full Metal Gear RC Servo (2 pack)	1	\$28.89	\$28.89	Purchased/Provided by Valencia College
14	DS3218MG	20KG Full Metal Gear RC Servo	1	\$13.59	\$13.59	Purchased/Provided by Valencia College
15	MG996R	55g Metal Gear Servo (4 pack)	1	\$16.68	\$16.68	Purchased/Provided by Valencia College
16	RPI3-MODB- 1GB	Raspberry Pi 3B	1	\$35.00	\$35.00	Purchased/Provided by Valencia College
18	N/A	4pcs 6mm Flange Coupling Connector	2	\$8.99	\$17.98	Purchased/Provided by Valencia College
19	N/A	1400 Pcs M4 Nuts Washers Kit	1	\$21.99	\$21.99	Purchased/Provided by Valencia College
20	N/A	100 Pieces 2020 Series M4 T Nuts	1	\$5.99	\$5.99	Purchased/Provided by Valencia College
21	N/A	600 Pcs M3 Screws Assortment Kit	1	\$8.99	\$8.99	Purchased/Provided by Valencia College
22	N/A	Rod End Bearings 5mm Female Thread	1	\$8.99	\$8.99	Purchased/Provided by Valencia College
23	B0DN6CY7JN	ESP32-Cam (2 Pack)	1	\$18.99	\$18.99	Purchased/Provided by Valencia College
24	N/A	M5 x 250mm Fully Threaded Rod 2Pcs	1	\$5.49	\$5.49	Purchased/Provided by Valencia College
Total (USD):					\$764.90	

Table 21. Final Comprehensive Budget

ASTRAEUS Bill of Materials						
Line Item	Part #	Name	Qty	Price (USD)	Total	Supplier
1	N/A	V Slot Aluminum Extrusion	1	\$59.99	\$59.99	Purchased/Provided by Valencia College
2	N/A	OVERTURE PETG Filament 1.75mm	8	\$15.99	\$127.92	Purchased/Provided by Valencia College
3	N/A	HuskyLens AI Camera	1	\$59.90	\$59.90	Purchased/Provided by Valencia College
4	B0B825HRB9	WWZMDiB Constant Current CC CV Buck Converter Module	2	\$9.99	\$19.98	Purchased/Provided by Valencia College
5	12V30Ah	12V 30Ah Lithium LiFePO4 Battery	1	\$62.99	\$62.99	Purchased/Provided by Valencia College
6	N/A	Greartisan DC 12V 10RPM Gear Motor	6	\$14.99	\$89.94	Purchased/Provided by Valencia College
7	VNH5019	Motor Driver Carrier	3	\$29.95	\$89.85	Purchased/Provided by Valencia College
8	GP2Y0A21YK0F	Sharp IR Analog Distance Sensor (10-80cm)	2	\$8.59	\$17.18	Purchased/Provided by Valencia College
9	N20-BT06	75:1 N20 Micro Gear Motor	2	\$6.15	\$12.30	Purchased/Provided by Valencia College
10	ADS1115	16-bit 4 Channel I2C ADC PGA Converter	1	\$15.99	\$15.99	Purchased by Pedro
11	N/A	4 Channels I2C Logic Level Converter Bi-Directional 3.3V- 5V Shifter	1	\$7.49	\$7.49	Purchased by Pedro
12	1352	Mini Maestro 12-Channel Servo Controller	1	\$32.95	\$32.95	Purchased/Provided by Valencia College
13	DS3225MG	25kg Full Metal Gear RC Servo (2 pack)	2	\$28.89	\$57.78	Purchased/Provided by Valencia College

14	DS3245SG	45kg Full Metal Gear RC Servo	2	\$31.99	\$63.98	Purchased/Provided by Valencia College
15	MG996R	55g Metal Gear Servo (4 pack)	1	\$16.68	\$16.68	Purchased/Provided by Valencia College
16	RPI3-MODB-1GB	Raspberry Pi 3 Model B	1	\$49.79	\$49.79	Purchased/Provided by Valencia College
18	N/A	4pcs 6mm Flange Coupling Connector	4	\$8.99	\$35.96	Purchased/Provided by Valencia College
19	N/A	1400 Pcs M4 Nuts Washers Kit	2	\$21.99	\$43.98	Purchased/Provided by Valencia College
20	N/A	100 Pieces 2020 Series M4 T Nuts	1	\$5.99	\$5.99	Purchased/Provided by Valencia College
21	N/A	600 Pcs M3 Screws Assortment Kit	1	\$8.99	\$8.99	Purchased/Provided by Valencia College
22	N/A	Rod End Bearings 5mm FemaleThread	1	\$8.99	\$8.99	Purchased/Provided by Valencia College
23	B0CDLKX842	14 AWG, 2-conductor CCA wire (Red & Black) 100 ft	1	\$20.99	\$20.99	Purchase by Mark
24	B0BKGZRM8C	18 AWG, 41-strand, low-voltage wire 100 ft	1	\$25.49	\$25.49	Purchased by Mark
25	Frienda-67892	Frienda Servo Extension Cables	2	\$8.99	\$17.98	Purchased by Mark
26	GUB-6-20	GUBCUB Terminal Block Kit (6 Circuit, Duel-Row, 20A-30A, 200V-450V)	4	\$10.49	\$41.96	Purchased by Mark
27		282 Pieces Car Fuses Assortment Kit	1	\$13.99	\$13.99	
28	N/A	M5 x 250mm Fully Threaded Rod 2Pcs	1	\$5.49	\$5.49	Purchased/Provided by Valencia College
Total (USD):					\$1,014.52	

While assembling and testing Astraeus, we encountered an issue with the original motor drivers. A wiring mistake during early integration caused the onboard MOSFETs to fail, requiring the repurchase

of replacement VNH5019 motor driver units. This unexpected event was the most significant deviation from the proposed budget. It also highlighted the importance of modular wiring and power isolation in future designs. The finalized comprehensive budget in Table 6 reflects all components required for a fully functional rover assuming no further hardware failures occur. If additional components were to fail during extended testing, the total cost could increase marginally beyond the current estimate.

4.3 Environmental Aspects

We minimized the environmental footprint of the Astraeus project by selecting low-power, energy-efficient components and using modular, repairable designs to reduce waste. Materials used in the construction of the rover, such as PETG for 3D-printed parts, are recyclable and were used with minimal waste through optimized slicing and support reuse.

All electronic components, including batteries and any damaged parts, were disposed of following Valencia College's electronic waste recycling procedures to prevent environmental contamination [16]. Additionally, the use of a solar panel for demonstration supports the project's alignment with sustainable energy systems and environmentally conscious design [17].

4.4 Health and Safety Considerations

We ensured that all electrical components and connections in Astraeus were securely enclosed or routed through protected cable paths to prevent accidental contact and electrical hazards. Wires were insulated and labeled, and fuses were used on the power supply lines to safeguard the system from short circuits or component failure. Each subsystem, including the motor drivers and servo controller, was mounted with sufficient clearance to prevent arcing and overheating.

To minimize mechanical risk, the rover's moving components, especially the rocker-bogie suspension, robotic arm, and cleaning brush were evaluated for pinch points and collision hazards. The robotic arm's range of motion and torque were analyzed to ensure that no crushing forces exceed acceptable safety thresholds during operation. All servos used for the arm are programmed to operate within safe speed and torque limits. The arm uses spring assistance to reduce motor strain and prevent sudden, high-force movements [18].

The cleaning mechanism, a rotating brush driven by a low-RPM motor, was tested to ensure that it would not produce hazardous contact force. Based on test observations, the system does not generate the threshold force of 150 N identified by OSHA and ISO standards as the limit for crush injuries. Similarly, the brush motor's output falls below 80 N, which is the injury threshold commonly associated with class 1 finger injuries such as ring avulsion [19].

To avoid injury during field testing, team members wore gloves, safety glasses, and closed-toe shoes, particularly when handling Martian regolith simulant or operating the rover on uneven terrain. Additionally, the rover's AI vision and obstacle avoidance features help prevent unintended collisions during autonomous operation. By integrating mechanical safeguards, electrical protection, and responsible operating procedures, the Astraeus project was carried out with a strong commitment to health and safety at all stages of design, testing, and demonstration.

4.5 Ethical Aspects

Project Astraeus strictly adheres to the IEEE Code of Ethics throughout the development process by ensuring that the rover operates reliably, safely, and without posing harm to users, observers, or the testing environment [20]. We prioritized safety in both hardware design and testing protocols, implementing safeguards such as modular power wiring, obstacle detection, and low-force mechanisms.

All team members maintained complete transparency in all performance reports, documentation, and presentations, clearly stating any limitations or challenges encountered during the project. All budget deviations, timeline adjustments, and hardware design changes were documented honestly, reflecting realistic expectations based on available resources and field test data.

The team respected intellectual property and credited all external contributions appropriately, including academic references, vendor support, and faculty guidance. We actively sought, accepted, and applied honest criticism throughout the proposal, design, and testing phases. Revisions to the arm system, frame structure, and electrical layout were driven by both advisor input and peer feedback.

Members of the Astraeus team worked collaboratively, treating one another fairly and professionally. At no point did the design or implementation of Astraeus violate the IEEE ethical commitment to avoid injury to others or their property. The project was conducted with a shared goal of promoting responsible engineering that aligns with real-world standards and classroom integrity. See Appendix G for a full list of specified IEEE ethical clauses followed throughout the project.

4.6 Sustainability Considerations

Sustainability was an important focus during the design and implementation of Astraeus, particularly regarding energy efficiency, modularity, and responsible component usage. The full system draws approximately 231.15 watts during peak operation, including the control platform, motors, robotic arm, and vision sensors. Power is supplied by a 12V 30Ah LiFePO₄ deep-cycle battery rated at 360 watt-hours, which offers high energy density, long rechargeability, and a lower environmental impact compared to traditional battery chemistries.

The use of rechargeable batteries significantly reduces electronic waste and environmental contamination. Components such as the Raspberry Pi 3 Model B, analog IR sensors, and micro gear motors were selected based on their energy efficiency relative to performance requirements. The robotic arm was carefully designed using servo motors that provide the necessary torque without drawing excess current. Combined with modular wiring and distributed fusing, the power system promotes safe, efficient energy use and simplifies repairs.

All motion and control algorithms were programmed to minimize unnecessary activity. Obstacle avoidance, solar panel alignment, and cleaning sequences were optimized to reduce energy consumption during autonomous operation. The brush system activates only when proper alignment has been achieved, which conserves energy and limits wear on components.

The modular mechanical design of Astraeus supports long-term sustainability. Each major subsystem, including the drive assembly, robotic arm, vision module, and power wiring, is designed for individual serviceability. This enables future teams to reuse or replace components without discarding the entire system. Additionally, the documentation and control structure created for Astraeus can serve as a foundation for future educational projects, further extending the utility and life span of the system.

Chapter 5

Conclusion

Summary

This chapter provides a summary of the Astraeus project, highlighting the design objectives, development process, and key testing results. It evaluates the performance of major subsystems, including the robotic arm and cleaning mechanism, and discusses how the final prototype met its engineering requirements. The chapter concludes with observations on project limitations and recommendations for future improvements.

- 5.1 Summary and Conclusion
- 5.2 Suggestions for future work

5.1 Summary and Conclusion

The Astraeus prototype was developed to demonstrate the feasibility of a fully autonomous solar panel cleaning rover capable of supporting long-duration missions in extraterrestrial environments. Designed around a modular aluminum chassis, Astraeus integrated a rocker-bogie suspension for terrain navigation, a five-axis robotic arm equipped with a dual-rotation brush for cleaning, and a vision-based targeting system using the HuskyLens AI camera. Despite using non-space-grade components, the rover was engineered to simulate real-world functionality through subsystem integration, preprogrammed motion sequences, and internal data logging.

Throughout its entire development, key engineering considerations were addressed, including structural adaptability, autonomous task execution, and system safety. Safety was addressed through passive design strategies, including low-speed actuation and robust mechanical stops to prevent overextension. In addition, sustainability and ethical engineering practices guided design decisions to ensure responsible use of materials and energy. The successful operation of the cleaning mechanism, combined with consistent navigation and performance logging, confirms Astraeus as a viable proof of concept for future off-world maintenance platforms. This chapter provides a comprehensive summary of the project's outcomes, engineering achievements, and areas for future improvement.

5.1.2 Robotic Arm Performance and Brush Results

The robotic arm met its high-level engineering requirement of operating a variable speed cleaning brush with sufficient control to effectively remove dust and debris from solar panel surfaces. The dual-rotation brush was able to maintain a consistent speed throughout testing. Although the system did not include real-time pressure feedback, the combination of servo speed tuning and joint positioning allowed for stable brush contact with the panel surface. Under controlled conditions using measured quantities of Martian regolith simulant, the brush demonstrated reliable dust agitation and removal. The testing confirmed that cleaning was consistent at the defined operating speed, fulfilling the requirement to verify performance through repeatable results.

Figures 48 and 49 present before-and-after comparison of the solar panel surface following the execution of a single brushing routine using Sequence 1 on both sides of the panel. In the initial state, a layer of fine particulate dust covered the panel, simulating Martian regolith

accumulation. After performing the full cleaning sequence, the robotic arm successfully agitated and swept the dust off the panel's surface, with visible improvements in cleanliness both times.

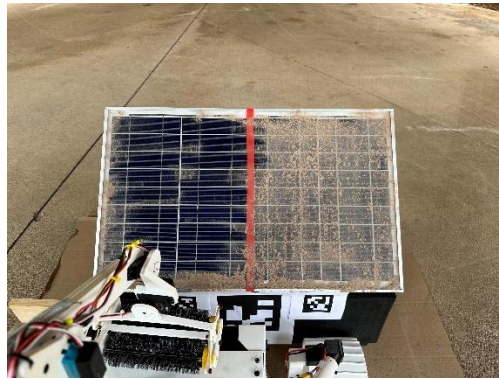


Figure 82. Panel Section 1 Cleaned

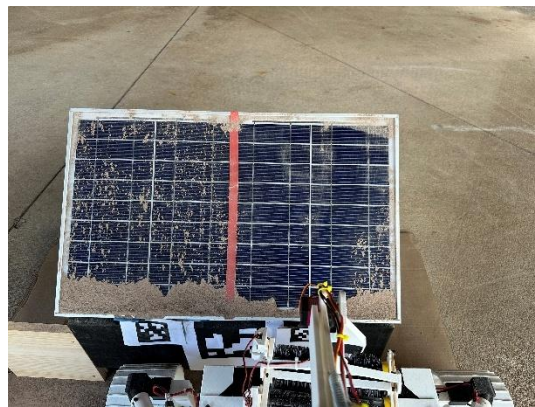


Figure 83. Panel Section 2 Cleaned

Sequence 1.1 was developed as an experimental routine to simulate a zigzag brushing pattern from the top to the bottom of the solar panel. This pattern was designed to provide broader coverage through directional shifting after each pass. Although implemented and included in the Maestro configuration file, it was not used in final testing due to mechanical limitations. The horizontal orientation of the brush in this sequence proved less effective, resulting in inconsistent bristle contact and reduced cleaning efficiency during lateral movements.

This outcome revealed a constraint in the current five-axis arm configuration: the inability to rotate the brush along its own axis. Without axial rotation, the brush cannot maintain perpendicular contact when changing sweep direction. For future improvements, a sixth axis of

motion is recommended to allow the brush to rotate 90 degrees, enabling it to align properly with the panel surface throughout both vertical and horizontal passes. This would improve contact pressure, coverage consistency, and overall cleaning performance. Even though the rover has no dynamic alignment or real-time correction, the robotic arm's brushing sequence proved capable of effectively agitating and displacing dust in a single pass. Dust was observed to lift from the surface and be swept downward off the panel's edge, validating the system's mechanical design and demonstrating the functionality of its cleaning mechanism under static control.

The medium-level engineering requirement of documenting the cleaning process was also satisfied. Each cleaning cycle was automatically logged to the centralized database hub for *Astraeus*, allowing detailed tracking of when and how often brushing routines were executed. This logging system captured sequence IDs, timestamps, and operation status flags, providing a structured record for post-test review. In addition, before-and-after photographs were taken for each panel cleaned, enabling visual analysis of dust and dirt removal. Together, these data sources offered a complete picture of cleaning performance and operational reliability during the trial period. To see the data log, refer to Figure 77 located back in chapter 3.

5.1.2 Autonomous Capabilities Results

The autonomous navigation system of *Astraeus* successfully met all high- and medium-level engineering requirements related to mobility, obstacle avoidance, target recognition, and return-to-base functionality. These capabilities were achieved through a combination of sensor-based logic, onboard April Tag visual markers, and internal data logging managed through the Raspberry Pi.

To initiate the cleaning sequence, *Astraeus* autonomously traveled toward the solar panel using April Tag 1 as the primary navigation reference. Upon entering visual range of AprilTag 2, the system transitioned to an alignment routine. *Astraeus* adjusted its heading in response to the marker's position and aligned itself with the panel to ensure proper brush positioning. This alignment was critical for the accuracy of the cleaning operation, and the rover was able to complete this routine without human assistance.

After cleaning, *Astraeus* successfully executed a reverse maneuver followed by a 180-degree turn. It then returned to the starting point using April Tag 3 as its final target. This

confirmed that the rover could return to its original base location autonomously, meeting the medium-level engineering requirement for return-to-base behavior.

Obstacle avoidance was also validated during testing. Astraeus was able to detect and avoid large objects placed within its path. The proximity sensors accurately identified obstacles within a range of 40 cm and halted the rover's motion to prevent collisions. This behavior was repeatable across multiple trials and demonstrated compliance with the high-level requirement for autonomous collision avoidance.

Throughout the process, key navigation events were recorded using a local server hosted by the onboard Raspberry Pi. Events such as travel initiation, obstacle detection, tag recognition, alignment, cleaning start, cleaning completion, and return-to-base were each logged with corresponding timestamps and location tags. A full demonstration of Astraeus performing autonomous navigation, alignment, obstacle avoidance, cleaning, and return-to-base is available at the following link: <https://youtube.com/shorts/Ls7SQvhasBQ>

During navigation testing, the Astraeus rover encountered some performance limitations with its vision system. As the rover increased in speed, the Huskylens struggled to process the rapidly changing visual field. The resulting motion blur made it difficult for the camera to maintain consistent recognition of April Tag markers. When an April Tag moved out of view, the system registered as lost and immediately triggered a safety protocol. This included logging the loss event and switching the rover into manual override mode to ensure it did not continue navigating without visual confirmation. This fail-safe behavior proved essential in preventing misalignment or unintended movement. The issue was traced to the limited resolution and frame rate of the HuskyLens AI camera. While higher-end cameras capable of real-time motion tracking were evaluated, their cost exceeded five hundred dollars and placed them outside the project's budget. This limitation presents a clear opportunity for future system upgrades in next-generation designs.

5.2 Suggestions for Future Work

While the current iteration of the Astraeus rover effectively demonstrates the ability to clean solar panels in Martian conditions, several enhancements could be made to improve its autonomy, efficiency, and adaptability. Future iterations of this project could integrate more

advanced systems to optimize decision-making, power management, and operational longevity. The following improvements outline key areas for development.

5.2.1 Solar Panel Efficiency Monitoring

One of the main challenges in maintaining solar panels on Mars is determining when cleaning is necessary. Instead of operating on a fixed schedule, the rover could benefit from a system that actively measures the efficiency of each solar panel in real-time. This could be achieved by monitoring power output and comparing it to expected performance under current environmental conditions. A built-in algorithm could detect a drop in efficiency caused by dust accumulation and trigger a cleaning cycle only when needed.

This approach would not only optimize power generation but also reduce unnecessary cleaning cycles, minimizing mechanical wear on the brush system and conserving the rover's energy. Sensors such as pyranometers (to measure solar irradiance) or voltage and current sensors could be integrated to provide real-time efficiency data. This would allow the rover to prioritize cleaning based on the most affected panels rather than performing routine cleaning on all panels equally.

5.2.2 Self-Docking Charging Station

Currently, the rover operates on battery power, requiring periodic manual recharging. Implementing a self-docking charging station would significantly extend its operational time and autonomy by allowing the rover to recharge without human intervention. By incorporating an autonomous docking system, the rover could return to a designated charging station when battery levels reach a critical threshold.

The docking system could include an alignment mechanism using Infrared Proximity Sensor to help the rover precisely position itself with charging connectors. Magnetic contacts or inductive charging could be explored to enable a seamless energy transfer process. This feature would be particularly valuable for long-duration missions where manual recharging is impractical. Additionally, an automated docking system could facilitate remote updates, diagnostics, and system recalibration without requiring direct access to the rover, improving long-term reliability and efficiency.

References

- [1] Yvonne K. McKenna, “Humans to Mars,” NASA, Online. Available at: <https://www.nasa.gov/humans-in-space/humans-to-mars>.
- [2] NASA, "Mars Exploration Rover (MER) - Opportunity," NASA Science, <https://science.nasa.gov/mission/mer-opportunity/> (accessed Jul. 15, 2025).
- [3] NASA, *Multi-Mission Radioisotope Thermoelectric Generator (MMRTG)*, NASA Facts, NF-2020-05-619-HQ, May 2020. [Online]. Available: <https://rps.nasa.gov>
- [4] SolarCleano, "Solar Panel Cleaning Robot – F1," *SolarCleano*, <https://solarcleano.com/product/solar-panel-cleaning-robot-f1> (accessed Jul. 15, 2025).
- [5] A. Verma, C. Yadav, B. Singh, A. Gupta, J. Mishra, and A. Saxena, “Design of Rocker-Bogie Mechanism,” *Int. J. Innov. Sci. Res. Technol.*, vol. 2, no. 5, pp. 312–316, May 2017.
- [6] B. D. Harrington and C. Voorhees, “The challenges of designing the rocker-bogie suspension for the Mars Exploration Rover,” in *Proc. 37th Aerospace Mechanisms Symp.*, Johnson Space Center, Houston, TX, USA, May 2004, pp. 185–196.
- [7] DFRobot, Metal DC Geared Motor - 12V 100RPM 42kg.cm, SKU: FIT0492-B, Apr. 2017. [Online]. Available: <https://www.dfrobot.com/product-1472.html>
- [8] STMicroelectronics, VNH5019A-E: Automotive Fully Integrated H-Bridge Motor Driver Datasheet, Rev. 11, Jun. 2017. [Online]. Available: <https://www.st.com>
- [9] Pololu, Micro Metal Gearmotor (N20) Series – Pololu 75:1 Gear Ratio, 12V Motor, Rev. 6.1, Pololu Corporation, 2023. [Online]. Available: <https://www.pololu.com/file/0J1928/pololu-micro-metal-gearmotors-rev-6-1.pdf>
- [10] Pololu Corporation, Pololu Maestro Servo Controller User’s Guide, 2022. [Online]. Available: <https://www.pololu.com/docs/0J40/all>

- [11] Solartech Power, “SPM030P-WP-F: 30 Watt Polycrystalline Solar Panel – Waterproof,” *Solarflexion*, [Online]. Available: https://www.solarflexion.com/v/vspfiles/files/pdfs/solartech-power/SPM030P-WP-F_Data_Sheet.pdf. [Accessed: July 15, 2025].
- [12] SHARP Corporation, “GP2D120 Distance Measuring Sensor,” Sharp Datasheet, 2006.
- [13] Texas Instruments, ADS111x Ultra-Small, Low-Power, I2C-Compatible, 860SPS, 16-Bit ADCs with Internal Reference, Oscillator, and Programmable Comparator Datasheet (Rev. E), Dec. 2024. [Online]. Available: <https://www.ti.com/lit/pdf/SBAS444>
- [14] DFRobot, HuskyLens V1.0 AI Vision Sensor, SKU: SEN0305 / SEN0336. Accessed: Jul. 12, 2025. [Online]. Available: https://wiki.dfrobot.com/HUSKYLENS_V1.0_SKU_SEN0305_SEN0336
- [15] RS Components, Raspberry Pi 3 Model B Product Sheet, RS Components, Accessed: Jul. 12, 2025. [Online]. Available: www.rs-components.com/raspberrypi
- [16] Valencia College, “Waste & Recycling,” Sustainability at Valencia College, [Online]. Available: <https://valenciacollege.edu/about/sustainability/waste-recycling.php>. [Accessed: Jul. 12, 2025].
- [17] Y. Shen, “Electronic product regulations in the United States: An overview,” Compliance Gate. [Online]. Available: <https://www.compliancegate.com/electronic-product-regulations-united-states/>. [Accessed: Feb. 16, 2025].
- [18] D. Mewes, “Safeguarding crushing points by limitations of forces,” PubMed, 2003. [Online]. Available: <https://pubmed.ncbi.nlm.nih.gov/12820907/>. [Accessed: Feb. 15, 2025].
- [19] “Hand Injury Risk Assessment,” Occupational Safety and Health Administration (OSHA). [Online]. Available: <https://www.osha.gov/hand-injury-prevention>. [Accessed: Feb. 18, 2025].
- [20] IEEE, “IEEE Code of Ethics,” IEEE.org, 2024. [Online]. Available: <https://www.ieee.org/about/corporate/governance/p7-8.html>. [Accessed: Jul. 12, 2025].

APPENDICES

Appendix A – Email Correspondence

Appendix B – VNH5019 Datasheet

Appendix C – Maestro Servo Controller User Guild

Appendix D - Solartech SPM030P-WP-F data sheet

Appendix E – GP2D120 Datasheet

Appendix F – Raspberry Pi 3 Model B Datasheet

Appendix G – Maestro Code Script

Appendix H – Astraeus Source Code

Appendix I – IEEE Code of Ethics

Appendix A – Email Correspondence

In this appendix is the email sent from Dr. Andrea Boca about Solar Panel Positioning

From: Dr. Andreea Boca, Solar Array Specialist, NASA Jet Propulsion Laboratory
Sent: Thursday, May 8, 2025 – 1:03 PM
To: Mark Figueroa mfigueroa88@valenciacollege.edu
Cc: Pedro Cabrera pcabrera6@valenciacollege.edu; Dr. Debbie Hall dhall@valenciacollege.edu
Subject: Re: ASTRAEUS Project Overview (Mark & Pedro)

Hi Mark,

I think assuming that the solar panels you clean are mounted at a 45-degree angle relative to the ground is as good a starting point as any. Panels in a solar farm on Mars could end up being mounted in pretty much any orientation, from parallel to the ground all the way to vertical, plus there is also be the possibility of sun tracking which would translate into a slowly variable angle depending on time of day/sol. Perhaps a future design iteration of your robot could make the cleaning process adaptable, i.e. work fine regardless of the angle the panels are mounted at, if feasible. But for now there's nothing wrong with starting simple and planning to build up from there later on. Hope this helps,

Andreea.

Appendix B – VNH5019 Datasheet

In this appendix is the datasheet for operating the VNH5019 Motor Drivers.

Block diagram and pin description

VNH5019A-E

Figure 2. Configuration diagram (top view)

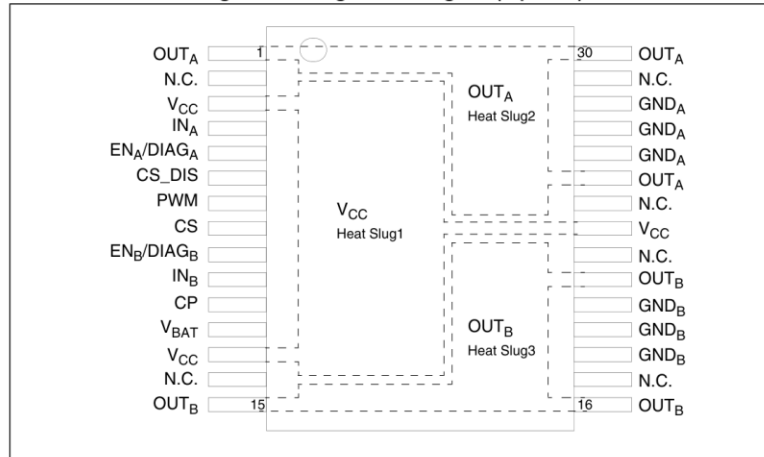


Table 1. Suggested connections for unused and non connected pins

Connection / pin	Current sense	N.C.	OUTx	INPUTx, PWM DIAGx/ENx CS_DIS
Floating	Not allowed	X	X	X
To ground	Through 1 kΩ resistor	X	Not allowed	Through 10 kΩ resistor

Table 2. Pin definitions and functions

Pin	Symbol	Function
1, 25, 30	OUT _A , Heat Slug2	Source of high-side switch A / drain of low-side switch A, power connection to the motor
2, 14, 17, 22, 24, 29	N.C.	Not connected
3, 13, 23	V _{CC} , Heat Slug1	Drain of high-side switches and connection to the drain of the external PowerMOS used for the reverse battery protection
12	V _{BAT}	Battery connection and connection to the source of the external PowerMOS used for the reverse battery protection
5	EN _A /DIAG _A	Status of high-side and low-side switches A; open drain output. This pin must be connected to an external pull-up resistor. When externally pulled low, it disables half-bridge A. In case of fault detection (thermal shutdown of a high-side FET or excessive ON-state voltage drop across a low-side FET), this pin is pulled low by the device (see Table 13: Truth table in fault conditions (detected on OUT_A)).

Table 2. Pin definitions and functions (continued)

Pin	Symbol	Function
6	CS_DIS	Active high CMOS compatible pin to disable the current sense pin
4	IN _A	Clockwise input. CMOS compatible
7	PWM	PWM input. CMOS compatible.
8	CS	Output of current sense. This output delivers a current proportional to the motor current, if CS_DIS is low or left open. The information can be read back as an analog voltage across an external resistor.
9	EN _B /DIAG _B	Status of high-side and low-side switches B; Open drain output. This pin must be connected to an external pull up resistor. When externally pulled low, it disables half-bridge B. In case of fault detection (thermal shutdown of a high-side FET or excessive ON-state voltage drop across a low-side FET), this pin is pulled low by the device (see Table 13: Truth table in fault conditions (detected on OUTA)).
10	IN _B	Counter clockwise input. CMOS compatible
11	CP	Connection to the gate of the external MOS used for the reverse battery protection
15, 16, 21	OUT _B , Heat Slug3	Source of high-side switch B / drain of low-side switch B, power connection to the motor
26, 27, 28	GND _A	Source of low-side switch A and power ground ⁽¹⁾
18, 19, 20	GND _B	Source of low-side switch B and power ground ⁽¹⁾

1. GND_A and GND_B must be externally connected together

Table 3. Block descriptions⁽¹⁾

Name	Description
Logic control	Allows the turn-on and the turn-off of the high-side and the low-side switches according to the Table 12 .
Overvoltage + undervoltage	Shut down the device outside the range [4.5 V to 24 V] for the battery voltage.
High-side, low-side and clamp voltage	Protect the high-side and the low-side switches from the high-voltage on the battery line in all configuration for the motor.
High-side and low-side driver	Drive the gate of the concerned switch to allow a proper R _{DS(on)} for the leg of the bridge.
Linear current limiter	Limits the motor current, by reducing the high-side switch gate-source voltage when short-circuit to ground occurs.
High-side and low-side overtemperature protection	In case of short-circuit with the increase of the junction temperature, it shuts down the concerned driver to prevent its degradation and to protect the die.
Low-side overload detector	Detects when low-side current exceeds shutdown current and latches off the concerned low-side.

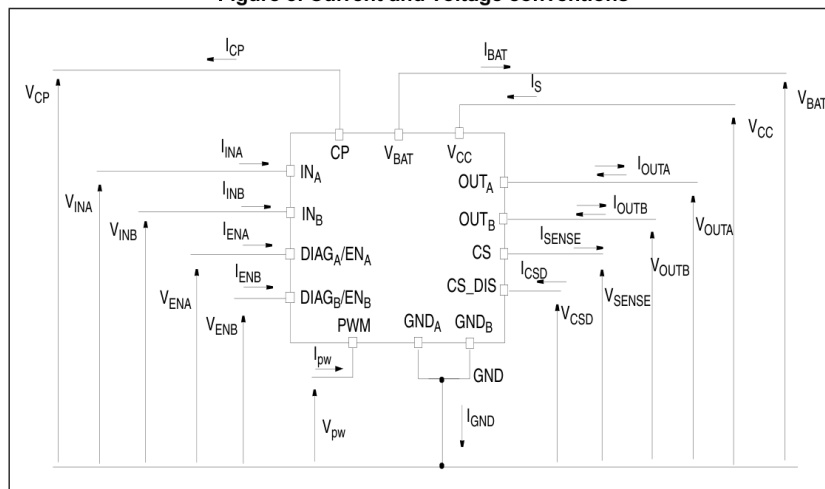
Table 3. Block descriptions⁽¹⁾ (continued)

Name	Description
Charge pump	Provides the voltage necessary to drive the gate of the external PowerMOS used for the reverse polarity protection
Fault detection	Signalizes an abnormal condition of the switch (output shorted to ground or output shorted to battery) by pulling down the concerned ENx/DIAGx pin.
Power limitation	Limits the power dissipation of the high-side driver inside safe range in case of short to ground condition.

1. See [Figure 1](#)

2 Electrical specifications

Figure 3. Current and voltage conventions



2.1 Absolute maximum ratings

Stressing the device above the rating listed in the “absolute maximum ratings” table may cause permanent damage to the device. These are stress ratings only and operation of the device at these or any other conditions above those indicated in the operating sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability. Refer also to the STMicroelectronics SURE program and other relevant quality document.

Table 4. Absolute maximum rating

Symbol	Parameter	Value	Unit
V _{BAT}	Maximum battery voltage ⁽¹⁾	-16	V
		+41	V
V _{CC}	Maximum bridge supply voltage	+ 41	V
I _{max}	Maximum output current (continuous)	30	A
I _R	Reverse output current (continuous)	-30	A
I _{IN}	Input current (IN _A and IN _B pins)	+/- 10	mA
I _{EN}	Enable input current (DIAG _A /EN _A and DIAG _B /EN _B pins)	+/- 10	mA
I _{pw}	PWM input current	+/- 10	mA
I _{CP}	CP output current	+/- 10	mA
I _{CS_DIS}	CS_DIS input current	+/- 10	mA

Table 4. Absolute maximum rating (continued)

Symbol	Parameter	Value	Unit
V_{CS}	Current sense maximum voltage	$V_{CC} - 41$ $+V_{CC}$	V V
V_{ESD}	Electrostatic discharge (human body model: $R = 1.5\text{ k}\Omega$, $C = 100\text{ pF}$)	2	kV
T_C	Case operating temperature	-40 to 150	°C
T_{STG}	Storage temperature	-55 to 150	°C

1. This applies with the n-channel MOSFET used for the reverse battery protection. Otherwise V_{BAT} has to be shorted to V_{CC} .

2.2 Thermal data

Table 5. Thermal data

Symbol	Parameter	Max. value	Unit
$R_{thj-case}$	Thermal resistance junction-case HSD	1.7	°C/W
	Thermal resistance junction-case LSD	3.2	°C/W
$R_{thj-amb}$	Thermal resistance junction-ambient	See Figure 18	°C/W

2.3 Electrical characteristics

Values specified in this section are for $8\text{ V} < V_{CC} < 21\text{ V}$, $-40\text{ °C} < T_J < 150\text{ °C}$, unless otherwise specified.

Table 6. Power section

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_{CC}	Operating bridge supply voltage		5.5		24	V
I_S	Supply current	OFF-state with all fault cleared and $EN_X = 0\text{ V}$ (standby): $IN_A = IN_B = PWM = 0$; $T_J = 25\text{ °C}$; $V_{CC} = 13\text{ V}$ $IN_A = IN_B = PWM = 0$		10	15 60	μA μA
		OFF-state (no standby): $IN_A = IN_B = PWM = 0$; $EN_X = 5\text{ V}$			6	mA
		ON-state: IN_A or $IN_B = 5\text{ V}$, no PWM IN_A or $IN_B = 5\text{ V}$, PWM = 20 kHz		4	8 8	mA mA
R_{ONHS}	Static high-side resistance	$I_{OUT} = 15\text{ A}$; $T_J = 25\text{ °C}$		12.0		m Ω
		$I_{OUT} = 15\text{ A}$; $T_J = -40\text{ °C}$ to 150 °C			26.5	
R_{ONLS}	Static low-side resistance	$I_{OUT} = 15\text{ A}$; $T_J = 25\text{ °C}$		6.0		m Ω
		$I_{OUT} = 15\text{ A}$; $T_J = -40\text{ °C}$ to 150 °C			11.5	
V_f	High-side free-wheeling diode forward voltage	$I_f = 15\text{ A}$, $T_J = 150\text{ °C}$		0.6	0.8	V
$I_{L(off)}$	High-side OFF-state output current (per channel)	$T_J = 25\text{ °C}$; $V_{OUTX} = EN_X = 0\text{ V}$; $V_{CC} = 13\text{ V}$			3	μA
		$T_J = 125\text{ °C}$; $V_{OUTX} = EN_X = 0\text{ V}$; $V_{CC} = 13\text{ V}$			5	

Table 7. Logic inputs (IN_A , IN_B , EN_A , EN_B , PWM, CS_DIS)

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_{IL}	Low-level input voltage	Normal operation ($DIAG_X/EN_X$ pin acts as an input pin)			0.9	V
V_{IH}	High-level input voltage	Normal operation ($DIAG_X/EN_X$ pin acts as an input pin)	2.1			V
I_{INL}	Low-level input current	$V_{IN} = 0.9\text{ V}$	1			μA
I_{INH}	High-level input current	$V_{IN} = 2.1\text{ V}$			10	μA
V_{HYST}	Input hysteresis voltage	Normal operation ($DIAG_X/EN_X$ pin acts as an input pin)	0.15			V

Table 7. Logic inputs (IN_A, IN_B, EN_A, EN_B, PWM, CS_DIS) (continued)

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V _{ICL}	Input clamp voltage	I _{IN} = 1 mA	5.5	6.3	7.5	V
		I _{IN} = -1 mA	-1.0	-0.7	-0.3	
V _{DIAG}	Enable low-level output voltage	Fault operation (DIAG _X /EN _X pin acts as an output pin); I _{EN} = 1 mA			0.4	V

Table 8. Switching (V_{CC} = 13 V, R_{LOAD} = 0.87 Ω, T_j = 25 °C)

Symbol	Parameter	Test conditions	Min	Typ	Max	Unit
f	PWM frequency		0		20	kHz
t _{d(on)}	HSD rise time	Input rise time < 1 μs (see Figure 9)			250	μs
t _{d(off)}	HSD fall time	Input rise time < 1 μs (see Figure 9)			250	μs
t _r	LSD rise time	(see Figure 8)		1	2	μs
t _f	LSD fall time	(see Figure 8)		1	2	μs
t _{DEL}	Delay time during change of operating mode	(see Figure 7)	200	400	1600	μs
t _{rr}	High-side free wheeling diode reverse recovery time	(see Figure 10)		110		ns
I _{RM}	Dynamic cross-conduction current	I _{OUT} = 15 A (see Figure 10)		2		A

Table 9. Protection and diagnostic

Symbol	Parameter	Test conditions	Min	Typ	Max	Unit
V _{USD}	V _{CC} undervoltage shutdown			4.5	5.5	V
V _{USDhyst}	V _{CC} undervoltage shutdown hysteresis			0.5		V
V _{OV}	V _{CC} overvoltage shutdown		24	27	30	V
I _{LIM_H}	High-side current limitation		30	50	70	A
I _{SD_LS}	Low-side shutdown current		70	115	160	A
V _{CLPHS} ⁽¹⁾	High-side clamp voltage (V _{CC} to OUT _A = 0 or OUT _B = 0)	I _{OUT} = 15 A	43	48	54	V
V _{CLPLS} ⁽¹⁾	Low-side clamp voltage (OUT _A = V _{CC} or OUT _B = V _{CC} to GND)	I _{OUT} = 15 A	27	30	33	V
T _{TSD} ⁽²⁾	Thermal shutdown temperature	V _{IN} = 2.1 V	150	175	200	°C

Appendix C – Maestro Servo Controller User Guild

In this appendix is the important documentation from the Maestro Servo Controller User Guild.

Servo power connections are provided in the upper right corner of the Micro Maestro board. Servo power is passed directly to the servos without going through a regulator, so the only restrictions on your servo power supply are that it must be within the operating range of your servos and provide enough current for your application. Please consult the datasheets for your servos to determine an appropriate servo power source, and note that a ballpark figure for the current draw of an average straining servo is 1 A.

You can power the Maestro's processor and servos from a single power supply by connecting the positive power line to both VIN and the servo power ports. An easy way to accomplish this on the Micro Maestro is to solder a wire on the bottom of the board between VIN and one of the servo power connections as shown in the picture to the right. Only one ground connection is needed because all ground pins on the board are connected.



Micro Maestro configured to use a single power supply for both board and servos.

The **5V (out)** power output allows you to power your own 5V devices from the on-board 50mA regulator or directly from USB. The on-board regulator is used whenever VIN is powered; in this case, since the Maestro requires 30 mA, there is about 20 mA available to power other devices.

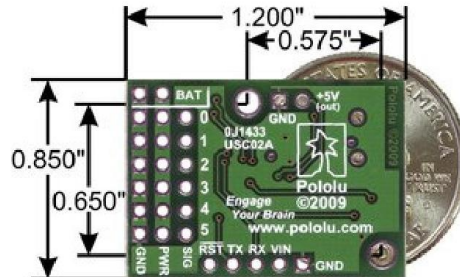
The **SIG** lines (**0**, **1**, **2**, ...) are used for sending pulses to servos, controlling digital outputs, and measuring analog voltages. These lines are protected by 220Ω resistors. The total current limit (in or out) for these pins is 60 mA, but when using the on-board regulator the current *out* is limited to 20 mA (see above.)

The **RX** line is used to receive non-inverted TTL (0–5 V) serial bytes, such as those from microcontroller UARTs. These bytes can either be serial commands for the Maestro, arbitrary bytes to send back to the computer via the USB connection, or both. For more information about the Maestro's serial interface, see **Section 5.a**. Note that the Maestro will probably be able to receive 3.3V TTL serial bytes, but it is not guaranteed to read 3.3V as high on the RX pin, so you should boost 3.3V TTL serial signals to above 4V if you want to ensure reliable operation.

The **TX** line transmits non-inverted TTL (0–5 V) serial bytes. These bytes can either be responses to serial commands sent to the Maestro, or arbitrary bytes sent from the computer via the USB connection.

The **RST** pin can be driven low to reset the Maestro's microcontroller, but this should not be necessary for typical applications. The line is internally pulled high, so it is safe to leave this pin unconnected. Driving **RST** low is roughly equivalent to powering off the Maestro; it will **not** reset any of the configuration parameters stored in non-volatile memory. To reset the configuration parameters, select

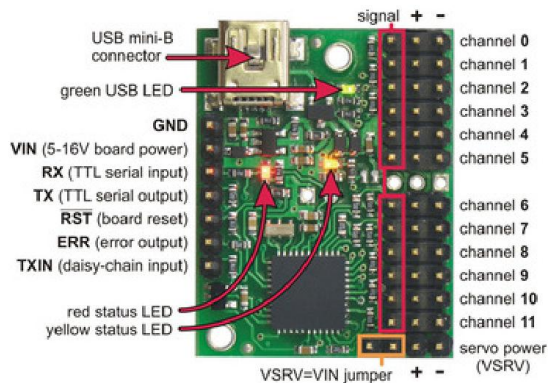
Device > Reset to default settings... in the Maestro Control Center.



Micro Maestro 6-channel USB servo controller bottom view with quarter for size reference.

The dimensions of the Micro Maestro PCB are 1.2" × 0.85". The mounting holes have a diameter of 0.086" and are intended for #2 or M2 screws. The vertical and horizontal distances between the two mounting holes are 0.65" and 0.575". The Micro Maestro weighs 3.0 g (0.11 oz) without header pins.

1.b. Mini Maestro Pinout and Components



Mini Maestro 12-channel USB servo controller (fully assembled) labeled top view.

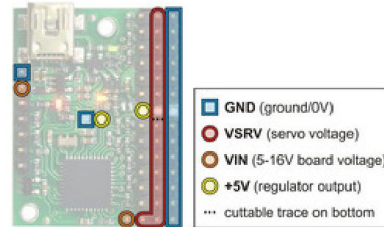
The processor and the servos can have separate power supplies.

Processor power must come either from USB or from an external 5–16V power supply connected to the **VIN** and **GND** inputs on the left side of the board. It is safe to have an external power supply connected at the same time that USB is connected; in that case the processor will be powered from the external supply. Note that if the external supply falls below 5 V, correct operation is not guaranteed, even if USB is also connected.

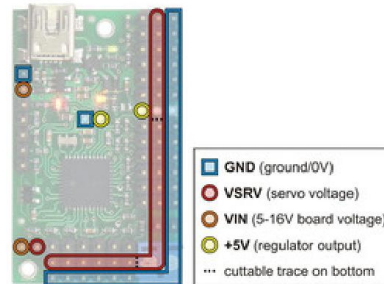
Servo power connections are provided in the lower right corner of the Mini Maestro board. On the Mini Maestro 18 and 24, you can make servo power connections via a 2-pin terminal block or a 2-pin 0.1" header; the Mini Maestro 12 only has a 2-pin 0.1" header for connecting servo power. Servo power is passed directly to the servos without going through a regulator, so the only restrictions on your servo power supply are that it must be within the operating range of your servos and provide enough current for your application. Please consult the datasheets for your servos to determine an appropriate servo power source, and note that a ballpark figure for the current draw of an average straining servo is 1 A.

You can power the Maestro's processor and servos from a single power supply by connecting the positive power line to both VIN and the servo power ports (only one ground connection is needed because all ground pins on the board are connected). The recommended way to do this is to connect your power supply to the dedicated servo power pins in the corner of the board and use the included blue shorting block to connect the pins labeled "VSRV=VIN".

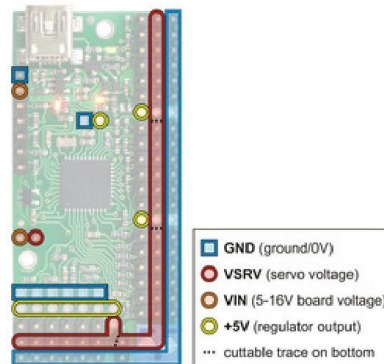
The **5V (out)** power output allows you to power your own 5V devices from the 100mA on-board regulator or directly from USB. The on-board regulator is used whenever VIN is powered; in this case, since the Maestro requires 50 mA, there is about 50 mA available to power other devices.



Mini Maestro 12 power pins.



Mini Maestro 18 power pins.



Mini Maestro 24 power pins.

The signal lines (0, 1, 2, ...) are used for sending pulses to servos, controlling digital outputs, and measuring voltages. The total current limit (in or out) for these pins is 200 mA, but when using the on-board regulator the current *out* is limited to 50 mA (see above.)

The **RX** line is used to receive non-inverted TTL (0–5 V) serial bytes, such as those from microcontroller UARTs. These bytes can either be serial commands for the Maestro, arbitrary bytes to send back to the computer via the USB connection, or both. For more information about the Maestro's serial interface, see **Section 5.a**. Note that the Maestro will probably be able to receive 3.3V TTL serial bytes, but it is not guaranteed to read 3.3V as high on the RX pin, so you should boost 3.3V TTL serial signals to above 4V if you want to ensure reliable operation.

The **TX** line transmits non-inverted TTL (0–5 V) serial bytes. These bytes are either generated by the Mini Maestro itself (as responses to serial commands or arbitrary bytes sent from the computer via the USB connection), or they come from the TXIN line.

The **RST** pin can be driven low to reset the Maestro's microcontroller, but this should not be necessary for typical applications. The line is internally pulled high, so it is safe to leave this pin unconnected. Driving **RST** low is roughly equivalent to powering off the Maestro; it will **not** reset any of the configuration parameters stored in non-volatile memory. To reset the configuration parameters, select **Device > Reset to default settings...** in the Maestro Control Center.

The **ERR** line is an output that is tied to the red error/user LED. It is driven high when the red LED is on, and it is pulled low through the red LED when the red LED is off. The red LED turns on when an error occurs, turns off when the error flags have been cleared, and can also be controlled by the user script. Since the ERR line is never driven low, it is safe to connect the ERR line of multiple Mini Maestros together. Please note, however, that doing this will cause the red LEDs of all connected Mini Maestros to turn on whenever one of the Mini Maestros turns on its red LED. For more information on the possible error conditions and response options, please see **Section 4.e**.

The **TXIN** line is a serial input line that makes it easy to chain together multiple Mini Maestros. Any serial bytes received on this line will be buffered through an AND gate and transmitted on the TX line. See **Section 5.g** for more information about daisy chaining.

3. Getting Started

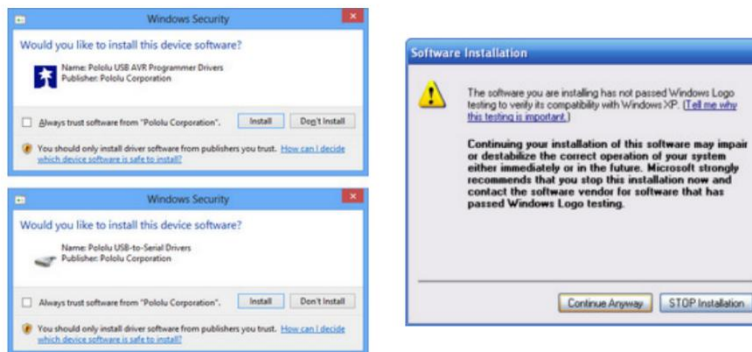
3.a. Installing Windows Drivers and Software



If you are using Windows XP, you will need to have **Service Pack 3** [https://en.wikipedia.org/wiki/Windows_XP#Service_Pack_3] installed before installing the drivers for the Maestro. See below for details.

Before you connect your Maestro to a computer running Microsoft Windows, you should install its drivers:

1. Download the **Maestro Servo Controller Windows Drivers and Software** [<https://www.pololu.com/file/0J266/maestro-windows-130422.zip>] (5MB zip)
2. Open the ZIP archive and run *setup.exe*. The installer will guide you through the steps required to install the Maestro Control Center, the Maestro command-line utility (UscCmd), and the Maestro drivers on your computer. If the installer fails, you might have to extract all the files to a temporary directory, right click *setup.exe*, and select "Run as administrator".
3. During the installation, Windows will ask you if you want to install the drivers. Click "Install" (Windows Vista, Windows 7, and later) or "Continue Anyway" (Windows XP).



4. After the installation is finished, your start menu should have a shortcut to the *Maestro Control Center* (in the *Pololu* folder). This is a Windows application that allows you to configure, control, debug, and get real-time feedback from the Maestro. There will also be a command-line utility called *UscCmd* which you can run at a Command Prompt.

Windows 10, Windows 8, Windows 7, and Windows Vista users: After following the steps above, you can connect a Maestro to your computer, and your computer should automatically install the

Appendix D – Solartech SPM030P-WP-F data sheet

In this appendix is the important documentation for the Solartech SPM030P-WP-F data sheet.

**SOLARTECH[®]**
POWER, INC.

W-Series
30W PV Module
SPM030P-WP-F

Solartech W-Series Modules

Solartech photovoltaic W-Series Modules are constructed with high efficient polycrystalline solar cells and produce higher output per module than others in it class. This industrial grade module is an industry standard among various industry professionals.

Features

Class 1, Division 2, (C1D2) Group A,B,C and D

- Accessible junction box with 4-1/2" knockout for ease of installation.
- (EVA) with TPT cushions the solar cells within the laminate an ensures the operating characteristics of the solar cells under virtually any climatic condition
- Rigid anodized aluminum frame and low iron tempered glass
- Easily accessible grounding points on all four corners for fast installation
- Proven junction box technology

Reliability

- Proven superior field performance
- Tight power tolerance

Qualifications and Certifications





Applications

- Traffic & Safety
- Federal Government
- Oil & Gas
- Security
- Telecommunications
- Water and Wastewater
- Weather & Environmental Monitoring
- RV Camper
- Emergency Power
- Telemetry
- SCADA, RTU, GPS
- Marine
- Area Lighting & Sign

Model Number SPM030P-WP-F

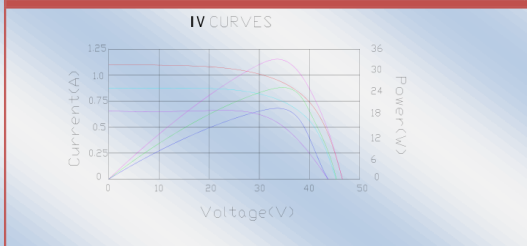
Electrical Characteristics

Max power(Pm)	30W
Maximum power voltage(Vpm)	33.8V
Maximum power current (Ipm)	0.89A
Short circuit current (Isc)	1.03A
Open circuit voltage (Voc)	41.2V
Module efficiency	10.9%
Tolerance	±5%
Nominal Voltage	24V
Temperature coefficient of Voc	-0.36%/K
Temperature coefficient of Pm	-0.46%/K
Temperature coefficient of Isc	0.05%/K
NOCT	48°C ± 2°C
Maximum series fuse rating	10A
Maximum system voltage	600V

Mechanical Characteristics

Construction	Tempered glass, silicon cell, EVA, Polyester with Tedlar
Solar Cells	68 cells (156mm x19.5mm) in a 4x17 matrix connected in series
Front Cover	High transmission 3.2mm(1/8") glass
Encapsulant	EVA(Double layers)
Back Cover	White polyester
Frame	Anodized aluminum
Junction Box	IP65, UL94-5VA material
Diodes	Schottky by-pass diodes
Terminal	Accept 8-14 AWG wire
Dimensions	26.2in(666mm)x16.2in(412mm)x1.38in(35mm)
Weight	7.7lb (3.5kg)
Operating Temperature	-40°C ~ 90°C
Storage Humidity	<90%

IV Curves

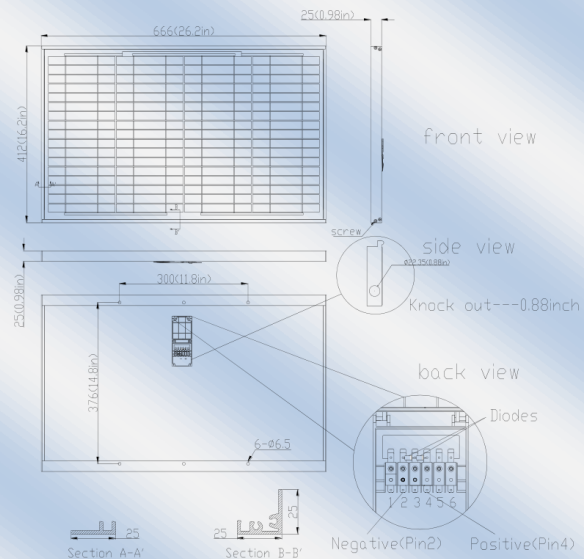


Warranty

25-year limited warranty of 80% power output;
12 year limited warranty of 90% power output;
2-year limited warranty of materials and workmanship*

Certifications

UL 1703 certification
ETL Class I ,Division 2,Groups A,B,C and D certification



Appendix E – GP2D120 Datasheet

In this appendix is the important documentation from the GP2D120 IR sensors.



GP2D120 Optoelectronic Device

FEATURES

- Analog output
- Effective range: 4 to 30 cm
- Typical response time: 39 ms
- Typical start up delay: 44 ms
- Average Current Consumption: 33 mA

DESCRIPTION

The GP2D120 is a distance measuring sensor with integrated signal processing and analog voltage output.

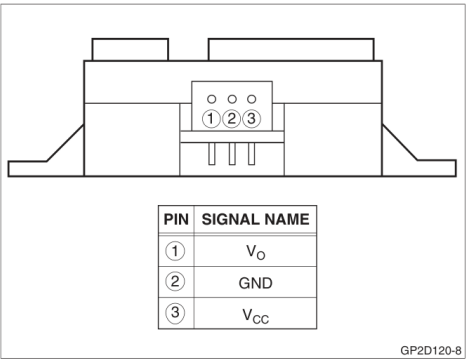


Figure 1. Pinout

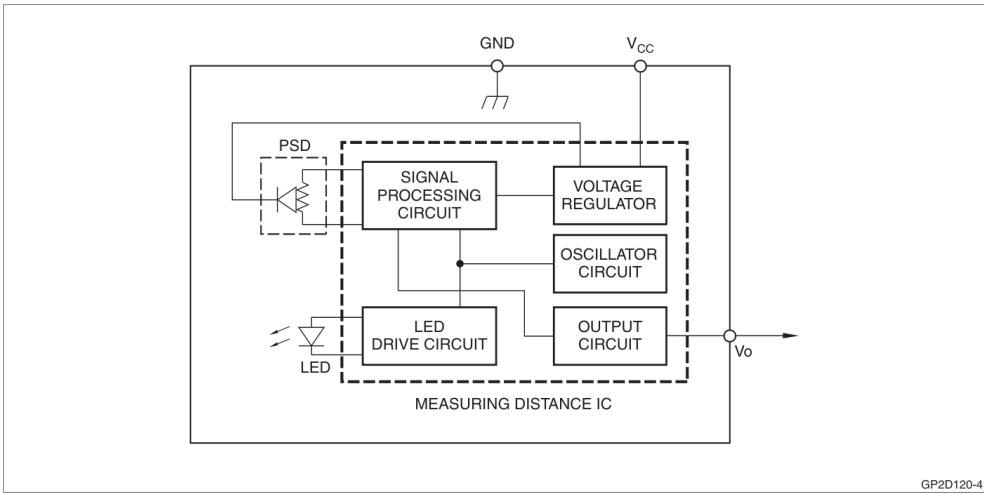


Figure 2. Block Diagram

ELECTRICAL SPECIFICATIONS

Absolute Maximum Ratings

$T_a = 25^\circ\text{C}$, $V_{CC} = 5\text{ VDC}$

PARAMETER	SYMBOL	RATING	UNIT
Supply Voltage	V_{CC}	-0.3 to +7	V
Output Terminal Voltage	V_O	-0.3 to ($V_{CC} + 0.3$)	V
Operating Temperature	T_{opr}	-10 to +60	$^\circ\text{C}$
Storage Temperature	T_{stg}	-40 to +70	$^\circ\text{C}$

Operating Supply Voltage

PARAMETER	SYMBOL	RATING	UNIT
Operating Supply Voltage	V_{CC}	4.5 to 5.5	V

Electro-optical Characteristics

$T_a = 25^\circ\text{C}$, $V_{CC} = 5\text{ VDC}$

PARAMETER	SYMBOL	CONDITIONS	MIN.	TYP.	MAX.	UNIT	NOTES
Measuring Distance Range	ΔL		4	—	30	cm	1, 2
Output Terminal Voltage	V_O	$L = 30\text{ cm}$	0.25	0.4	0.55	V	1, 2
Output Voltage Difference	ΔV_O	Output change at ΔL (30 cm – 4 cm)	1.95	2.25	2.55	V	1, 2
Average Supply Current	I_{CC}	$L = 30\text{ cm}$	—	33	50	mA	1, 2

NOTES:

- Measurements made with Kodak R-27 Gray Card, using the white side, (90% reflectivity).
- L = Distance to reflective object.

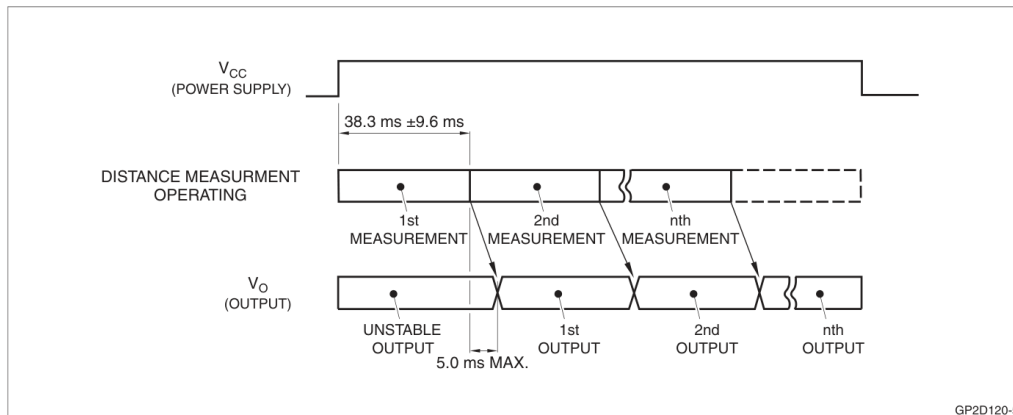


Figure 3. Timing Diagram

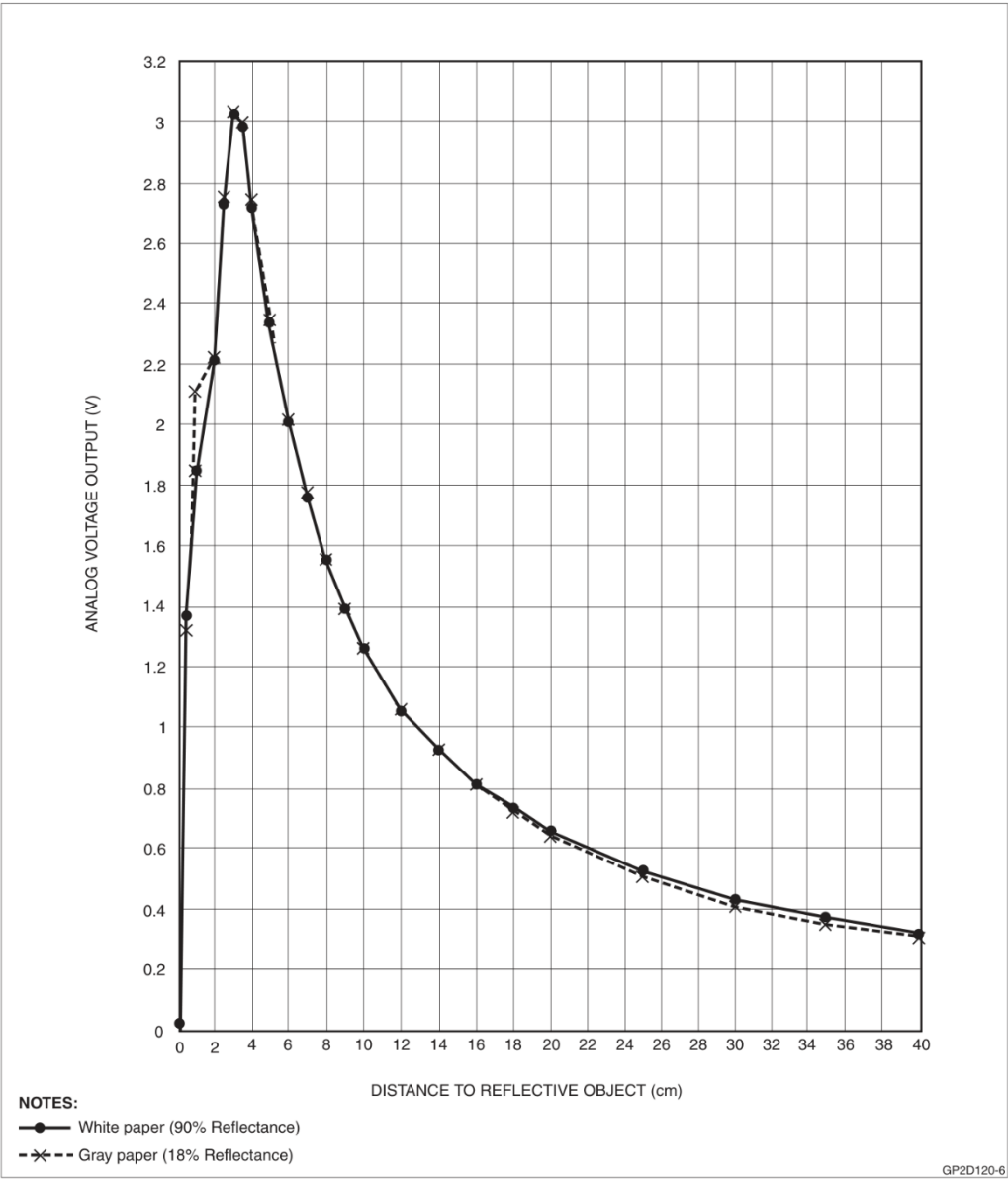


Figure 4. GP2D120 Example of Output Distance Characteristics

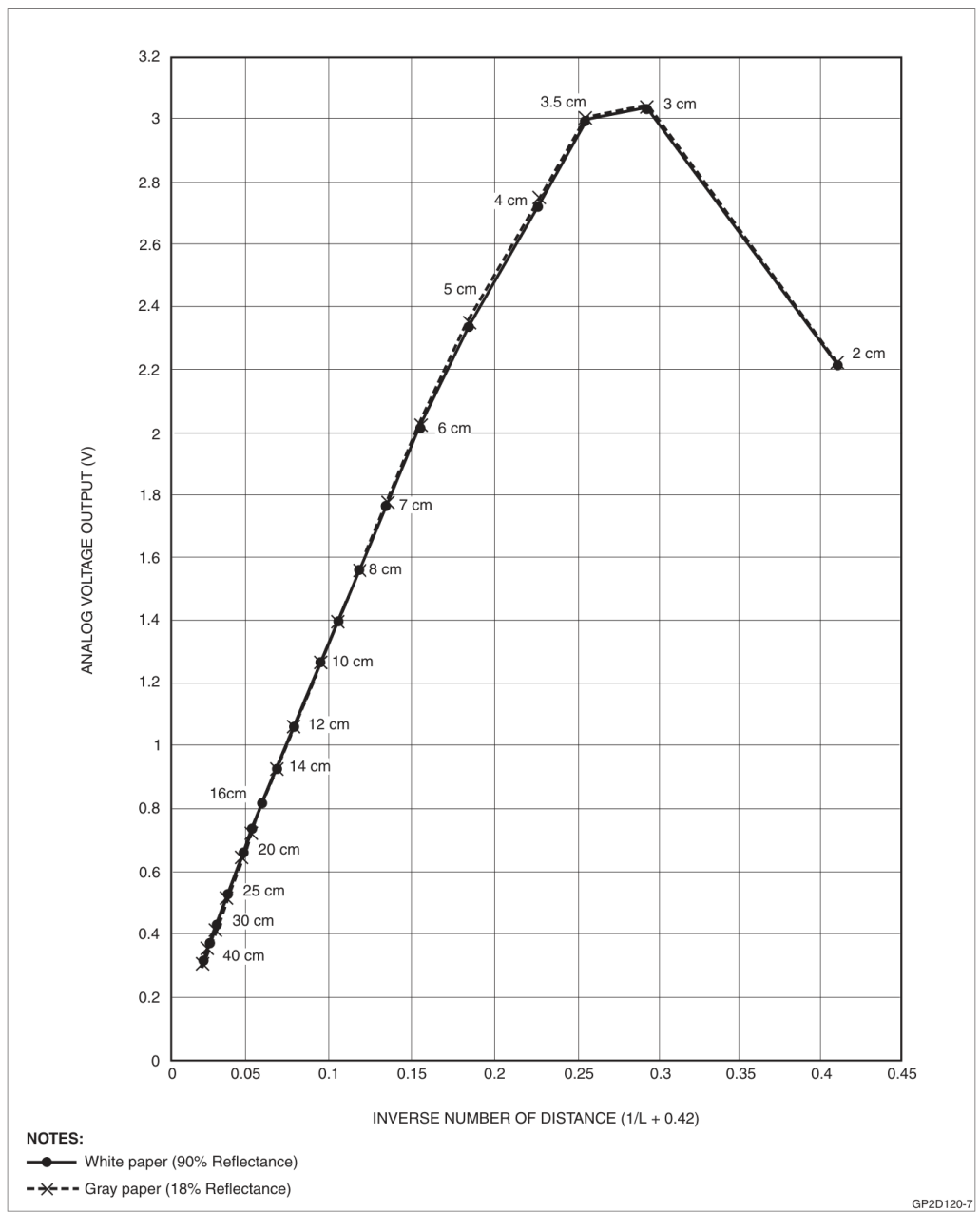
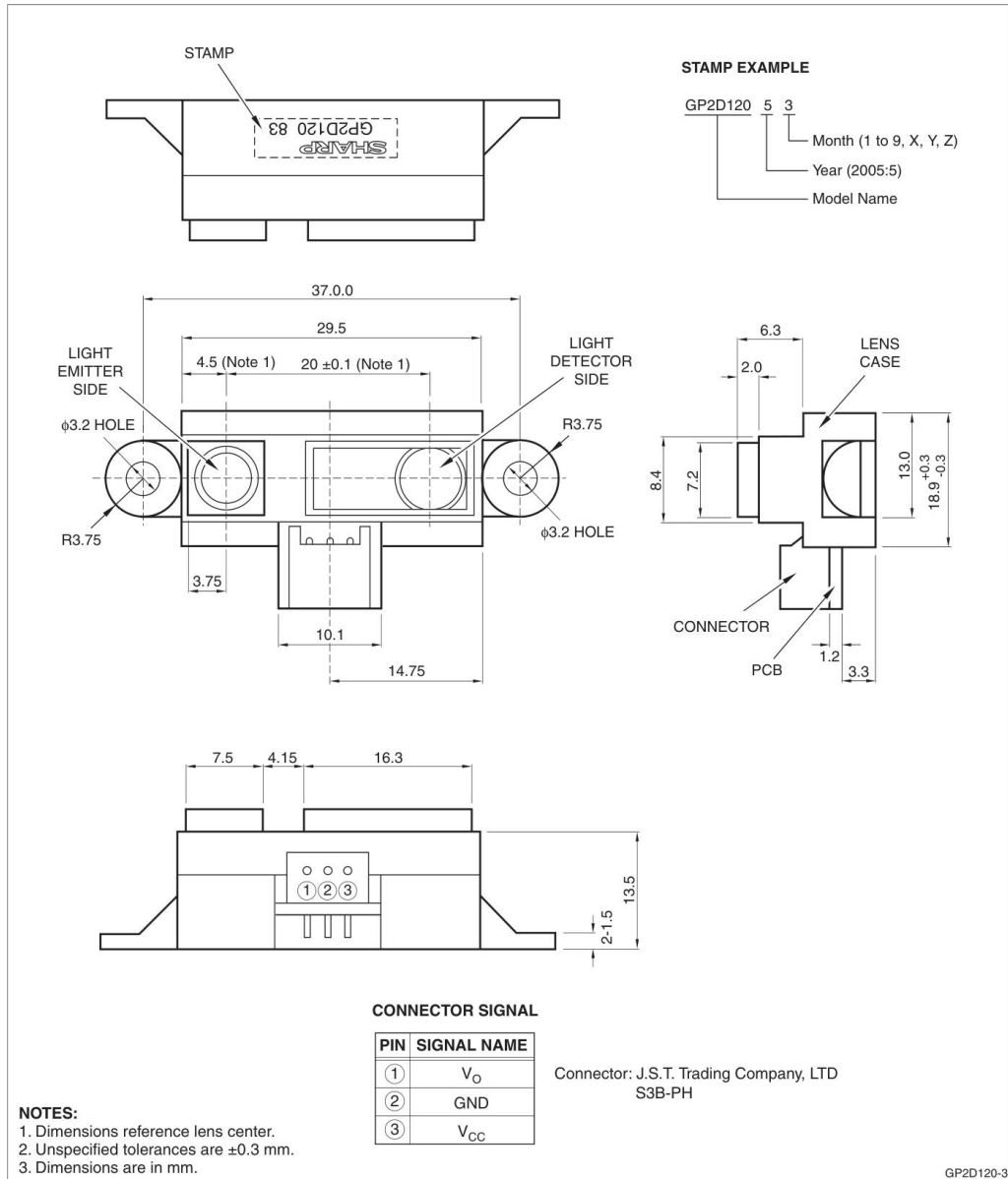


Figure 5. GP2D120 Example of Output Characteristics with Inverse Number of Distance

PACKAGE SPECIFICATIONS



Appendix F – Raspberry Pi 3 Model B

In this appendix is the important documentation from the Raspberry Pi 3 Model B datasheet.



Raspberry Pi

Raspberry Pi 3 Model B

Specifications

Processor	Broadcom BCM2387 chipset. 1.2GHz Quad-Core ARM Cortex-A53 802.11 b/g/n Wireless LAN and Bluetooth 4.1 (Bluetooth Classic and LE)
GPU	Dual Core VideoCore IV® Multimedia Co-Processor. Provides Open GL ES 2.0, hardware-accelerated OpenVG, and 1080p30 H.264 high-profile decode. Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure
Memory	1GB LPDDR2
Operating System	Boots from Micro SD card, running a version of the Linux operating system or Windows 10 IoT
Dimensions	85 x 56 x 17mm
Power	Micro USB socket 5V1, 2.5A

Connectors:

Ethernet	10/100 BaseT Ethernet socket
Video Output	HDMI (rev 1.3 & 1.4) Composite RCA (PAL and NTSC)
Audio Output	Audio Output 3.5mm jack, HDMI USB 4 x USB 2.0 Connector
GPIO Connector	40-pin 2.54 mm (100 mil) expansion header: 2x20 strip Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines
Camera Connector	15-pin MIPI Camera Serial Interface (CSI-2)
Display Connector	Display Serial Interface (DSI) 15 way flat flex cable connector with two data lanes and a clock lane
Memory Card Slot	Push/pull Micro SDIO

Key Benefits

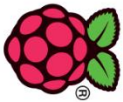
- Low cost
- 10x faster processing
- Consistent board format
- Added connectivity

Key Applications

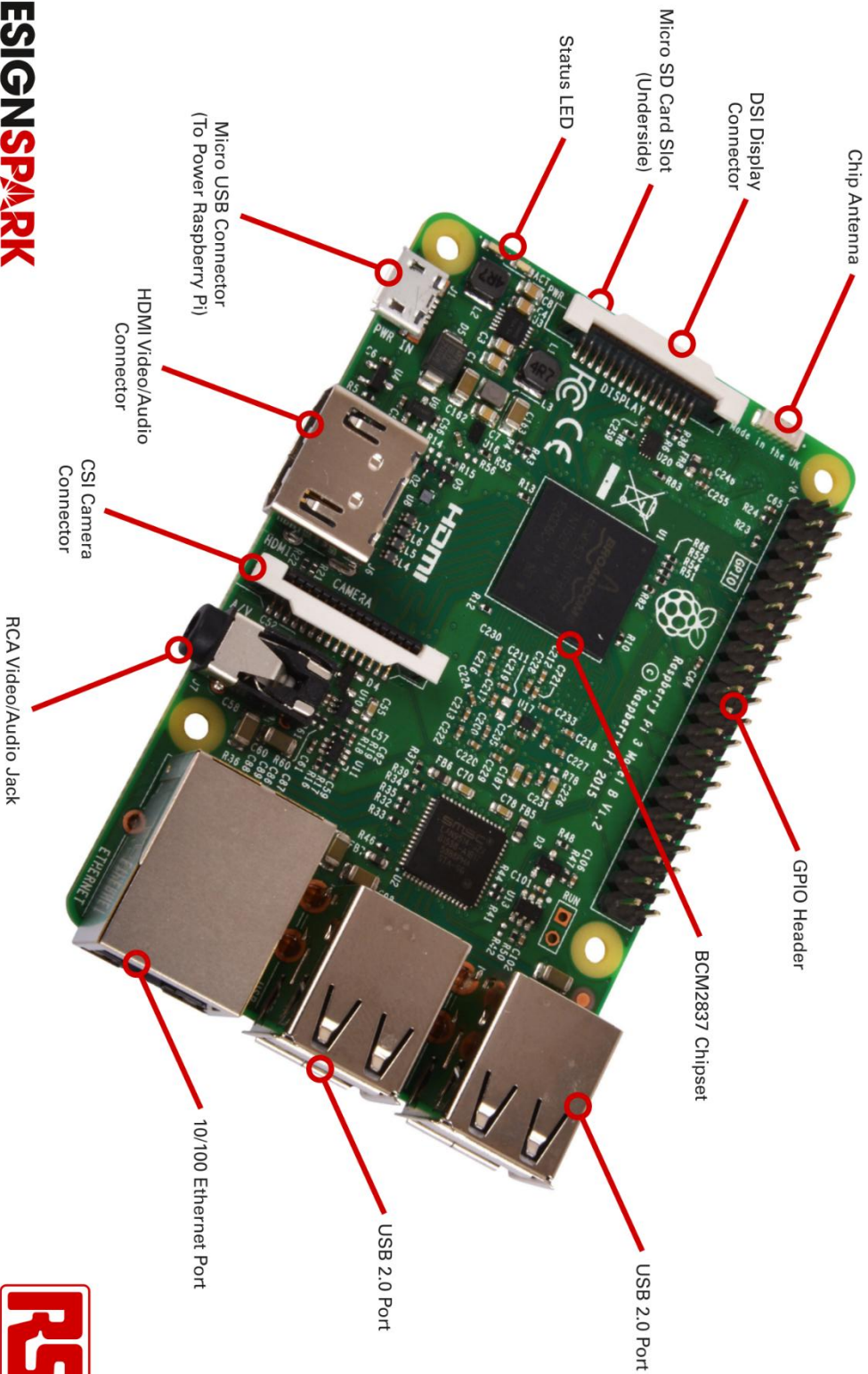
- Low cost PC/tablet/laptop
- Media centre
- Industrial/Home automation
- Print server
- Web camera
- Wireless access point
- Environmental sensing/monitoring (e.g. weather station)
- IoT applications
- Robotics
- Server/cloud server
- Security monitoring
- Gaming

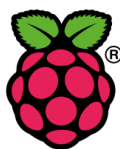


www.rs-components.com/raspberrypi



Raspberry Pi 3 Model B





Raspberry Pi Frequently Asked Questions

How do I get connected?

To get started with your Pi you will need;

- A monitor or TV screen to set-up your Pi
- A keyboard to interact with your Pi
- A mouse to navigate your Pi
- A power supply
- An SD card with the latest version of New Out Of Box Software (NOOBS), to install the operating system that you would like to use.

To get **sound** and **video** you will need cables to suit what your screen or monitor accepts. For those with monitors that accept VGA, a HDMI to VGA adaptor is needed in addition to a HDMI cable, unless you use the composite video output from the Pi.

For an **internet connection**, the Pi B+ and Pi 2 B have an ethernet port. You also have the option of adding a WiFi Adapter/Dongle which may mean that you need a USB Hub if you have run out of USB ports. The Pi 3 already has 802.11 b/g/n wireless LAN and Bluetooth 4.1 (Bluetooth Classic and Low Energy).

Powering my Pi

The Pi has a 5 V microUSB power socket, located on the bottom left hand corner of your Pi board.

Version	Recommended Power Supply Current Capacity
Pi B	1.2 A
Pi A+	700 mA
Pi B+	1.8 A
Pi 2 B	1.8 A
Pi 3 B	2.5 A

Generally, the more USB ports and interfaces you use on your Pi, the more power you are going to need - be careful.

We advise to look at buying a powered USB hub - this means less pressure on your Pi whilst still being able to incorporate all the features and functionality that you want to. When connecting any devices to your Pi, it is advisable to always check the power rating.

Batteries are not a recommended power supply for your Pi.

Note: The Official Raspberry Power Supply Unit for Pi 3 is not a general purpose power supply and must only be used for the Pi 3.



Appendix G – Maestro Code Script

The following appendix shows the Maestro configuration file showing serial settings, servo channel assignments, motion sequences, and onboard scripts used to automate the Astraeus robotic arm

```
<!--Pololu Maestro servo controller settings file, http://www.pololu.com/catalog/product/1350-->
<UscSettings version="1">
  <NeverSuspend>false</NeverSuspend>
  <SerialMode>USB_DUAL_PORT</SerialMode>
  <FixedBaudRate>9600</FixedBaudRate>
  <SerialTimeout>0</SerialTimeout>
  <EnableCrc>false</EnableCrc>
  <SerialDeviceNumber>12</SerialDeviceNumber>
  <SerialMiniSscOffset>0</SerialMiniSscOffset>
  <Channels MiniMaestroServoPeriod="80000" ServoMultiplier="1">
    <!--Period = 20 ms-->
    <!--Channel 0-->
    <Channel name="Base" mode="Servo" min="3968" max="8000" homemode="Off" home="3968"
speed="10" acceleration="10" neutral="6000" range="1905" />
    <!--Channel 1-->
    <Channel name="Shoulder" mode="Servo" min="3968" max="8000" homemode="Off" home="3968"
speed="5" acceleration="5" neutral="6000" range="1905" />
    <!--Channel 2-->
    <Channel name="Elbow" mode="Servo" min="3968" max="8000" homemode="Off" home="3968"
speed="10" acceleration="10" neutral="6000" range="1905" />
    <!--Channel 3-->
    <Channel name="Wrist U/D" mode="Servo" min="3968" max="8000" homemode="Off"
home="3968" speed="10" acceleration="10" neutral="6000" range="1905" />
    <!--Channel 4-->
    <Channel name="Wrist L/R" mode="Servo" min="3968" max="8000" homemode="Off"
home="3968" speed="10" acceleration="10" neutral="6000" range="1905" />
    <!--Channel 5-->
    <Channel name="" mode="Servo" min="3968" max="8000" homemode="Off" home="3968"
speed="0" acceleration="0" neutral="6000" range="1905" />
    <!--Channel 6-->
    <Channel name="" mode="Servo" min="3968" max="8000" homemode="Off" home="3968"
speed="0" acceleration="0" neutral="6000" range="1905" />
    <!--Channel 7-->
    <Channel name="" mode="Servo" min="3968" max="8000" homemode="Off" home="3968"
speed="0" acceleration="0" neutral="6000" range="1905" />
    <!--Channel 8-->
    <Channel name="" mode="Servo" min="3968" max="8000" homemode="Off" home="3968"
speed="0" acceleration="0" neutral="6000" range="1905" />
    <!--Channel 9-->
    <Channel name="" mode="Servo" min="3968" max="8000" homemode="Off" home="3968"
speed="0" acceleration="0" neutral="6000" range="1905" />
    <!--Channel 10-->
    <Channel name="" mode="Servo" min="3968" max="8000" homemode="Off" home="3968"
speed="0" acceleration="0" neutral="6000" range="1905" />
```

```

<!--Channel 11-->
<Channel name="" mode="Servo" min="3968" max="8000" homemode="Off" home="3968"
speed="0" acceleration="0" neutral="6000" range="1905" />
</Channels>
<Sequences>
<Sequence name="Sequence 0" useSpeedAndAcceleration="false">
<Frame name="Initial Position" duration="500">6000 6000 7549 6000 6000 0 0 0 0 0 0 s 10 10 10
10 10 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0</Frame>
</Sequence>
<Sequence name="Sequence 1" useSpeedAndAcceleration="false">
<Frame name="Initial Position" duration="2000">6000 6000 7549 6000 6000 0 0 0 0 0 0 s 10 5 10
10 10 0 0 0 0 0 0 a 10 5 10 10 10 0 0 0 0 0 0</Frame>
<Frame name="ElbowExtension" duration="3750">6000 6000 4202 6000 6000 0 0 0 0 0 0 s 10 10
10 10 10 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0</Frame>
<Frame name="WristAdjustment" duration="2000">6000 6000 4202 7060 6140 0 0 0 0 0 0 s 10 10
10 10 10 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0</Frame>
<Frame name="BaseRotation" duration="1000">6277 6000 4202 7060 6140 0 0 0 0 0 0 s 10 10 10
10 10 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0</Frame>
<Frame name="ArmExtension" duration="4000">6277 7745 4202 8000 6140 0 0 0 0 0 0 s 10 5 10
10 10 0 0 0 0 0 0 a 10 5 10 10 10 0 0 0 0 0 0</Frame>
<Frame name="SweepDownL" duration="3000">6512 6903 6551 7393 6081 0 0 0 0 0 0 s 10 5 10
10 10 0 0 0 0 0 0 a 10 5 10 10 10 0 0 0 0 0 0</Frame>
<Frame name="Center" duration="1000">5944 7060 6336 7021 5905 0 0 0 0 0 0 s 10 5 10 10 10 0
0 0 0 0 0 0 a 10 5 10 10 10 0 0 0 0 0 0</Frame>
<Frame name="Raise For Clearance" duration="1500">5944 6277 6336 6355 5905 0 0 0 0 0 0 s 10
5 10 10 10 0 0 0 0 0 0 a 10 5 10 10 10 0 0 0 0 0 0</Frame>
<Frame name="CenterElbowExtension" duration="2750">6000 6000 4202 6000 6000 0 0 0 0 0 0 s
10 10 10 10 10 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0</Frame>
<Frame name="CenterWristAdjustment" duration="2000">6000 6000 4202 7060 6000 0 0 0 0 0 0 s
10 5 10 10 10 0 0 0 0 0 0 a 10 5 10 10 10 0 0 0 0 0 0</Frame>
<Frame name="CenterArmExtension" duration="4000">6000 7647 4202 7510 5827 0 0 0 0 0 0 s
10 5 10 10 10 0 0 0 0 0 0 a 10 5 10 10 10 0 0 0 0 0 0</Frame>
<Frame name="CenterSweepDown" duration="3000">6000 6649 6884 7608 5827 0 0 0 0 0 0 s 10
5 10 10 10 0 0 0 0 0 0 a 10 5 10 10 10 0 0 0 0 0 0</Frame>
<Frame name="Raise For Clearance" duration="1500">5944 6277 6336 6355 5905 0 0 0 0 0 0 s 10
5 10 10 10 0 0 0 0 0 0 a 10 5 10 10 10 0 0 0 0 0 0</Frame>
<Frame name="ElbowExtension2" duration="2750">6000 6000 4202 6000 6000 0 0 0 0 0 0 s 10 10
10 10 10 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0</Frame>
<Frame name="WristAdjustment2" duration="2000">6000 6000 4202 7060 5435 0 0 0 0 0 0 s 10 5
10 10 10 0 0 0 0 0 0 a 10 5 10 10 10 0 0 0 0 0 0</Frame>
<Frame name="BaseRotation2" duration="1000">5592 6000 4202 7060 5435 0 0 0 0 0 0 s 10 5 10
10 10 0 0 0 0 0 0 a 10 5 10 10 10 0 0 0 0 0 0</Frame>
<Frame name="ArmExtension2" duration="4000">5592 7745 4202 7804 5396 0 0 0 0 0 0 s 10 5 10
10 10 0 0 0 0 0 0 a 10 5 10 10 10 0 0 0 0 0 0</Frame>
<Frame name="SweepDownR" duration="3000">5592 6766 6590 7432 5435 0 0 0 0 0 0 s 10 5 10
10 10 0 0 0 0 0 0 a 10 5 10 10 10 0 0 0 0 0 0</Frame>
<Frame name="Center" duration="1000">5944 7060 6336 7021 5905 0 0 0 0 0 0 s 10 5 10 10 10 0
0 0 0 0 0 0 a 10 5 10 10 10 0 0 0 0 0 0</Frame>
<Frame name="Raise For Clearance" duration="1500">5944 6277 6336 6355 5905 0 0 0 0 0 0 s 10
5 10 10 10 0 0 0 0 0 0 a 10 5 10 10 10 0 0 0 0 0 0</Frame>

```

```

    <Frame name="Initial Position" duration="2000">6000 6000 7549 6000 6000 0 0 0 0 0 0 0 s 10 5 10
10 10 0 0 0 0 0 0 0 a 10 5 10 10 10 0 0 0 0 0 0 0</Frame>
  </Sequence>
  <Sequence name="Sequence 1.1" useSpeedAndAcceleration="false">
    <Frame name="Initial Position" duration="2000">6000 6000 7549 6000 6000 0 0 0 0 0 0 0 s 10 10 10
10 10 0 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0 0</Frame>
    <Frame name="ElbowExtension" duration="3750">6000 6000 4202 6000 6000 0 0 0 0 0 0 0 s 10 10
10 10 10 0 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0 0</Frame>
    <Frame name="WristAdjustment" duration="2000">6000 6000 4202 7060 6140 0 0 0 0 0 0 0 s 10 10
10 10 10 0 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0 0</Frame>
    <Frame name="BaseRotation" duration="1000">6277 6000 4202 7060 6140 0 0 0 0 0 0 0 s 10 10 10
10 10 0 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0 0</Frame>
    <Frame name="ArmExtension" duration="3000">6277 7745 4202 6982 6140 0 0 0 0 0 0 0 s 10 10 10
10 10 0 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0 0</Frame>
    <Frame name="ShiftRight" duration="2000">5592 7745 4202 7060 5435 0 0 0 0 0 0 0 s 10 10 10 10
10 0 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0 0</Frame>
    <Frame name="ShiftDown" duration="1000">5592 7451 4809 7060 5435 0 0 0 0 0 0 0 s 10 10 10 10
10 0 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0 0</Frame>
    <Frame name="ShiftLeft" duration="2000">6512 7549 4809 6845 5964 0 0 0 0 0 0 0 s 10 10 10 10
10 0 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0 0</Frame>
    <Frame name="ShiftDown2" duration="1000">6297 7099 5396 6845 6062 0 0 0 0 0 0 0 s 10 10 10
10 10 0 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0 0</Frame>
    <Frame name="ShiftRight2" duration="2000">5670 7119 5396 7021 5377 0 0 0 0 0 0 0 s 10 10 10 10
10 0 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0 0</Frame>
    <Frame name="ShiftDown3" duration="1000">5651 7080 5768 6962 5494 0 0 0 0 0 0 0 s 10 10 10
10 10 0 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0 0</Frame>
    <Frame name="ShiftLeft2" duration="2000">6414 7021 5768 6962 6023 0 0 0 0 0 0 0 s 10 10 10 10
10 0 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0 0</Frame>
    <Frame name="ShiftDown4" duration="1000">6414 6825 6238 7158 6023 0 0 0 0 0 0 0 s 10 10 10
10 10 0 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0 0</Frame>
    <Frame name="ShiftRight3" duration="2000">5475 7119 5749 6766 5377 0 0 0 0 0 0 0 s 10 10 10 10
10 0 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0 0</Frame>
    <Frame name="ShiftDown5" duration="1000">5475 6903 6297 6962 5318 0 0 0 0 0 0 0 s 10 10 10
10 10 0 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0 0</Frame>
    <Frame name="Shiftleft3" duration="2000">6590 6825 6297 6962 6179 0 0 0 0 0 0 0 s 10 10 10 10
10 0 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0 0</Frame>
    <Frame name="Center" duration="1000">5905 6982 6453 7001 5651 0 0 0 0 0 0 0 s 10 10 10 10 10 0
0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0 0</Frame>
    <Frame name="ClearanceRaise" duration="2000">5905 6473 6453 6316 5651 0 0 0 0 0 0 0 s 10 10
10 10 10 0 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0 0</Frame>
    <Frame name="Initial Position" duration="2000">6000 6000 7549 6000 6000 0 0 0 0 0 0 0 s 10 10 10
10 10 0 0 0 0 0 0 0 a 10 10 10 10 10 0 0 0 0 0 0 0</Frame>
  </Sequence>
</Sequences>
<Script ScriptDone="true">### Sequence subroutines: ###

# Sequence 0
sub Sequence_0
  500 6000 6000 7549 6000 6000 0
  0 0 0 0 0 0 frame_0..11 # Initial Position
return

```

```

# Sequence 1
sub Sequence_1
  2000 6000 6000 7549 6000 6000 0
  0 0 0 0 0 0 frame_0..11 # Initial Position
  3750 4202 frame_2 # ElbowExtension
  2000 7060 6140 frame_3_4 # WristAdjustment
  1000 6277 frame_0 # BaseRotation
  4000 7745 8000 frame_1_3 # ArmExtension
  3000 6512 6903 6551 7393 6081 frame_0..4 # SweepDownL
  1000 5944 7060 6336 7021 5905 frame_0..4 # Center
  1500 6277 6355 frame_1_3 # Raise For Clearance
  2750 6000 6000 4202 6000 6000 frame_0..4 # CenterElbowExtension
  2000 7060 frame_3 # CenterWristAdjustment
  4000 7647 7510 5827 frame_1_3_4 # CenterArmExtension
  3000 6649 6884 7608 frame_1..3 # CenterSweepDown
  1500 5944 6277 6336 6355 5905 frame_0..4 # Raise For Clearance
  2750 6000 6000 4202 6000 6000 frame_0..4 # ElbowExtension2
  2000 7060 5435 frame_3_4 # WristAdjustment2
  1000 5592 frame_0 # BaseRotation2
  4000 7745 7804 5396 frame_1_3_4 # ArmExtension2
  3000 6766 6590 7432 5435 frame_1..4 # SweepDownR
  1000 5944 7060 6336 7021 5905 frame_0..4 # Center
  1500 6277 6355 frame_1_3 # Raise For Clearance
  2000 6000 6000 7549 6000 6000 frame_0..4 # Initial Position
  return
# Sequence 1.1
sub Sequence_11
  2000 6000 6000 7549 6000 6000 0
  0 0 0 0 0 0 frame_0..11 # Initial Position
  3750 4202 frame_2 # ElbowExtension
  2000 7060 6140 frame_3_4 # WristAdjustment
  1000 6277 frame_0 # BaseRotation
  3000 7745 6982 frame_1_3 # ArmExtension
  2000 5592 7060 5435 frame_0_3_4 # ShiftRight
  1000 7451 4809 frame_1_2 # ShiftDown
  2000 6512 7549 6845 5964 frame_0_1_3_4 # ShiftLeft
  1000 6297 7099 5396 6062 frame_0..2_4 # ShiftDown2
  2000 5670 7119 7021 5377 frame_0_1_3_4 # ShiftRight2
  1000 5651 7080 5768 6962 5494 frame_0..4 # ShiftDown3
  2000 6414 7021 6023 frame_0_1_4 # ShiftLeft2
  1000 6825 6238 7158 frame_1..3 # ShiftDown4
  2000 5475 7119 5749 6766 5377 frame_0..4 # ShiftRight3
  1000 6903 6297 6962 5318 frame_1..4 # ShiftDown5
  2000 6590 6825 6179 frame_0_1_4 # Shiftleft3
  1000 5905 6982 6453 7001 5651 frame_0..4 # Center
  2000 6473 6316 frame_1_3 # ClearanceRaise
  2000 6000 6000 7549 6000 6000 frame_0..4 # Initial Position
  return
sub frame_0..11
  11 servo

```

```
10 servo
9 servo
8 servo
7 servo
6 servo
5 servo
4 servo
3 servo
2 servo
1 servo
0 servo
delay
return

sub frame_2
2 servo
delay
return

sub frame_3_4
4 servo
3 servo
delay
return

sub frame_0
0 servo
delay
return

sub frame_1_3
3 servo
1 servo
delay
return

sub frame_0..4
4 servo
3 servo
2 servo
1 servo
0 servo
delay
return

sub frame_3
3 servo
delay
return

sub frame_1_3_4
```

```
4 servo
3 servo
1 servo
delay
return

sub frame_1..3
3 servo
2 servo
1 servo
delay
return

sub frame_1..4
4 servo
3 servo
2 servo
1 servo
delay
return

sub frame_0_3_4
4 servo
3 servo
0 servo
delay
return

sub frame_1_2
2 servo
1 servo
delay
return

sub frame_0_1_3_4
4 servo
3 servo
1 servo
0 servo
delay
return

sub frame_0..2_4
4 servo
2 servo
1 servo
0 servo
delay
return

sub frame_0_1_4
```



```
4 servo
1 servo
0 servo
delay
return
</Script>
</UscSettings>
```

Appendix H – Astraeus Source Code

The following appendix documents the comprehensive source code that Astraeus runs on.

Appendix H.1 – start_all.py

This script serves as the entry point for the system, initializing shared memory and launching both the Flask web server and the main control loop as separate processes using Python's multiprocessing module.

```
1  import multiprocessing
2  import os
3  import sys
4  from multiprocessing import Process, Manager
5
6  # Path setup for dashboard access
7  BASE_DIR = os.path.dirname(os.path.abspath(__file__))
8  DASHBOARD_DIR = os.path.join(BASE_DIR, "dashboard")
9  sys.path.append(DASHBOARD_DIR)
10
11 #PROCESS TARGETS
12 def run_flask(shared):
13     from shared_state import set_shared
14     set_shared(shared)
15
16     from app import app
17     print("[BOOT] Flask process starting...", flush=True)
18     app.run(host='0.0.0.0', port=5000, debug=False)
19
20 def run_main(shared):
21     from shared_state import set_shared
22     set_shared(shared)
23
24     from main import main_loop
25     print("[BOOT] Main control loop starting...", flush=True)
26     main_loop()
27
28 # MAIN RUNNER
29 if __name__ == "__main__":
30     multiprocessing.set_start_method("spawn", force=True)
31
32     # Initialize shared memory once here
33     manager = Manager()
```

```

34     shared = manager.dict({
35         "command": None,
36         "speed": 40,
37         "mode": "idle",
38         "task": None
39     })
40
41     print("[BOOT] Launching Flask and Main loop in parallel", flush=True)
42
43     p1 = Process(target=run_flask, args=(shared,))
44     p2 = Process(target=run_main, args=(shared,))
45
46     p1.start()
47     p2.start()
48
49     print("[BOOT] Both processes started", flush=True)
50
51     p1.join()
52     p2.join()
53

```

Appendix H.2 – main.py

This file contains the main control loop for the system. It continuously checks the selected mode (manual or autonomous) and executes the appropriate behavior, including handling commands, triggering sequences, and running alignment or cleaning tasks.

```
1  import os
2  import sys
3  import time
4
5  # Path setup to access dashboard utilities
6  BASE_DIR = os.path.dirname(os.path.abspath(__file__))
7  DASHBOARD_DIR = os.path.join(BASE_DIR, "dashboard")
8  sys.path.append(DASHBOARD_DIR)
9
10 # Shared state
11 from command_center import get_command, get_speed, clear_command, get_task,
    clear_task, get_mode
12 from maestro_module import trigger_sequence
13 from autonomy import run_autonomy, autonomy_busy, alignment_sequence
14 from logger import log_event, log_warning, log_alert
15 import motors
16 from motors import send_drive_command
17
18 #Manual Command Handling
19 def handle_manual():
20     cmd = get_command()
21     speed = get_speed()
22
23     if cmd:
24         print(f"[MANUAL] Received command: {cmd} | Speed: {speed}%")
25
26     # special functions
27     if cmd == "align1":
28         log_event("[MANUAL] Aligning with Tag 2 (Panel 1)")
29         success = alignment_sequence(2)
30         if success:
31             log_event("[MANUAL] Alignment with Tag 2 successful")
32         else:
33             log_warning("[MANUAL] Alignment with Tag 2 failed")
34
35     elif cmd == "align2":
36         log_event("[MANUAL] Aligning with Tag 3 (Panel 2)")
37         success = alignment_sequence(3)
```

```

38         if success:
39             log_event("[MANUAL] Alignment with Tag 3 successful")
40         else:
41             log_warning("[MANUAL] Alignment with Tag 3 failed")
42
43     elif cmd == "seq1":
44         log_event("Sequence 1 initiated. On standby for 40 seconds.")
45         trigger_sequence(1)
46         time.sleep(2)
47         motors.run_cleaner(100)
48         time.sleep(43)
49         motors.stop_cleaner()
50         time.sleep(4)
51         print("[MANUAL] Sequence 1 complete.")
52
53     elif cmd == "seq2":
54         log_event("Sequence 2 initiated. On standby for 60 seconds.")
55         trigger_sequence(2)
56         time.sleep(2)
57         motors.run_cleaner(100)
58         time.sleep(43)
59         motors.stop_cleaner()
60         time.sleep(4)
61         print("[MANUAL] Sequence 2 complete.")
62
63     else:
64         send_drive_command(cmd, speed)
65
66     clear_command()
67
68 #Autonomous Task Handling
69 def handle_autonomy():
70     if autonomy_busy:
71         return #Already running
72
73     task = get_task()
74     if task and task != "None":
75         print(f"[AUTO] Starting task: {task}")
76         run_autonomy(task)
77         clear_task()
78
79 #Main Loop
80 def main_loop():
81     print("MAIN LOOP is running!")
82     last_mode = None

```

```

83
84     while True:
85         mode = get_mode()
86
87         if mode != last_mode:
88             print(f"[MAIN] Mode switched to: {mode}")
89             last_mode = mode
90
91         if mode == "manual":
92             if not autonomy_busy:
93                 handle_manual()
94             else:
95                 print("[MAIN] Ignored manual input – autonomy is active.")
96
97         elif mode == "autonomous":
98             handle_autonomy()
99
100         time.sleep(0.2)
101
102 if __name__ == "__main__":
103     motors.setup()
104     main_loop()

```

Appendix H.3 – autonomy.py

This module manages the autonomous behavior of the rover. It includes logic for driving toward and aligning with AprilTags, executing cleaning sequences, transitioning between panels, and returning home. It also includes fail-safes and emergency stop mechanisms to handle obstacles and vision loss during operation.

```
1  import time
2  from visual_module import get_tag_data
3  from sharp_sensor import emergency_stop_check, get_smoothed_distances
4  from maestro_module import trigger_sequence
5  import motors
6  from motors import run_cleaner, stop_cleaner
7  from logger import log_event, log_warning, log_alert
8  from command_center import set_mode
9
10 #configurable thresholds
11 TAG4_STOP_WIDTH = 100 # Adjustable: real-world tested width to stop at
    home
12 ALERT_INTERVAL = 10 # Seconds between lost tag alerts
13 ALIGN_X_CENTER = 160 # Center of frame (x)
14 ALIGN_TOLERANCE = 30 # Pixel tolerance for alignment
15 ALIGN_WIDTH_THRESHOLD = 150 # Tag width threshold for closeness
16 SHARP_TOLERANCE = 3 #cm diff between left and right sensors
17 FORWARD_SPEED = 13
18 ADJUST_SPEED_LOW = 9
19 ADJUST_SPEED_HIGH = 13
20 BACKOFF_SPEED = -20
21 BACKOFF_DURATION = 0.5
22 MIN_SAFE_DISTANCE_CM = 7
23 MAX_TRACK_SPEED = 15
24 CENTER_X = 160
25 MAX_DEVIATION = 160 #Full frame width deviation from center = full
    steering effect
26 MAX_TAG_SPEED = 15
27
28 # === Runtime flags ===
29 autonomy_busy = False
30 _e_stop_enabled = False
31 _last_alert_time = {} #Track last alert time per tag_id
32
33 def log_alert_throttled(tag_id, msg):
34     now = time.time()
```

```

35     if tag_id not in _last_alert_time or now - _last_alert_time[tag_id] >=
ALERT_INTERVAL:
36         log_alert(f"[ALERT] Tag {tag_id} lost: {msg}")
37         _last_alert_time[tag_id] = now
38
39 def fail_and_switch_to_manual(reason):
40     log_alert(f"[AUTO FAIL] {reason} - switching to manual mode.")
41     motors.stop_motors()
42     set_mode("manual")
43
44 def drive_until_tag_detected(primary_tag, target_tag, timeout=30):
45     """
46     Drives toward `primary_tag` but switches to `target_tag` when seen.
47     Includes grace period if tags are briefly lost before failing.
48     """
49     print(f"[AUTO] Driving toward Tag {primary_tag}, watching for Tag
{target_tag}...")
50     last_seen_time = time.time()
51     grace_period = 10 # Seconds to allow brief tag loss before fail
52     base_speed = MAX_TAG_SPEED
53     steer_aggression = 0.85 # More aggressive steering response
54
55     while True:
56         if _e_stop_enabled:
57             emergency_stop_check()
58
59         primary_data = get_tag_data(primary_tag)
60         target_data = get_tag_data(target_tag)
61
62         # Target tag found? Switch now
63         if target_data:
64             print(f"[AUTO] Tag {target_tag} detected! Switching pursuit.")
65             return True
66
67         #still following primary tag
68         if primary_data:
69             last_seen_time = time.time() #reset loss timer
70             x = primary_data['x']
71             error = x - CENTER_X
72             correction = min(1.0, abs(error) / MAX_DEVIATION)
73             adjustment = int(correction * base_speed * steer_aggression)
74
75             if error > 0:
76                 left = base_speed - adjustment
77                 right = base_speed + adjustment

```



```

78         else:
79             left = base_speed + adjustment
80             right = base_speed - adjustment
81
82             motors.set_motor_speed(left, right)
83
84         else:
85             # Tag not visible – stop but don't fail immediately
86             motors.stop_motors()
87             time_since_seen = time.time() - last_seen_time
88
89             if time_since_seen >= grace_period:
90                 log_warning(f"[AUTO] Tag not seen for
{int(time_since_seen)}s... holding...")
91                 if time_since_seen >= timeout:
92                     log_alert_throttled(target_tag, f"Timed out after
{timeout}s without seeing any tag")
93                     return False
94
95             time.sleep(0.1)
96
97 def drive_to_tag(tag_id, stop_at_width=None):
98     print(f"[AUTO] Driving to Tag {tag_id}...")
99
100    while True:
101        if _e_stop_enabled:
102            emergency_stop_check()
103
104        data = get_tag_data(tag_id)
105
106        if data:
107            x = data['x']
108            width = data['w']
109            print(f"[AUTO] Tag {tag_id} visible | X={x} W={width}")
110
111            if stop_at_width and width >= stop_at_width:
112                motors.stop_motors()
113                print(f"[AUTO] Tag {tag_id} width reached stop threshold
({stop_at_width})")
114                return True
115
116            # Steering Logic
117            error = x - CENTER_X # + = too far right, - = too far left
118            steer = max(-1.0, min(1.0, error / MAX_DEVIATION)) #
normalize -1 to 1

```

```

119         adjustment = int(steer * MAX_TRACK_SPEED * 0.6) # scale
           effect (~60% max turn)
120
121         left_speed = MAX_TRACK_SPEED - adjustment
122         right_speed = MAX_TRACK_SPEED + adjustment
123
124         # Clamp speeds
125         left_speed = max(-100, min(100, left_speed))
126         right_speed = max(-100, min(100, right_speed))
127
128         motors.set_motor_speed(left_speed, right_speed)
129
130     else:
131         log_alert_throttled(tag_id, "Not visible during drive")
132         motors.stop_motors()
133
134         time.sleep(0.1)
135
136 def alignment_sequence(tag_id):
137     print(f"[AUTO] Starting alignment sequence with Tag {tag_id}...")
138
139     #tunable Parameters
140     ALIGNMENT_DISTANCE_THRESHOLD = 40.0
141     STOP_DISTANCE = 14.0
142     MAX_REVERSE_ATTEMPTS = 3
143     APPROACH_SPEED = 12
144     BOOST_SPEED = 15
145     TURN_FORCE = 1.4
146
147     X_TARGET = 162
148     X_TOLERANCE = 30
149     IR_DIFF_TOL = 5
150     WIDTH_MIN = 160
151     FINAL_HOLD_TIME = 2
152
153     # state Tracking
154     lost_tag_start = None
155     lost_attempts = 0
156     reverse_attempts = 0
157     TAG_LOST_GRACE = 5
158     stable_start = None
159
160     while True:
161         data = get_tag_data(tag_id)
162

```

```

163         if not data:
164             if not lost_tag_start:
165                 lost_tag_start = time.time()
166                 print("[AUTO] Tag temporarily lost. Starting grace
timer...")
167             elif time.time() - lost_tag_start > TAG_LOST_GRACE:
168                 lost_attempts += 1
169                 print(f"[AUTO] Tag not found. Reversing... (Attempt
{lost_attempts})")
170                 motors.set_motor_speed(-20, -20)
171                 time.sleep(0.5)
172                 motors.stop_motors()
173                 lost_tag_start = None
174                 if lost_attempts > 3:
175                     log_alert(f"[AUTO FAIL] Could not complete alignment
with Tag {tag_id}")
176                     return False
177             else:
178                 motors.stop_motors()
179                 time.sleep(0.1)
180             continue
181
182     lost_tag_start = None
183
184     # sensor & Tag Data
185     x = data['x']
186     width = data['w']
187     left_cm, right_cm = get_smoothed_distances()
188     avg_ir = (left_cm + right_cm) / 2
189     ir_diff = abs(left_cm - right_cm)
190     x_error = abs(x - X_TARGET)
191
192     print(f"[ALIGN] X={x:.1f} W={width:.1f} IR_L={left_cm:.1f}
IR_R={right_cm:.1f}")
193
194     #Final Success Check (Override Everything)
195     if (x_error <= X_TOLERANCE and
196         width >= WIDTH_MIN and
197         ir_diff <= IR_DIFF_TOL and
198         left_cm <= STOP_DISTANCE and
199         right_cm <= STOP_DISTANCE):
200
201         motors.stop_motors()
202         if not stable_start:
203             stable_start = time.time()

```

```

204         elif time.time() - stable_start >= FINAL_HOLD_TIME:
205             print("[AUTO] Alignment + approach confirmed. Complete.")
206             log_event(f"[AUTO] Final alignment success with Tag
{tag_id}")
207             return True
208         else:
209             print("[AUTO] Holding to confirm alignment stability...")
210             time.sleep(0.25)
211             continue
212     else:
213         stable_start = None
214
215     #Case: TOO CLOSE + NOT ALIGNED
216     if avg_ir < STOP_DISTANCE + 3 and x_error > X_TOLERANCE:
217         if reverse_attempts < MAX_REVERSE_ATTEMPTS:
218             print("[RECOVERY] Too close and misaligned. Backing
up...")
219             motors.set_motor_speed(-20, -20)
220             time.sleep(0.5)
221             motors.stop_motors()
222             reverse_attempts += 1
223             continue
224         else:
225             log_alert("[AUTO FAIL] Cannot align due to close
proximity.")
226             return False
227
228     # Case: FAR ENOUGH, DO FULL ALIGNMENT
229     if avg_ir >= ALIGNMENT_DISTANCE_THRESHOLD:
230         print("[STAGE 1] Long-range alignment in progress...")
231         if x < X_TARGET - X_TOLERANCE:
232             motors.set_motor_speed(-int(BOOST_SPEED * TURN_FORCE),
BOOST_SPEED)
233         elif x > X_TARGET + X_TOLERANCE:
234             motors.set_motor_speed(BOOST_SPEED, -int(BOOST_SPEED *
TURN_FORCE))
235         elif ir_diff > IR_DIFF_TOL:
236             if left_cm > right_cm:
237                 motors.set_motor_speed(BOOST_SPEED - 2, BOOST_SPEED +
2)
238             else:
239                 motors.set_motor_speed(BOOST_SPEED + 2, BOOST_SPEED -
2)
240         else:
241             motors.set_motor_speed(BOOST_SPEED, BOOST_SPEED)

```

```

242
243     # case: Aligned but not at panel → move in carefully
244     elif (x_error <= X_TOLERANCE and width >= WIDTH_MIN and ir_diff <=
IR_DIFF_TOL):
245         print("[STAGE 2] Approaching panel slowly...")
246         motors.set_motor_speed(APPROACH_SPEED, APPROACH_SPEED)
247
248     # Case: Mid-range and off alignment, adjust while approaching
249     else:
250         print("[STAGE 3] Mid-range adjustment...")
251         if x < X_TARGET - X_TOLERANCE:
252             motors.set_motor_speed(-int(APPROACH_SPEED * TURN_FORCE),
APPROACH_SPEED)
253         elif x > X_TARGET + X_TOLERANCE:
254             motors.set_motor_speed(APPROACH_SPEED, -int(APPROACH_SPEED
* TURN_FORCE))
255         elif ir_diff > IR_DIFF_TOL:
256             if left_cm > right_cm:
257                 motors.set_motor_speed(APPROACH_SPEED - 2,
APPROACH_SPEED + 2)
258             else:
259                 motors.set_motor_speed(APPROACH_SPEED + 2,
APPROACH_SPEED - 2)
260         else:
261             motors.set_motor_speed(APPROACH_SPEED, APPROACH_SPEED)
262
263     time.sleep(0.25)
264
265 def reverse_and_turn_180():
266     print("[AUTO] Reversing before 180 turn...")
267     motors.set_motor_speed(-30, -30)
268     time.sleep(1.0)
269     print("[AUTO] Performing 180 turn...")
270     motors.set_motor_speed(40, -40)
271     time.sleep(3.7)
272     motors.stop_motors()
273     print("[AUTO] 180 turn complete.")
274
275 def transition_to_next_panel():
276     print("Step 1: Reverse curve right (left wheel faster)...")
277     motors.set_motor_speed(-40, -20)
278     time.sleep(0.8)
279
280     print("Step 2: Rotate left to face Panel 2...")

```

```

281     motors.set_motor_speed(30, -30)
282     time.sleep(0.6)
283
284     motors.stop_motors()
285     print("Reposition complete. Ready to align with Panel 2.")
286
287 def return_home():
288     global _e_stop_enabled
289     log_event("[AUTO] Returning home via Tag 4")
290     reverse_and_turn_180()
291
292     _e_stop_enabled = True
293     timeout = 20 # seconds without seeing tag triggers failure
294     last_seen_time = time.time()
295
296     print("[AUTO] Seeking Tag 4 for home return...")
297     while True:
298         if _e_stop_enabled:
299             emergency_stop_check()
300
301         data = get_tag_data(4)
302         if data:
303             last_seen_time = time.time() # reset timeout
304             x = data['x']
305             width = data['w']
306             log_event(f"[AUTO] Tag 4 visible | X={x} W={width}")
307
308             if width >= TAG4_STOP_WIDTH:
309                 motors.stop_motors()
310                 log_event("[AUTO] Rover successfully returned to home
position.")
311                 break
312
313             # Steering logic
314             error = x - CENTER_X
315             steer = max(-1.0, min(1.0, error / MAX_DEVIATION))
316             adjustment = int(steer * MAX_TRACK_SPEED * 0.6)
317             left_speed = MAX_TRACK_SPEED - adjustment
318             right_speed = MAX_TRACK_SPEED + adjustment
319
320             left_speed = max(-100, min(100, left_speed))
321             right_speed = max(-100, min(100, right_speed))
322             motors.set_motor_speed(left_speed, right_speed)
323         else:
324             motors.stop_motors()

```

```

325         log_warning("[AUTO] Tag 4 not currently visible")
326
327         if time.time() - last_seen_time > timeout:
328             _e_stop_enabled = False
329             motors.stop_motors()
330             fail_and_switch_to_manual("Failed to detect Tag 4 during
return home")
331             return
332
333         time.sleep(0.1)
334
335     _e_stop_enabled = False
336
337 def run_autonomy(task):
338     global autonomy_busy, _e_stop_enabled
339     if autonomy_busy:
340         log_warning("[AUTO] Already running. Ignoring duplicate task.")
341         return
342
343     autonomy_busy = True
344     try:
345         log_event(f"[AUTO] Starting task: {task}")
346
347         if task == "panel1":
348             _e_stop_enabled = True
349             if not drive_until_tag_detected(1, 2, timeout=20):
350                 fail_and_switch_to_manual("Lost sight of both Tag 1 and
Tag 2 during approach")
351                 return
352             _e_stop_enabled = False
353
354             if not alignment_sequence(2):
355                 fail_and_switch_to_manual("Alignment with Tag 2 failed")
356                 return
357
358             log_event("Sequence 1 initiated. On standby for 40 seconds.")
359             trigger_sequence(1)
360             time.sleep(2)
361             motors.run_cleaner(100)
362             time.sleep(43)
363             motors.stop_cleaner()
364             time.sleep(4)
365
366             log_event("Returning home...")

```

```

367         return_home()
368
369     elif task == "panel2":
370         _e_stop_enabled = True
371         if not drive_until_tag_detected(1, 3, timeout=20):
372             fail_and_switch_to_manual("Lost sight of both Tag 1 and
Tag 3 during approach")
373             return
374         _e_stop_enabled = False
375
376         if not alignment_sequence(3):
377             fail_and_switch_to_manual("Alignment with Tag 3 failed")
378             return
379
380         log_event("Sequence 1 initiated. On standby for 40 seconds.")
381         trigger_sequence(1)
382         time.sleep(2)
383         motors.run_cleaner(100)
384         time.sleep(43)
385         motors.stop_cleaner()
386         time.sleep(4)
387         log_event("Returning home...")
388         return_home()
389
390     elif task == "all":
391         _e_stop_enabled = True
392         if not drive_until_tag_detected(1, 2, timeout=20):
393             fail_and_switch_to_manual("Lost sight of both Tag 1 and
Tag 2 during approach")
394             return
395         _e_stop_enabled = False
396
397         if not alignment_sequence(2):
398             fail_and_switch_to_manual("Alignment with Tag 2 failed")
399             return
400
401         log_event("Sequence 1 initiated. On standby for 40 seconds.")
402         run_cleaner(50)
403         trigger_sequence(1)
404         time.sleep(40)
405         stop_cleaner()
406
407         log_event("Transitioning to Panel 2...")
408         transition_to_next_panel()
409

```



```

410         _e_stop_enabled = True
411         #try to align with Tag 3 immediately. If not visible, fallback
to drive_until
412         if not get_tag_data(3):
413             if not drive_until_tag_detected(1, 3, timeout=20):
414                 fail_and_switch_to_manual("Lost sight of both Tag 1
and Tag 3 after transition")
415                 return
416         _e_stop_enabled = False
417
418         if not alignment_sequence(3):
419             fail_and_switch_to_manual("Alignment with Tag 3 failed")
420             return
421
422         log_event("Sequence 1 initiated. On standby for 40 seconds.")
423         run_cleaner(50)
424         trigger_sequence(1)
425         time.sleep(40)
426         stop_cleaner()
427
428         log_event("Returning home...")
429         return_home()
430
431     else:
432         log_warning(f"[AUTO] Unknown task: {task}")
433
434     finally:
435         autonomy_busy = False
436         print("[AUTO] Task complete. Autonomy released.")
437
438 #Standalone Testing
439 if __name__ == "__main__":
440     print("[TEST] Autonomy Movement Test Mode")
441     while True:
442         print("\nSelect a test:")
443         print("1. Test 180° Turn")
444         print("2. Test Reverse and Realign (Panel Transition)")
445         print("3. Exit")
446         choice = input("Enter choice (1/2/3): ").strip()
447
448         if choice == "1":
449             reverse_and_turn_180()
450         elif choice == "2":

```

```
451         transition_to_next_panel()
452     elif choice == "3":
453         print("Exiting test mode.")
454         break
455     else:
456         print("Invalid selection.")
457
```

Appendix H.4 – motors.py

This file controls all motor functions using the gpiozero library, including drive motors and the cleaning mechanism. It provides utility functions to set motor speeds, run the cleaner, stop all movement, and interpret high-level drive commands like forward, backward, or turn.

```
1  from gpiozero import Motor, PWMOutputDevice
2  from time import sleep
3
4  # Pin Config (BCM)
5  LEFT_IN_A   = 17
6  LEFT_IN_B   = 27
7  LEFT_PWM    = 18
8
9  RIGHT_IN_A  = 23
10 RIGHT_IN_B  = 24
11 RIGHT_PWM   = 19
12
13 CLEAN_IN_A  = 5
14 CLEAN_IN_B  = 6
15 CLEAN_PWM   = 13
16
17 # setup Motors and PWM
18 left_motor = Motor(forward=LEFT_IN_A, backward=LEFT_IN_B)
19 right_motor = Motor(forward=RIGHT_IN_A, backward=RIGHT_IN_B)
20 clean_motor = Motor(forward=CLEAN_IN_A, backward=CLEAN_IN_B)
21
22 left_pwm = PWMOutputDevice(LEFT_PWM)
23 right_pwm = PWMOutputDevice(RIGHT_PWM)
24 clean_pwm = PWMOutputDevice(CLEAN_PWM)
25
26 #Setup Function
27 def setup():
28     stop_motors()
29     stop_cleaner()
30     print("[GPIOZERO] Motors initialized.")
31
32 # Motor Control
33 def _set_single_motor(motor, pwm_device, speed, label=""):
34     speed = max(-100, min(100, speed))
35     duty = abs(speed) / 100.0
36     direction = "Stopped"
37
38     if speed > 0:
```

```

39         motor.forward()
40         pwm_device.value = duty
41         direction = "Forward"
42     elif speed < 0:
43         motor.backward()
44         pwm_device.value = duty
45         direction = "Reverse"
46     else:
47         motor.stop()
48         pwm_device.value = 0.0
49
50     if label:
51         print(f"[MOTOR] {label} => Speed: {abs(speed)}% | Direction:
{direction}")
52
53 def set_motor_speed(left_speed, right_speed):
54     _set_single_motor(left_motor, left_pwm, left_speed, label="Left
Motor")
55     _set_single_motor(right_motor, right_pwm, right_speed, label="Right
Motor")
56
57 def send_drive_command(cmd, speed):
58     if cmd == "forward":
59         set_motor_speed(speed, speed)
60     elif cmd == "backward":
61         set_motor_speed(-speed, -speed)
62     elif cmd == "left":
63         set_motor_speed(-speed, speed)
64     elif cmd == "right":
65         set_motor_speed(speed, -speed)
66     elif cmd == "forward_left":
67         set_motor_speed(speed // 2, speed)
68     elif cmd == "forward_right":
69         set_motor_speed(speed, speed // 2)
70     elif cmd == "backward_left":
71         set_motor_speed(-(speed // 2), -speed)
72     elif cmd == "backward_right":
73         set_motor_speed(-speed, -(speed // 2))
74     elif cmd == "stop":
75         stop_motors()
76     else:
77         print(f"[WARN] Unknown command: {cmd}")
78
79 def run_cleaner(speed):
80     _set_single_motor(clean_motor, clean_pwm, speed, label="Cleaner")

```

```
81
82 def stop_motors():
83     left_motor.stop()
84     right_motor.stop()
85     left_pwm.value = 0.0
86     right_pwm.value = 0.0
87
88 def stop_cleaner():
89     clean_motor.stop()
90     clean_pwm.value = 0.0
91
92 def cleanup():
93     stop_motors()
94     stop_cleaner()
95     print("[GPIOZERO] Cleanup complete.")
96
97 #Test Block
98 if __name__ == "__main__":
99     stop_motors()
100
```

Appendix H.5 – sharp_sensors.py

```
1  import board
2  import busio
3  import adafruit_ads1x15.ads1115 as ADS
4  from adafruit_ads1x15.analog_in import AnalogIn
5  from collections import deque
6  import time
7  from logger import *
8  import motors
9  import signal
10 import sys
11
12 # I2C setup
13 i2c = busio.I2C(board.SCL, board.SDA)
14 ads = ADS.ADS1115(i2c)
15 ads.gain = 1 # Set gain for 3.3V max input
16
17 # Channel setup
18 left_sensor = AnalogIn(ads, ADS.P0) # A0
19 right_sensor = AnalogIn(ads, ADS.P1) # A1
20
21 # Smoothing buffers
22 left_buffer = deque(maxlen=10)
23 right_buffer = deque(maxlen=10)
24
25 #Exponential Moving Average
26 left_ema = None
27 right_ema = None
28 alpha = 0.2
29
30 _last_triggered = 0
31 _obstacle_start_time = None
32
33 def voltage_to_distance(voltage):
34     if voltage <= 0.1:
35         return float('inf') #Out of range
36     return round(27.86 / (voltage - 0.1), 2)
37
38 def update_buffers():
39     global left_ema, right_ema
40     left_voltage = left_sensor.voltage
41     right_voltage = right_sensor.voltage
42
43     left_dist = voltage_to_distance(left_voltage)
```

```

44     right_dist = voltage_to_distance(right_voltage)
45
46     left_buffer.append(left_dist)
47     right_buffer.append(right_dist)
48
49     left_ema = left_dist if left_ema is None else alpha * left_dist + (1 -
alpha) * left_ema
50     right_ema = right_dist if right_ema is None else alpha * right_dist +
(1 - alpha) * right_ema
51
52 def get_smoothed_distances():
53     update_buffers()
54     return round(left_ema, 2), round(right_ema, 2)
55
56 def is_obstacle_detected(threshold_cm=40.0):
57     left_dist, right_dist = get_smoothed_distances()
58     return left_dist < threshold_cm or right_dist < threshold_cm
59
60 def emergency_stop_check(threshold_cm=40.0, cooldown=7):
61     global _last_triggered, _obstacle_start_time
62
63     now = time.time()
64     if now - _last_triggered < cooldown:
65         return
66
67     if is_obstacle_detected(threshold_cm):
68         print("Obstacle detected! Pausing movement...")
69         log_warning("Obstacle detected! Pausing movement...")
70         motors.stop_motors()
71
72         if _obstacle_start_time is None:
73             _obstacle_start_time = now
74         elif now - _obstacle_start_time > 20:
75             print("Obstacle still present after 1 minute! Logging
alert...")
76             log_alert("Obstacle still present after 1 minute! Logging
alert...")
77
78             time.sleep(5)
79             _last_triggered = time.time()
80     else:
81         _obstacle_start_time = None # Reset timer if path is clear
82
83 # Graceful shutdown
84 def shutdown_handler(sig, frame):

```

```

85     print("\nExiting. Cleaning up GPIO...")
86     motors.cleanup()
87     sys.exit(0)
88
89 signal.signal(signal.SIGINT, shutdown_handler)
90
91 # Test block
92 if __name__ == "__main__":
93     motors.setup()
94     print("Running Sharp Sensor Test...")
95     while True:
96         left, right = get_smoothed_distances()
97         print(f"Left: {left:.2f} cm | Right: {right:.2f} cm")
98
99         #log_event(f"Left: {left:.2f} cm | Right: {right:.2f} cm")
100
101         #emergency_stop_check()
102
103         time.sleep(0.2)
104

```


Appendix H.6 – visual_module.py

This module interfaces with the HuskyLens AI vision sensor over I2C to detect and track AprilTags. It provides smoothed position and size data for specified tag IDs using a rolling buffer, improving stability in visual alignment tasks.

```
1  import sys
2  import os
3  import time
4  from collections import deque
5  from huskylib import HuskyLensLibrary
6  from logger import log_event
7
8  # Init HuskyLens in I2C mode
9  huskylens = HuskyLensLibrary("I2C", address=0x32)
10
11 #smoothing buffers for each tag ID
12 _buffers = {
13     1: deque(maxlen=5),
14     2: deque(maxlen=5),
15     3: deque(maxlen=5),
16     4: deque(maxlen=5)
17 }
18
19 def get_tag_data(tag_id):
20     """
21     Returns smoothed data for a given tag_id.
22     Output: dict with x, y, width, height or None if tag not found.
23     """
24     try:
25         huskylens.requestAll()
26         blocks = huskylens.blocks()
27         if not isinstance(blocks, list):
28             blocks = [blocks]
29
30         for block in blocks:
31             if hasattr(block, 'ID') and block.ID == tag_id:
32                 data = {'x': block.x, 'y': block.y, 'w': block.width, 'h':
33                     block.height}
34                 _buffers[tag_id].append(data)
35                 return _get_smoothed(tag_id)
36     except Exception as e:
37         print(f"[visual_module] Error: {e}")
38     return None
```

```

38
39 def _get_smoothed(tag_id):
40     """
41     Returns the average of x, y, w, h from the buffer for the given tag.
42     """
43     buf = _buffers[tag_id]
44     if not buf:
45         return None
46     avg = {
47         'x': round(sum(d['x'] for d in buf) / len(buf), 2),
48         'y': round(sum(d['y'] for d in buf) / len(buf), 2),
49         'w': round(sum(d['w'] for d in buf) / len(buf), 2),
50         'h': round(sum(d['h'] for d in buf) / len(buf), 2)
51     }
52     return avg
53
54 # test
55 if __name__ == "__main__":
56     while True:
57         for tag_id in [1, 2, 3, 4]:
58             data = get_tag_data(tag_id)
59             if data:
60                 print(f"Tag {tag_id}: X={data['x']} Y={data['y']}
61                     W={data['w']} H={data['h']}")
62                 log_event(f"Tag {tag_id}: X={data['x']} Y={data['y']}
63                     W={data['w']} H={data['h']}")
64                 time.sleep(0.5)

```

Appendix H.7 – maestro_module.py

This module provides a lightweight interface for triggering subroutines on the Pololu Maestro controller using serial communication. It abstracts the low-level protocol into a simple `trigger_sequence()` function, allowing the system to initiate scripted hardware actions like servo movements.

```
1  """
2  maestro_module.py
3
4  Custom Python interface for triggering Pololu Maestro script subroutines
   via serial commands.
5
6  DISCLAIMER:
7  The hexadecimal command values used in this module are based on the
   official Pololu Maestro
8  Serial Script Command Protocol as documented in the Pololu Maestro User's
   Guide:
9
10 https://www.pololu.com/docs/pdf/0j40/maestro.pdf
11
12 All protocol definitions, device numbers, and command structures follow
   the specifications
13 outlined by Pololu. This module simply provides a lightweight and project-
   specific interface
14 to those commands and does not represent original protocol design.
15
16 Author: Pedro Cabrera
17 Date: 2025-07-11
18 """
19
20 import serial
21 import time
22
23 #CONFIGURATION
24 DEVICE_NUMBER = 0x0C # Default Maestro device number (0x0C = 12)
25 PORT = "/dev/ttyACM0" # Serial port where Maestro is connected
26 BAUDRATE = 9600 # Default USB baudrate for Maestro communication
27
28 def trigger_sequence(subroutine: int):
29     """
30     Triggers a script subroutine on the Maestro controller.
31
```

```

32     Args:
33         subroutine (int): The subroutine number to run (0 = first defined
in Maestro script)
34         """
35         command = [0xAA, DEVICE_NUMBER, 0x27, subroutine]
36         try:
37             with serial.Serial(PORT, BAUDRATE, timeout=1) as maestro:
38                 maestro.write(bytearray(command))
39                 print(f"[Maestro] Triggered subroutine {subroutine}")
40         except Exception as e:
41             print(f"[Maestro] Error triggering subroutine {subroutine}: {e}")
42
43 #TEST BLOCK
44 if __name__ == "__main__":
45     print("Testing Maestro subroutine trigger...")
46     trigger_sequence(1)
47     print("40 sec delay initiated")
48     time.sleep(40)
49     print("sequence complete")
50

```

Appendix H.8 – logger.py

This module handles system logging by writing timestamped messages to an SQLite database. It supports three log levels, INFO, WARNING, and ALERT, and provides utility functions for structured logging throughout the application.

```
1  import sqlite3
2  import datetime
3  import os
4
5  DB_FILE = os.path.join(os.path.dirname(__file__), 'event_log.db')
6
7  def log_event(message, level='INFO'):
8      timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
9      conn = sqlite3.connect(DB_FILE)
10     c = conn.cursor()
11     c.execute("INSERT INTO logs (timestamp, level, message) VALUES (?, ?,
12     ?)",
13               (timestamp, level, message))
14     conn.commit()
15     conn.close()
16     print(f"[{timestamp}] ({level}) {message}")
17
18 def log_warning(message):
19     log_event(message, level='WARNING')
20
21 def log_alert(message):
22     log_event(message, level='ALERT')
```

Appendix H.9 – shared_state.py

This file provides a simple interface for setting and retrieving a global shared state dictionary across multiple processes. It allows safe, centralized management of shared variables such as the current operating mode.

```
1  shared = None
2
3  def set_shared(s):
4      global shared
5      shared = s
6
7  def get_shared():
8      return shared
9
10 def set_mode(new_mode):
11     if shared is not None:
12         shared["mode"] = new_mode
13
14
```

Appendix H.10 – command_center.py

This module provides helper functions to manage shared commands, speed, mode, and tasks through the shared memory dictionary. It serves as the central hub for setting and retrieving control parameters used by different system components.

```
1  from shared_state import get_shared
2
3  def set_command(cmd):
4      get_shared()["command"] = cmd
5
6  def get_command():
7      return get_shared().get("command", None)
8
9  def clear_command():
10     get_shared()["command"] = None
11
12  def set_speed(speed):
13     get_shared()["speed"] = speed
14
15  def get_speed():
16     return get_shared().get("speed", 40)
17
18  def set_mode(mode):
19     get_shared()["mode"] = mode
20
21  def get_mode():
22     return get_shared().get("mode", "idle")
23
24  def set_task(task):
25     get_shared()["task"] = task
26
27  def get_task():
28     return get_shared().get("task", None)
29
30  def clear_task():
31     get_shared()["task"] = None
32
```

Appendix H.11 – failsafe_manual.py

This script provides a keyboard-controlled fallback interface for manual operation using the curses library. It allows real-time movement via WASD-style input and includes hotkeys to trigger cleaning sequences and alignment routines, ensuring basic functionality even if the web dashboard becomes unavailable.

```
1  import curses
2  import time
3  import motors
4  from maestro_module import trigger_sequence
5  from autonomy import alignment_sequence
6
7  #constants
8  SPEED = 40
9  ALIGN_TAG_1 = 2
10 ALIGN_TAG_2 = 3
11
12 # movement Commands Map
13 key_to_command = {
14     'w': 'forward',
15     'a': 'left',
16     's': 'backward',
17     'd': 'right',
18     'q': 'forward_left',
19     'e': 'forward_right',
20     'z': 'backward_left',
21     'x': 'backward_right'
22 }
23
24 def main(screen):
25     curses.cbreak()
26     screen.nodelay(True)
27     screen.keypad(True)
28
29     print(" Failsafe Manual Mode Active (Ctrl+C to exit)\n")
30     print(" Hold movement keys (WASDQEZX) to drive")
31     print(" Press 1 = Sequence 1 | 2 = Sequence 2")
32     print(" Press 3 = Align Panel 1 | 4 = Align Panel 2\n")
33
34     last_key = None
35     motors.setup()
36
```



```

37     try:
38         while True:
39             key = screen.getch()
40             if key == -1:
41                 if last_key:
42                     motors.stop_motors()
43                     print(f"[STOP] Released {last_key.upper()}")
44                     last_key = None
45                     time.sleep(0.01)
46                     continue
47
48             try:
49                 char = chr(key).lower()
50             except ValueError:
51                 continue # ignore unprintable chars
52
53             # handle one-time actions
54             if char == '1':
55                 print("[SEQ] Triggering Sequence 1")
56                 motors.run_cleaner(60)
57                 trigger_sequence(1)
58                 print("[CLEANER] Running for 40 seconds...")
59                 time.sleep(40)
60                 motors.stop_cleaner()
61                 print("[CLEANER] Stopped after sequence 1")
62
63             elif char == '2':
64                 print("[SEQ] Triggering Sequence 2")
65                 motors.run_cleaner(60)
66                 trigger_sequence(2)
67                 print("[CLEANER] Running for 60 seconds...")
68                 time.sleep(60)
69                 motors.stop_cleaner()
70                 print("[CLEANER] Stopped after sequence 2")
71
72             elif char == '3':
73                 print("[ALIGN] Aligning to Panel 1 (Tag 2)")
74                 alignment_sequence(ALIGN_TAG_1)
75             elif char == '4':
76                 print("[ALIGN] Aligning to Panel 2 (Tag 3)")
77                 alignment_sequence(ALIGN_TAG_2)
78             elif char in key_to_command:
79                 if char != last_key:
80                     cmd = key_to_command[char]
81                     motors.send_drive_command(cmd, SPEED)

```

```

82         print(f"[MOVE] {char.upper()} → {cmd} at {SPEED}%")
83         last_key = char
84     else:
85         if last_key:
86             motors.stop_motors()
87             print(f"[STOP] Released {last_key.upper()}")
88             last_key = None
89
90         time.sleep(0.01)
91
92     except KeyboardInterrupt:
93         print("\n Exiting manual mode...")
94         motors.cleanup()
95         time.sleep(0.5)
96
97 if __name__ == "__main__":
98     curses.wrapper(main)

```

Appendix H.12 – init_db.py

This script initializes the SQLite database used for logging system events. It creates a logs table with fields for ID, timestamp, severity level, and message. This setup is required before any logging can occur.

```
1  import sqlite3
2
3  def initialize_db():
4      conn = sqlite3.connect('event_log.db')
5      c = conn.cursor()
6      c.execute("""
7          CREATE TABLE IF NOT EXISTS logs (
8              id INTEGER PRIMARY KEY AUTOINCREMENT,
9              timestamp TEXT NOT NULL,
10             level TEXT NOT NULL,
11             message TEXT NOT NULL
12         )
13     """)
14     conn.commit()
15     conn.close()
16
17 if __name__ == "__main__":
18     initialize_db()
19     print("Database initialized.")
```

Appendix H.13 – app.py

This Flask application powers the web-based control dashboard. It provides routes for selecting modes, sending commands, adjusting speed, viewing logs, and assigning tasks. It also interfaces with shared memory to synchronize actions between the UI and the rover's core logic.

```
1  import os
2  import sys
3  sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
4  '..')))
5  from command_center import set_command, set_speed, set_task, set_mode
6  from flask import Flask, render_template, request, redirect, url_for,
7  jsonify
8  import sqlite3
9
10
11 current_speed = 40
12 current_mode = "idle" # Options: "idle", "manual", "autonomous"
13
14 app = Flask(__name__)
15
16 BASE_DIR = os.path.abspath(os.path.join(os.path.dirname(__file__), '..'))
17 DB_PATH = os.path.join(BASE_DIR, 'event_log.db')
18
19 selected_task = None # Stores current panel selection
20
21 #Database Helper
22 def fetch_logs(level_filter=None, limit=50, offset=0):
23     conn = sqlite3.connect(DB_PATH)
24     c = conn.cursor()
25     if level_filter and level_filter != "ALL":
26         c.execute("SELECT * FROM logs WHERE level=? ORDER BY id DESC LIMIT
27 ? OFFSET ?", (level_filter, limit, offset))
28     else:
29         c.execute("SELECT * FROM logs ORDER BY id DESC LIMIT ? OFFSET ?",
30 (limit, offset))
31     logs = c.fetchall()
32     conn.close()
33     return logs
34
35 #Routes
36
37 @app.route("/")
38 def home():
39     return render_template("home.html")
```

```

35
36 @app.route("/mode")
37 def mode_selection():
38     return render_template("mode.html")
39
40 @app.route("/autonomous", methods=["GET", "POST"])
41 def autonomous():
42     global selected_task
43     if request.method == "POST":
44         selected_task = request.form["task"]
45         set_task(selected_task)
46         return redirect(url_for("autonomous"))
47     return render_template("autonomous.html", selected_task=selected_task)
48
49 @app.route("/manual")
50 def manual_control():
51     return render_template("manual.html")
52
53 @app.route("/logs")
54 def show_logs():
55     level = request.args.get("level", "ALL")
56     logs = fetch_logs(level)
57     return render_template("index.html", logs=logs, level=level)
58
59 @app.route("/logs/more")
60 def load_more_logs():
61     level = request.args.get("level", "ALL")
62     offset = int(request.args.get("offset", 50))
63     logs = fetch_logs(level, offset=50, limit=50)
64     return render_template("log_table.html", logs=logs)
65
66 @app.route("/manual/command", methods=["POST"])
67 def manual_command():
68     data = request.get_json()
69     cmd = data.get("command")
70     set_command(cmd)
71     print(f"[Manual CMD] Sent to buffer: {cmd}")
72     return jsonify({"status": "ok"}), 200
73
74 @app.route("/manual/speed", methods=["POST"])
75 def set_speed_handler():
76     data = request.get_json()
77     speed = int(data.get("speed", 40))
78     set_speed(speed)
79     print(f"[Speed] Updated in buffer: {speed}%")

```

```

80     return jsonify({"status": "updated", "speed": speed}), 200
81
82 @app.route("/set_mode", methods=["POST"])
83 def set_mode_route():
84     data = request.get_json()
85     mode = data.get("mode")
86     if mode in ["manual", "autonomous"]:
87         set_mode(mode)
88         print(f"[Mode] Updated to {mode}")
89         return jsonify({"status": "ok", "mode": mode}), 200
90     return jsonify({"status": "error", "message": "Invalid mode"}), 400
91
92 @app.route("/get_mode", methods=["GET"])
93 def get_mode():
94     return jsonify({"mode": current_mode}), 200
95
96 #Flask run
97 if __name__ == "__main__":
98     app.run(host='0.0.0.0', port=5000, debug=False)
99

```

Appendix H.14 – joystick.js

This JavaScript file enables joystick-based manual control on the web dashboard using the NippleJS library. It translates joystick movements into directional commands and sends them to the backend via POST requests for real-time rover control.

```
1  const joystickZone = document.getElementById('joystick');
2  const joystick = nipplejs.create({
3    zone: joystickZone,
4    mode: 'static',
5    position: { left: '50%', top: '50%' },
6    color: 'blue',
7    size: 150
8  });
9
10 let lastSent = 0;
11 const SEND_INTERVAL = 300;
12
13 joystick.on('move', (evt, data) => {
14   if (!data || !data.angle) return;
15
16   const now = Date.now();
17   if (now - lastSent < SEND_INTERVAL) return;
18   lastSent = now;
19
20   // Flip horizontally + rotate -90°
21   let angle = 450 - data.angle.degree;
22   if (angle >= 360) angle -= 360;
23
24   let command = 'stop';
25
26   if (angle >= 337.5 || angle < 22.5) {
27     command = 'forward';
28   } else if (angle >= 22.5 && angle < 67.5) {
29     command = 'forward_right';
30   } else if (angle >= 67.5 && angle < 112.5) {
31     command = 'right';
32   } else if (angle >= 112.5 && angle < 157.5) {
33     command = 'backward_right';
34   } else if (angle >= 157.5 && angle < 202.5) {
35     command = 'backward';
36   } else if (angle >= 202.5 && angle < 247.5) {
37     command = 'backward_left';
38   } else if (angle >= 247.5 && angle < 292.5) {
```

```

39     command = 'left';
40 } else if (angle >= 292.5 && angle < 337.5) {
41     command = 'forward_left';
42 }
43
44     sendJoystickCommand(command);
45 });
46
47 joystick.on('end', () => {
48     sendJoystickCommand('stop');
49 });
50
51 function sendJoystickCommand(cmd) {
52     fetch('/manual/command', {
53         method: 'POST',
54         headers: { 'Content-Type': 'application/json' },
55         body: JSON.stringify({ command: cmd })
56     });
57 }
58

```


Appendix H.15 – home.html

This is the landing page for the Astraeus web interface. It provides users with navigation options to select between autonomous or manual operation modes and to view system logs. The design is clean and responsive for ease of use on desktop or mobile devices.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Astraeus Home</title>
6      <style>
7          body {
8              margin: 0;
9              font-family: 'Segoe UI', sans-serif;
10             background-color: #f4f4f4;
11         }
12
13         header {
14             background-color: black;
15             text-align: center;
16             padding: 1rem;
17         }
18
19         header img {
20             max-width: 100%;
21             height: auto;
22         }
23
24         .container {
25             text-align: center;
26             padding: 2rem;
27         }
28
29         h1 {
30             font-size: 2.5rem;
31             color: #333;
32         }
33
34         .button-container {
35             margin-top: 2rem;
36         }
37
38         .main-btn {
```

```

39         padding: 1rem 2rem;
40         margin: 1rem;
41         text-decoration: none;
42         color: white;
43         font-size: 1.2rem;
44         font-weight: bold;
45         border-radius: 8px;
46         display: inline-block;
47         transition: background-color 0.2s ease;
48     }
49
50     .main-btn:hover {
51         opacity: 0.9;
52     }
53
54     .mode-btn {
55         background-color: #007bff;
56     }
57
58     .logs-btn {
59         background-color: #28a745;
60     }
61 </style>
62 </head>
63 <body>
64
65 <header>
66     
68 </header>
69 <div class="container">
70     <h1>Welcome to Astraeus</h1>
71     <p>Select a mode or view logs from the rover system.</p>
72
73     <div class="button-container">
74         <a href="{{ url_for('mode_selection') }}" class="main-btn mode-
75         btn">⚙️ Mode Selection</a>
76         <a href="{{ url_for('show_logs') }}" class="main-btn logs-btn">📄
77         View Logs</a>
78     </div>
79 </div>
80 </body>
81 </html>

```

Appendix H.16 – index.html

This page displays system logs from the rover in a filterable and scrollable format. Users can view logs by severity (ALL, WARNING, ALERT), load older entries, and see real-time updates with visual indicators when new messages arrive. The interface is styled for clarity and ease of use.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Logger</title>
6      <style>
7          .log-btn {
8              display: inline-block;
9              margin: 0 10px;
10             padding: 0.5em 1.2em;
11             border-radius: 8px;
12             text-decoration: none;
13             font-weight: bold;
14             font-family: 'Segoe UI', sans-serif;
15             font-size: 14px;
16             transition: all 0.2s ease;
17             background-color: #e0e0e0;
18             color: #000;
19             box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
20         }
21
22         .log-btn:hover {
23             background-color: #ccc;
24         }
25
26         .log-btn.active {
27             border: 2px solid #007bff;
28             background-color: #d9edff;
29             color: #007bff;
30         }
31
32         .log-btn.warning {
33             background-color: #fff2c2;
34             color: #b8860b;
35         }
36
37         .log-btn.warning.active {
38             background-color: #ffeb99;
```

```

39         border-color: #b8860b;
40     }
41
42     .log-btn.alert {
43         background-color: #ffcccc;
44         color: #b22222;
45     }
46
47     .log-btn.alert.active {
48         background-color: #ff9999;
49         border-color: #b22222;
50     }
51
52     .log-btn.neutral {
53         background-color: #eee;
54         color: #333;
55     }
56
57     .log-btn.neutral:hover {
58         background-color: #ddd;
59     }
60 </style>
61 </head>
62 <body>
63     <header style="background-color:black; text-align:center;
padding:1rem;">
64         
65     </header>
66
67     <div class="container">
68         <div class="filter-tabs" style="text-align:center; margin: 1.5rem
0;">
69             <a href="{{ url_for('home') }}" class="log-btn neutral">
Home</a>
70             <a href="{{ url_for('show_logs', level='ALL') }}" class="log-
btn {{ 'active' if level == 'ALL' else '' }}">ALL</a>
71             <a href="{{ url_for('show_logs', level='WARNING') }}"
class="log-btn warning {{ 'active' if level == 'WARNING' else ''
}}">WARNING</a>
72             <a href="{{ url_for('show_logs', level='ALERT') }}"
class="log-btn alert {{ 'active' if level == 'ALERT' else '' }}">ALERT</a>
73         </div>
74
75         <div id="logTable">

```

```

76         {% include 'log_table.html' %}
77     </div>
78
79     <button id="loadMoreBtn" style="
80         display: block;
81         margin: 2rem auto;
82         padding: 0.75rem 1.5rem;
83         background: #333;
84         color: white;
85         border: none;
86         border-radius: 5px;
87         font-weight: bold;
88         cursor: pointer;
89     ">
90         Load Older Logs
91     </button>
92
93     <div id="newMessages" onclick="scrollToTop()" style="
94         position: fixed;
95         bottom: 0;
96         left: 50%;
97         transform: translateX(-50%);
98         background: #007bff;
99         color: white;
100        padding: 0.75em 1.5em;
101        border-radius: 1rem 1rem 0 0;
102        display: none;
103        z-index: 999;
104        cursor: pointer;
105        font-weight: bold;
106    ">
107        New logs available – click to return
108    </div>
109 </div>
110
111 <script>
112     const logTable = document.getElementById('logTable');
113     const newMessages = document.getElementById('newMessages');
114     const level = "{ { level } }";
115     let autoScroll = true;
116     let offset = 50;
117
118     function scrollToTop() {
119         window.scrollTo({ top: 0, behavior: 'smooth' });
120         autoScroll = true;

```

```

121         newMessages.style.display = 'none';
122     }
123
124     window.addEventListener('scroll', () => {
125         autoScroll = window.scrollY < 100;
126     });
127
128     function updateLogs() {
129         if (!autoScroll) return;
130
131         fetch("/logs" + (level !== 'ALL' ? `?level=${level}` : ''))
132             .then(res => res.text())
133             .then(html => {
134                 const newTable = document.createElement('div');
135                 newTable.innerHTML = html;
136                 const newContent = newTable.querySelector('table
tbody').innerHTML;
137
138                 const currentContent = logTable.querySelector('table
tbody').innerHTML;
139                 if (newContent !== currentContent) {
140                     logTable.querySelector('table tbody').innerHTML =
newContent;
141
142                     if (autoScroll) {
143                         window.scrollTo({ top: 0, behavior: 'smooth'
});
144                         newMessages.style.display = 'none';
145                     } else {
146                         newMessages.style.display = 'block';
147                     }
148                 }
149             })
150             .catch(err => {
151                 console.error("Error loading logs:", err);
152             });
153     }
154
155     document.getElementById("loadMoreBtn").addEventListener("click",
() => {
156         autoScroll = false;
157
158         fetch(`/logs/more?level=${level}&offset=${offset}`)
159             .then(res => res.text())
160             .then(html => {

```

```
161         const newTable = document.createElement('div');
162         newTable.innerHTML = html;
163         const newRows =
            newTable.querySelector('tbody').innerHTML;
164         logTable.querySelector('tbody').innerHTML += newRows;
165         offset += 50;
166     });
167 });
168
169     setInterval(updateLogs, 5000);
170 </script>
171</body>
172</html>
```

Appendix H.17 – autonomous.html

This page allows users to initiate autonomous cleaning tasks by selecting Panel 1, Panel 2, or both. It displays the current selection and posts the task back to the server. The interface is styled for clarity and user accessibility on various devices.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Select Panel</title>
6      <style>
7          body {
8              margin: 0;
9              font-family: 'Segoe UI', sans-serif;
10             background-color: #f4f4f4;
11         }
12
13         header {
14             background-color: black;
15             text-align: center;
16             padding: 1rem;
17         }
18
19         header img {
20             max-width: 100%;
21             height: auto;
22         }
23
24         .content {
25             max-width: 500px;
26             margin: 3rem auto;
27             background: white;
28             border-radius: 12px;
29             padding: 2rem;
30             box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);
31             text-align: center;
32         }
33
34         h2 {
35             margin-bottom: 1rem;
36             font-size: 1.8rem;
37         }
38
```



```

39     form button {
40         display: block;
41         width: 100%;
42         padding: 1rem;
43         margin: 0.5rem 0;
44         font-size: 1rem;
45         font-weight: bold;
46         border: none;
47         border-radius: 8px;
48         cursor: pointer;
49         transition: all 0.2s ease;
50     }
51
52     .panel1 {
53         background-color: #007bff;
54         color: white;
55     }
56
57     .panel1:hover {
58         background-color: #0056b3;
59     }
60
61     .panel2 {
62         background-color: #28a745;
63         color: white;
64     }
65
66     .panel2:hover {
67         background-color: #1e7e34;
68     }
69
70     .all {
71         background-color: #ff9900;
72         color: white;
73     }
74
75     .all:hover {
76         background-color: #cc7a00;
77     }
78
79     .current-task {
80         margin-top: 1rem;
81         color: #555;
82         font-style: italic;
83     }

```

```

84
85     .home-link {
86         margin-top: 2rem;
87         display: inline-block;
88         color: #333;
89         text-decoration: none;
90         font-weight: bold;
91     }
92
93     .home-link:hover {
94         text-decoration: underline;
95     }
96 </style>
97 </head>
98 <body>
99
100<header>
101    
103</header>
104<div class="content">
105    <h2>Select Cleaning Mode</h2>
106    <form method="POST">
107        <button class="panel1" name="task" value="panel1">🧹 Clean Panel
108            1</button>
109        <button class="panel2" name="task" value="panel2">🧹 Clean Panel
110            2</button>
111        <button class="all" name="task" value="all">🧹 Clean Both
112            Panels</button>
113    </form>
114
115    {% if selected_task %}
116        <div class="current-task">
117            Selected Task: <strong>{{ selected_task|capitalize }}</strong>
118        </div>
119    {% endif %}
120
121    <a class="home-link" href="{ url_for('home') }">← Back to Home</a>
122</div>
123</body>
124</html>

```

Appendix H.18 – log_table.html

This HTML partial renders a styled table for displaying log entries, including ID, timestamp, severity level, and message content. It uses conditional formatting to highlight WARNING and ALERT levels, and is dynamically loaded into the logs dashboard.

```
1 <div style="display: flex; justify-content: center;">
2 <table style="
3     width: 90%;
4     max-width: 1000px;
5     border-collapse: collapse;
6     margin: 1rem auto;
7     font-family: 'Segoe UI', sans-serif;
8     font-size: 14px;
9     background: white;
10    border-radius: 8px;
11    overflow: hidden;
12    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
13    text-align: center;
14 ">
15     <thead style="background-color: #f9f9f9;">
16         <tr>
17             <th style="padding: 12px; border-bottom: 1px solid #ccc;
18 width: 5%;">#</th>
19             <th style="padding: 12px; border-bottom: 1px solid #ccc;
20 width: 20%;">Timestamp</th>
21             <th style="padding: 12px; border-bottom: 1px solid #ccc;
22 width: 15%;">Level</th>
23             <th style="padding: 12px; border-bottom: 1px solid #ccc; text-
24 align: left;">Message</th>
25         </tr>
26     </thead>
27     <tbody>
28         {% for entry in logs %}
29         <tr style="background-color:
30             {% if entry[2] == 'WARNING' %}#fff8dc
31             {% elif entry[2] == 'ALERT' %}#ffe6e6
32             {% else %}#ffffff{% endif %};
33             border-bottom: 1px solid #eee;">
34             <td style="padding: 10px;">{{ entry[0] }}</td>
35             <td style="padding: 10px;">{{ entry[1] }}</td>
36             <td style="padding: 10px; font-weight: bold;">{{ entry[2]
37 }}</td>
```

```
33         <td style="padding: 10px; text-align: left;">{{ entry[3]
    }}</td>
34     </tr>
35     {% endfor %}
36 </tbody>
37 </table>
38 </div>
```

Appendix H.19 – manual.html

This page provides an interface for manual rover control via a virtual joystick and control buttons. Users can initiate movement, trigger cleaning sequences, align with panels, and adjust speed using a slider. It communicates with the backend through AJAX to update rover commands in real time.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Manual Control</title>
6      <link rel="stylesheet" href="{ { url_for('static',
7      filename='style.css') } }">
8      <script src="{ { url_for('static', filename='js/nipplejs.min.js')
9      } }"></script> <!-- Local nipplejs -->
10     <script defer src="{ { url_for('static', filename='js/joystick.js')
11     } }"></script> <!-- Custom joystick logic -->
12     <style>
13         body {
14             margin: 0;
15             font-family: 'Segoe UI', sans-serif;
16             background-color: #f4f4f4;
17         }
18
19         header {
20             background-color: black;
21             text-align: center;
22             padding: 1rem;
23         }
24
25         header img {
26             max-width: 100%;
27             height: auto;
28         }
29
30         .container {
31             max-width: 600px;
32             margin: 2rem auto;
33             text-align: center;
34             background: white;
35             padding: 2rem;
36             border-radius: 12px;
```

```

34         box-shadow: 0 4px 15px rgba(0,0,0,0.1);
35     }
36
37     h2 {
38         margin-bottom: 1rem;
39     }
40
41     .control-btn {
42         margin: 0.5rem;
43         padding: 0.75rem 1.5rem;
44         font-size: 1rem;
45         font-weight: bold;
46         border: none;
47         border-radius: 8px;
48         cursor: pointer;
49         color: white;
50         background: #007bff;
51     }
52
53     .control-btn:hover {
54         background: #0056b3;
55     }
56
57     .slider {
58         margin-top: 2rem;
59     }
60
61     input[type="range"] {
62         width: 100%;
63     }
64
65     .back-link {
66         display: block;
67         margin-top: 2rem;
68         text-decoration: none;
69         color: #555;
70         font-weight: bold;
71     }
72
73     #joystick {
74         margin: 1rem auto;
75         width: 200px;
76         height: 200px;
77         background: #eee;
78         border-radius: 50%;

```

```

79         position: relative;
80     }
81 </style>
82 </head>
83 <body>
84
85 <header>
86     
87 </header>
88
89 <div class="container">
90     <h2>Manual Control</h2>
91
92     <div id="joystick"></div>
93
94     <div>
95         <button class="control-btn" onclick="sendCommand('align1')"> ♀
Align Panel 1</button>
96         <button class="control-btn" onclick="sendCommand('align2')"> ♀
Align Panel 2</button>
97         <button class="control-btn" onclick="sendCommand('seq1')"> ▶
Sequence 1</button>
98         <button class="control-btn" onclick="sendCommand('seq2')"> ▶
Sequence 2</button>
99     </div>
100
101     <div class="slider">
102         <label for="speed">Speed:</label>
103         <input type="range" id="speed" min="10" max="100" value="40"
      onchange="updateSpeed(this.value)">
104         <div id="speedDisplay">40%</div>
105     </div>
106
107     <a class="back-link" href="{ { url_for('mode_selection') }}">← Back to
      Mode Selection</a>
108 </div>
109
110 <script>
111     function sendCommand(cmd) {
112         fetch('/manual/command', {
113             method: 'POST',
114             headers: { 'Content-Type': 'application/json' },
115             body: JSON.stringify({ command: cmd })
116         });

```

```
117     }
118
119     function updateSpeed(value) {
120         document.getElementById('speedDisplay').innerText = value + '%';
121         fetch('/manual/speed', {
122             method: 'POST',
123             headers: { 'Content-Type': 'application/json' },
124             body: JSON.stringify({ speed: value })
125         });
126     }
127</script>
128
129</body>
130</html>
131
```


Appendix H.20 – mode.html

This page allows users to select the rover's operating mode, Autonomous or Manual. Each option triggers a backend update to change the system mode and navigates to the corresponding control interface. The design ensures quick and intuitive mode switching with visual feedback.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Mode Selection</title>
6      <style>
7          body {
8              margin: 0;
9              font-family: 'Segoe UI', sans-serif;
10             background-color: #f4f4f4;
11         }
12
13         header {
14             background-color: black;
15             text-align: center;
16             padding: 1rem;
17         }
18
19         header img {
20             max-width: 100%;
21             height: auto;
22         }
23
24         .content {
25             text-align: center;
26             padding: 3rem 1rem;
27         }
28
29         h1 {
30             font-size: 2rem;
31             margin-bottom: 1rem;
32         }
33
34         .mode-btn {
35             background: #007bff;
36             color: white;
37             padding: 1rem 2rem;
38             margin: 1rem;
```

```

39         font-size: 1.1rem;
40         font-weight: bold;
41         border: none;
42         border-radius: 8px;
43         text-decoration: none;
44         box-shadow: 0 4px 8px rgba(0,0,0,0.1);
45         display: inline-block;
46         transition: all 0.2s ease;
47     }
48
49     .mode-btn:hover {
50         background: #0056b3;
51         transform: translateY(-2px);
52     }
53
54     .manual-btn {
55         background: #28a745;
56     }
57
58     .manual-btn:hover {
59         background: #1e7e34;
60     }
61
62     .back-btn {
63         background: #333;
64     }
65
66     .back-btn:hover {
67         background: #111;
68     }
69 </style>
70 </head>
71 <body>
72
73 <header>
74     
76 </header>
77 <div class="content">
78     <h1>Select Operation Mode</h1>
79     <a href="{{ url_for('autonomous') }}" class="mode-btn"
80     onclick="setMode('autonomous') ">⚙️ Autonomous</a>
81     <a href="{{ url_for('manual_control') }}" class="mode-btn manual-btn"
82     onclick="setMode('manual') ">🎮 Manual</a>

```

```
81     <a href="{{ url_for('home') }}" class="mode-btn back-btn">← Back to
      Home</a>
82 </div>
83 <script>
84 function setMode(mode) {
85     fetch('/set_mode', {
86         method: 'POST',
87         headers: { 'Content-Type': 'application/json' },
88         body: JSON.stringify({ mode: mode })
89     });
90 }
91 </script>
92 </body>
93 </html>
```

Appendix I – IEEE Code of Ethics

The following clauses from the IEEE Code of Ethics were directly applied throughout Project Astraeus.

1. To hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices, and to disclose promptly factors that might endanger the public or the environment.
3. To be honest and realistic in stating claims or estimates based on available data.
5. To improve the understanding of technology, its appropriate application, and potential consequences.
6. To maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations.
7. To seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others.
9. To avoid injuring others, their property, reputation, or employment by false or malicious action.

Group Members



Mark A. Figueroa

- A.A. Engineering
- Pursuing a B.S In Electrical and Computer Engineering Technology (Spring 2026)
- Currently working within the BECET Program as a TA & serving as the Student Chair of the IEEE Student Branch at Valencia College



Pedro J. Cabrera

- A.A. Engineering
- Pursuing a B.S In Electrical and Computer Engineering Technology (Fall 2025)
- Currently working within the BECET Program as a TA & Industrial Cybersecurity Engineer at Siemens Energy.