

L.I.N.K.

Laser Integrated Navigation and Kinematics

Brian Corchuelo¹, Dylan Farnsworth², Santiago Castro³, Jeremy Goodenough¹

CREOL Group #4 – University of Central Florida, Florida, FL 32826 USA

¹PSE, ²EE, ³CPE



Special Thanks to Sponsor:



Abstract

L.I.N.K. is a smart binocular system which enables users to connect their view with a digital map interface, and overlay target data onto a HUD system built into the optical path of the binocular.

Advisors

Dr. Chung Chan & Dr. Paul Leisher

Reviewers

Dr. Qifeng Li, Dr. Yannick Salamin, Dr. Darren Hudson

Contents

1	Executive Summary	23
2	Introduction	23
2.1	Motivation & Background	23
2.2	Review of Prior Related Work	24
2.2.1	Vector X by Vectronix	24
2.2.2	F/A-18 Super Hornet Heads-Up Display	25
2.2.3	Google Maps	25
2.2.4	Garmin Foretrex	26
2.2.5	Strava	26
2.2.6	3D Printed Telescopes	26
2.2.7	MapBox Waypoint API	27
2.3	Goals & Objectives	27
2.3.1	Goals	27
2.3.1.1	Basic Goals	27
2.3.1.2	Advanced Goals	28
2.3.1.3	Stretch Goals	28
2.3.2	Objectives	28
2.3.2.1	Basic Objectives	28
2.3.2.2	Advanced Objectives	29
2.3.2.3	Stretch Objectives	29
2.4	Engineering Components	29
2.5	Project Features and Functionalities	33
2.6	Requirement Specifications	34

2.7	House of Quality	36
2.8	Project Block Diagram	37
2.8.1	Block Diagrams	37
2.9	Flow Charts	40
3	Research and Investigation	42
3.1	Digital Processing and Control	42
3.1.1	Digital Processing Technology Comparison	42
3.1.1.1	Microcontroller (MCU)	42
3.1.1.2	Field Programmable Gate Array (FPGA)	43
3.1.1.3	Single Board Computer (SBC)	43
3.1.1.4	Technology Comparison Summary (Table)	44
3.1.1.5	Final Choice of Digital Processing Technology	44
3.1.2	MCU Part Selection	44
3.1.2.1	ESP32	44
3.1.2.2	STM32	45
3.1.2.3	Nordic nRF52 / RP2040	45
3.1.2.4	MCU Comparison Summary (Table)	46
3.1.2.5	Final Choice of MCU Part	46
3.2	Rangefinding LiDAR	46
3.2.1	Rangefinding Technology Comparison	46
3.2.1.1	Time-of-Flight (ToF) LiDAR	46
3.2.1.2	Phase-Shift (Continuous Wave) LiDAR	47
3.2.1.3	Frequency-Modulated Continuous Wave (FMCW) Li- DAR	47
3.2.1.4	Technology Comparison Summary (Table)	48

3.2.1.5	Final Choice of Rangefinding Technique	48
3.2.2	Rangefinder Component Selection	49
3.2.2.1	PTYS-12X-251028 905nm Module	49
3.2.2.2	PTYS-15X-A260317 905nm Module	49
3.2.2.3	PTF-LD30X 1535nm Module	49
3.2.2.4	Part Comparison Summary (Table)	50
3.2.2.5	Final Choice of Rangefinder Part	50
3.3	Orientation and Navigation Sensors	50
3.3.1	IMU Technology Comparison	50
3.3.1.1	6-DoF IMU	51
3.3.1.2	9-DoF IMU (w/ Magnetometer)	51
3.3.1.3	Comparison Summary (Table)	52
3.3.1.4	Final Choice of IMU Technology	52
3.3.2	IMU Part Selection	52
3.3.2.1	Bosch BNO055	53
3.3.2.2	STMicroelectronics LSM9DS1	53
3.3.2.3	TDK InvenSense ICM-20948	53
3.3.2.4	Part Comparison Summary (Table)	53
3.3.2.5	Final Choice of IMU Part	54
3.3.3	GNSS Technology Comparison	54
3.3.3.1	GNSS Module Types / Accuracy Classes	54
3.3.3.2	Antenna Considerations	55
3.3.3.3	Comparison Summary (Table)	55
3.3.3.4	Final Choice of GNSS Technology	55
3.3.4	GNSS Part Selection	56

3.3.4.1	u-blox 7 Family (NEO-7 Class) Module	56
3.3.4.2	Quectel L76-L (Multi-GNSS) Module	56
3.3.4.3	MediaTek MT3339-Based GPS Modules	57
3.3.4.4	Part Comparison Summary (Table)	57
3.3.4.5	Final Choice of GNSS Part	57
3.3.5	Barometer Technology Comparison	58
3.3.5.1	Barometric Sensing Technologies	58
3.3.5.2	Comparison Summary (Table)	59
3.3.5.3	Final Choice of Barometer Technology	59
3.3.6	Barometer Part Selection	59
3.3.6.1	Bosch BMP280	59
3.3.6.2	BME280	60
3.3.6.3	TE Connectivity MS5611	60
3.3.6.4	Part Comparison Summary (Table)	60
3.3.6.5	Final Choice of Barometer Part	61
3.4	HUD Display and User Input	61
3.4.1	Display Screen Technology Comparison	61
3.4.1.1	OLED (Organic Light Emitting Diode)	62
3.4.1.2	LCD (Liquid Crystal Display)	62
3.4.1.3	Electronic Paper Display	62
3.4.1.4	Multi-Segment LED Displays	63
3.4.1.5	Technology Comparison Summary (Table)	63
3.4.1.6	Final Choice of Display Technology	63
3.4.2	Display Part Selection	64
3.4.2.1	SSD1306	64

3.4.2.2	SH1106	64
3.4.2.3	SSD1327	65
3.4.2.4	Part Comparison Summary (Table)	65
3.4.2.5	Final Choice of Display Part	65
3.4.3	Buttons / Encoders / Indicators	66
3.4.3.1	Input Method Comparison	66
3.4.3.2	Input Method Comparison Summary (Table)	67
3.4.3.3	Final Choice of User Input Method	67
3.4.4	Indicator LED Part Selection	67
3.4.4.1	Single-Color Discrete LED	67
3.4.4.2	RGB LED	68
3.4.4.3	Part Comparison Summary (Table)	68
3.4.4.4	Final Choice of Indicator Part	68
3.5	Power Supply	69
3.5.1	Energy Source Selection	69
3.5.1.1	Wall Outlet	69
3.5.1.2	Batteries	69
3.5.1.3	Power Banks	70
3.5.1.4	Solid-state Batteries	70
3.5.1.5	Energy Source Comparison Summary (Table)	70
3.5.1.6	Final Choice of Energy Source	70
3.5.2	Battery Selection	70
3.5.2.1	Lithium-Ion Cylindrical (18650)	71
3.5.2.2	Alkaline Batteries (Primary Cells)	71
3.5.2.3	Nickel-Metal Hydride (NiMH)	71

3.5.2.4	Lithium Polymer (LiPo)	71
3.5.2.5	Lithium-Ion Prismatic Cells	71
3.5.2.6	Battery Comparison Summary (Table)	72
3.5.2.7	Final Choice of Battery Selection	72
3.5.3	Voltage Regulation Selection	72
3.5.3.1	Linear Regulation	72
3.5.3.2	Switching Regulation	73
3.5.3.3	Voltage Regulation Comparison Summary (Table) . . .	73
3.5.3.4	Final Choice of Voltage Regulation Selection	73
3.5.4	Switching Regulator Selection	73
3.5.4.1	Buck Regulator	74
3.5.4.2	Boost Regulator	74
3.5.4.3	Buck-Boost Regulator	74
3.5.4.4	Switching Regulator Comparison Summary (Table) . .	74
3.5.4.5	Final Choice of Switching Regulators	74
3.5.5	3.3V Regulator Selection	75
3.5.5.1	LTC3113	75
3.5.5.2	TPSM863253	75
3.5.5.3	3.3 V Regulator Comparison Summary (Table)	75
3.5.5.4	Final Choice for 3.3 V Regulator	75
3.6	Communication	76
3.6.1	Wired Communication	76
3.6.1.1	I ² C	76
3.6.1.2	UART	76
3.6.1.3	SPI	76

3.6.1.4	Comparison Summary (Table)	77
3.6.1.5	Final Choice of Wired Communication Architecture . .	77
3.6.2	Wireless Communication	77
3.6.2.1	Bluetooth Low Energy (BLE)	78
3.6.2.2	Wi-Fi	78
3.6.2.3	Comparison Summary (Table)	79
3.6.2.4	Final Choice of Communication Protocol	79
3.7	Embedded Firmware	79
3.7.1	Embedded Programming Environment and Language Comparison	79
3.7.1.1	Arduino Framework with C++	80
3.7.1.2	ESP-IDF with C/C++	80
3.7.1.3	MicroPython	80
3.7.1.4	Comparison Summary (Table)	81
3.7.1.5	Final Choice of Embedded Programming Environment and Language	81
3.7.2	Firmware Architecture Selection	81
3.7.2.1	Cooperative State Machine Architecture	82
3.7.2.2	RTOS-Based Task Architecture	82
3.7.2.3	Comparison Summary(Table)	83
3.7.2.4	Final Choice of Firmware Architecture	83
3.7.3	Sensor Driver Strategy	84
3.7.3.1	Vendor Libraries vs. Custom Register-Level Drivers . .	84
3.7.3.2	Bus Protocol Considerations (I ² C / UART / SPI)	84
3.7.3.3	Final Choice of Driver Strategy	84
3.7.4	Data Packaging and Messaging	85
3.7.4.1	JSON vs. Binary Serialization	85

3.7.4.2	Delivery Strategy: Fire-and-Forget vs. ACK/Retry . . .	85
3.7.4.3	Final Choice of Data Packaging Strategy	86
3.8	Software Stack (Web + App)	86
3.8.1	Development Language Comparison	86
3.8.1.1	JavaScript / TypeScript	86
3.8.1.2	Python	87
3.8.1.3	PHP	87
3.8.1.4	Comparison Summary (Table)	87
3.8.1.5	Final Choice of Development Languages	88
3.8.2	Application Type Comparison	88
3.8.2.1	Traditional Web Application	88
3.8.2.2	Native Mobile Application	88
3.8.2.3	Progressive Web Application (PWA)	89
3.8.2.4	Comparison Summary (Table)	89
3.8.2.5	Final Choice of Application Type	89
3.8.3	Web Stack Technology Comparison	90
3.8.3.1	LAMP Stack	90
3.8.3.2	MERN Stack	90
3.8.3.3	React + Express + SQLite Stack	90
3.8.3.4	Comparison Summary (Table)	91
3.8.3.5	Final Choice of Web Stack	91
3.8.4	Server Hosting	91
3.8.4.1	Cloud-Hosted Server	92
3.8.4.2	Local Self-Hosted Server	92
3.8.4.3	Comparison Summary (Table)	93

	3.8.4.4	Final Choice of Server Architecture	93
3.9		Optical Stack	93
	3.9.1	Optical Stack Overview	93
	3.9.2	Optical Stack Requirements	94
	3.9.3	Telescope Type Selection	94
	3.9.3.1	Galilian Telescope	94
	3.9.3.2	Newtonian Telescope	94
	3.9.3.3	Keplarian Telescope	95
	3.9.3.4	Comparison Summary (Table)	95
	3.9.3.5	Final Choice of Telescope Type	95
	3.9.4	Binocular Type Selection	96
	3.9.4.1	Roof System	96
	3.9.4.2	Porro System	96
	3.9.4.3	Comparison Summary (Table)	96
	3.9.4.4	Final Choice of Binocular Type	97
	3.9.5	Optics Selection	97
	3.9.5.1	Objective Lens System	97
	3.9.5.2	Focus Extender	97
	3.9.5.3	Porro Prism Assembly	97
	3.9.5.4	Field Lens	98
	3.9.5.5	HUD Optical Injection	98
	3.9.5.6	Eyepiece Optics	98
3.10		Implemented Component and Software Revisions	99
	3.10.1	Digital Processing Revision	99
	3.10.2	GNSS Revision	99

3.10.3	Barometric Sensor Revision	100
3.10.4	HUD Display Revision	100
3.10.5	User Input Revision	100
3.10.6	Power-Supply Revision	100
3.10.7	Implemented Internal Communication Map	101
3.10.8	Implemented Firmware Update	101
3.10.9	Implemented Web and Server Stack Update	101
3.10.10	Chapter 3 Revision Summary	102
4	Standards and Design Constraints	102
4.1	Standards	102
4.1.1	Lithium Battery Standards	102
4.1.2	Communication Protocol Standards	103
4.1.2.1	Bluetooth Low Energy (BLE) Standard	103
4.1.2.2	I ² C Bus Standard	104
4.1.3	Software and Data Exchange Standards	104
4.1.4	Laser Classification Standard	105
4.1.4.1	Class 1 Laser	105
4.1.4.2	Class 2 Laser	106
4.1.4.3	Class 3 Laser	106
4.1.4.4	Class 4 Laser	106
4.1.5	Lens Standards	107
4.2	Design Constraints	107
4.2.1	Power Constraints	107
4.2.2	Revised Power Constraints for the Implemented Design	109
4.2.3	Laser Safety Constraints	109

4.2.3.1	Laser Concerns for LiDAR	109
4.2.3.2	Manual Shutter	110
4.2.3.3	Laser On Light	110
4.2.3.4	HUD Caution Label	110
4.2.3.5	Laser Classification Label Sticker	110
4.2.4	Optics Constraints	110
4.2.5	Practical Constraints	111
4.2.5.1	Time	111
4.2.5.2	Economic	112
4.2.5.3	Environmental	112
5	Application of ChatGPT and Other Similar Platforms	113
5.1	Case Study 1	114
5.2	Case Study 2	115
5.3	Case Study 3	117
5.4	Case Study 4	119
6	System Optical Design	122
6.1	Paraxial Design	122
6.1.1	Desired Optical Characteristics	122
6.1.2	Zemax Design, and Baseline Aberrations	122
6.2	Implementation of Off the Shelf Components	123
6.2.1	Full Component List	123
6.2.2	Optical Barrel #1: No Beam Splitter	124
6.2.3	Optical Stack #1: Optimization Results	124
6.2.4	Optical Barrel #2: Beam Splitter	125

6.2.5	Optical Barrel #2: Optimization Results	126
6.3	Integrated H.U.D. System	127
6.4	Conclusions	127
7	Hardware Design	128
7.1	Hardware General Architecture	128
7.2	Electrical Power System	128
7.2.1	Voltage Regulator	128
7.2.2	Power Distribution to Regulators	130
7.3	Sensors and Peripherals	130
7.3.1	USB to Serial Converter	130
7.3.2	Digital Pressure Sensor	131
7.3.3	Piezo Buzzer (Acquire Function)	132
7.3.4	GNSS Receiver Module	133
7.3.5	OLED Display (HUD)	133
7.3.6	Button Interface	134
7.3.7	LiDAR Module	134
7.4	Lower Level Subsystem	135
7.4.1	MCU	135
8	Software Design	136
8.1	System Software Architecture	136
8.2	Firmware Design	137
8.2.1	Firmware Boot and Initialization Sequence	138
8.2.2	Main Loop State Machine	140
8.2.3	Sensor Driver Architecture	141

8.2.4	Waypoint Capture Sequence	142
8.2.5	Local Queue and Retry Logic	142
8.2.6	HUD Display Design	143
8.3	Phone Application Design (PWA)	145
8.3.1	Application Architecture	145
8.3.2	BLE Communication Flow	147
8.3.3	Offline-First Data Pipeline	149
8.3.4	Map View and User Interface	151
8.3.5	PWA Installation and Service Worker	156
8.4	Server Design	157
8.4.1	API Endpoint Design	157
8.4.2	Database Schema	158
8.4.3	Authentication and Security	159
8.5	Data Transfer and Protocol Design	159
8.5.1	BLE GATT Service Structure	159
8.5.2	JSON Message Schemas	161
8.5.2.1	Telemetry Snapshot Schema	161
8.5.2.2	Pin Payload Schema	161
8.5.3	End-to-End Delivery Guarantee	162
8.5.4	Waypoint Coordinate Computation (Haversine Forward Projec- tion)	163
8.5.4.1	Inputs to the Computation	163
8.5.4.2	Closed-Form Destination-Point Formula	164
8.5.4.3	Firmware Implementation Ordering	164
8.5.4.4	Worked Numerical Example	165

8.5.4.5	Error Characteristics and Justification for the Spherical Model	165
8.5.4.6	Optional Target-Altitude Estimation	166
8.5.4.7	Cross-Platform Consistency	166
8.6	Software Components Summary	166
8.7	Integration	167
8.7.1	BLE Integration	168
8.7.2	HTTP Integration	168
8.7.3	Waypoint Computation Consistency	169
8.7.4	Integration Testing Approach	169
8.8	PCB Design	169
8.8.1	Voltage Regulator	170
8.8.2	Power Distribution to Regulators	173
8.8.3	USB to Serial Converter	174
8.8.4	Digital Pressure Sensor	175
8.8.5	Piezo Buzzer (Acquire Function)	176
8.8.6	GNSS Receiver Module	178
8.8.7	OLED Display (HUD)	179
8.8.8	Button Interface	181
8.8.9	LiDAR Module	182
8.8.10	MCU	184
9	System Testing and Evaluation	185
9.1	Testing Overview	185
9.2	Optoelectronics Feasibility Study and Testing	186
9.2.1	Optical Design Validation	186

9.2.2	Magnification	187
9.2.3	True and Apparent F.O.V.	188
9.2.4	Aberration Testing	189
9.2.5	HUD Optical Readability	189
9.2.6	LiDAR Optical Alignment	189
9.3	Hardware Testing	190
9.3.1	ESP32-S3 MCU Bring-Up Testing	190
9.3.2	I ² C Sensor Bus Testing	190
9.3.3	UART Peripheral Testing	191
9.3.4	HUD Display and User Input Testing	191
9.4	Sensor Suite Validation	192
9.4.1	Gyroscope Characterization	192
9.4.2	Barometer Characterization	193
9.4.3	GNSS Receiver Characterization	193
9.4.4	LiDAR Rangefinder Characterization	194
9.5	Software Testing	194
9.5.1	Embedded Firmware State Machine Testing	194
9.5.2	Waypoint Calculation Testing	195
9.5.3	BLE Communication Testing	195
9.5.4	PWA and Server Testing	196
9.5.5	Offline Queue and Synchronization Testing	196
9.6	Accuracy Validation	197
9.6.1	GPS/User Mapping Accuracy	197
9.6.2	Point of Interest/Target Mapping Accuracy	197
9.7	Performance Evaluation	198

9.8	Overall System Integration	198
9.9	Plan for Senior Design II	199
10	Administrative Content	200
10.1	Project Milestones	200
10.2	Bill of Materials	201
10.3	Distribution of Work	204
10.4	Sponsorship	205
11	Conclusion	205
	Appendix	206
A		206
	References	206
B	Component Datasheets	208
C	Software Source Code Listings	209
A.3.1	Firmware: Haversine Forward Projection (<code>waypoint.h</code>) . . .	209
A.3.2	Firmware: Local Pin Queue (<code>pin_queue.cpp</code>)	210
A.3.3	Firmware: BLE GATT Server Setup (<code>ble_server.cpp</code>) . .	210
A.3.4	PWA: Haversine Forward Projection (<code>geo.ts</code>)	211
A.3.5	Server: REST Endpoint and Validation (<code>routes/pins.js</code>) .	212
D	ChatGPT Prompts and Outcomes	213

List of Tables

I	Optical Engineering Components	29
II	Computer Engineering Components	30

III	Electrical Engineering Components	32
IV	Customer Requirements	34
V	System Specifications	35
VI	Pros and Cons for Digital Processing Technology	44
VII	Pros and Cons for Candidate MCU Platforms	46
VIII	Pros and Cons for Different LiDAR Techniques	48
IX	Pros and Cons for LiDAR Module Selection	50
X	Pros and Cons for IMU Technology Selection	52
XI	Pros and Cons for Candidate IMU IC Options (Dev Board Integration)	53
XII	Pros and Cons for GNSS Technology Selection	55
XIII	Pros and Cons for Candidate GNSS Module/Chipset Options	57
XIV	Pros and Cons for Barometer Technology Selection	59
XV	Pros and Cons for Candidate Barometer IC Options	60
XVI	Pros and Cons for HUD Display Technology Selection	63
XVII	Pros and Cons for Candidate OLED Display Controller Families	65
XVIII	Pros and Cons for User Input Method Selection	67
XIX	Pros and Cons for Indicator LED Selection	68
XX	Power Distribution Table for L.I.N.K.	69
XXI	Pros and Cons for Energy Source Selection	70
XXII	Pros and Cons for Battery Selection	72
XXIII	Pros and Cons for Voltage Regulation Selection	73
XXIV	Pros and Cons for Switching Regulator Selection	74
XXV	Pros and Cons for 3.3V Regulator Selection	75
XXVI	Pros and Cons for Wired Communication Interface Selection	77
XXVII	Pros and Cons for Wireless Communication Protocol Selection	79

XXVIII	Pros and Cons for Embedded Programming Environment and Language Selection	81
XXIX	Pros and Cons for Firmware Architecture Selection	83
XXX	Pros and Cons for Development Language Selection	87
XXXI	Pros and Cons for Application Type Selection	89
XXXII	Pros and Cons for Web Stack Technology Selection	91
XXXIII	Pros and Cons for Server Hosting / Deployment Selection	93
XXXIV	Pros and Cons for Telescope Type Selection	95
XXXV	Pros and Cons for Binocular Prism System Selection	96
XXXVI	Worst-Case Current for L.I.N.K.	108
XXXVII	Power Consumption and Runtime Estimation for L.I.N.K.	108
XXXVIII	Optical Component Summary for L.I.N.K. Optical Stack	123
XXXIX	Sensor Data Structures returned by driver read() functions	141
XL	Server API Endpoints	157
XLI	Database Schema: pins Table	158
XLII	BLE GATT Service and Characteristics	160
XLIII	Software Components Summary	166
XLIV	System Testing Verification Matrix	186
XLV	L.I.N.K. Performance Evaluation Summary	198
XLVI	End-to-End Integration Test Sequence	199
XLVII	Senior Design II Testing and Integration Plan	200

List of Figures

1	System Overview	34
2	House of Quality Diagram	37
3	System-Level Hardware Architecture Diagram for L.I.N.K.	38

4	LiDAR Block Diagram	39
5	Optical Stacks Block Diagram	40
6	High Level Software Flow Chart	41
7	Paraxial Design and Simulated Image.	122
8	Paraxial Spot Diagram and Ray Fan Plot.	123
9	Zemax Layout	124
10	No Beam Splitter: Spot Diagrams and Ray Fan Plots at Different Field Angles.124	
11	Simulated Image	125
12	Zemax Layout	126
13	Spot Diagrams and Ray Fan Plots at Different Field Angles.	126
14	Paraxial vs Final Simulated Image.	127
15	H.U.D. Zemax Layout	127
16	LM2596S 3V3 Buck Converter Schematic	129
17	LM2596S 5V Buck Converter Schematic	129
18	Power Architecture Schematic Showing Battery Connection	130
19	USB-A to Serial Converter Schematic	131
20	BMP280 Barometer Schematic	132
21	Buzzer Driver Schematic for SMT-0440-T-R Transducer	132
22	GNSS Module Schematic for u-blox NEO-M9N Receiver	133
23	OLED Display Connector Schematic	134
24	Button Interface Connector Schematic	134
25	LiDAR Module Connector Schematic	135
26	ESP32 Microcontroller Schematic	136
27	Software Architecture Block Diagram	137
28	Software Flowchart (Part 1): On-Device Waypoint Capture and BLE Transmit. 139	

29	Main Loop Timing Diagram	140
30	Queue State Diagram	143
31	HUD Layout Diagram	144
32	Real HUD Layout	144
33	PWA Component Architecture Diagram	146
34	Software Flowchart (Part 2): Phone PWA Processing, Offline Queueing, Server Upload, and ACK Return.	148
35	Offline-First Data Pipeline Flowchart	150
36	Desktop Web Client (Early Development Build) Showing the Original LINK MAP Layout, Recent Pings Panel, and Simulated-Ping Control	151
37	PWA Landing View with BLE Connect Control	153
38	PWA Map View with User, Binocular, and Waypoint Markers	154
39	Waypoint Detail Modal Showing Captured Sensor Context	155
40	BLE Status Panel Showing Pending-Queue Count During Simulated Offline Operation	156
41	BLE GATT Service Structure	160
42	End-to-End Delivery Sequence Diagram	162
43	System Integration Communication Flow	168
44	LM2596S 3V3 Regulator 2D PCB	170
45	LM2596S 3V3 Regulator CAM Preview	171
46	LM2596S 5V Regulator 2D PCB	172
47	LM2596S 5V Regulator CAM Preview	172
48	Power Architecture 2D PCB	173
49	Power Architecture CAM Preview	174
50	USB to Serial Converter 2D PCB	175
51	USB to Serial Converter CAM Preview	175

52	Digital Pressure Sensor and IMU 2D PCB	176
53	Digital Pressure Sensor and IMU CAM Preview	176
54	Piezo Buzzer 2D PCB	177
55	Piezo Buzzer CAM Preview	177
56	GNSS Receiver Module 2D PCB	178
57	GNSS Receiver Module CAM Preview	179
58	OLED Display Connector 2D PCB	180
59	OLED Display Connector CAM Preview	180
60	Button Interface Connector 2D PCB	181
61	Button Interface Connector CAM Preview	182
62	LiDAR Module Connector 2D PCB	183
63	LiDAR Module Connector CAM Preview	183
64	ESP32 Microcontroller 2D PCB	184
65	ESP32 Microcontroller CAM Preview	185
66	Annotated Optics Demonstration	187
67	Beam Expander Diagram	188

1 Executive Summary

L.I.N.K. is a senior design project designed at the University of Central Florida by four undergraduate students specializing in computer, electrical, and optical engineering. The system is designed to measure and record telemetry data and position data of distant objects. The system is based on a custom binocular platform, with a sensor suite and computation center. L.I.N.K. will provide accurate data to the user to map and survey points of interest.

The L.I.N.K. system employs optical binoculars, bore sighted LiDAR, gyroscope module, gnss module, and others to pull real time telemetric data about the system and target. This target is processed by an ESP32, where system position and target parameters are used to calculate the position of some target relative to the user. This target data is then plotted on a companion website via Bluetooth. The user is able to see themselves, and any user marked objects on the map.

L.I.N.K. is intended for field work where mapping distant points is of interest. Whether that be hiking, mapping or surveying. The device is stand alone and battery powered and can be taken along to the field for user operation.

This document outlines the design process of L.I.N.K., including the theoretical foundations, enabling technologies, and rationale behind part selection. It further details system architecture, hardware schematics, and firmware implementations. Standards and constraints guiding the project are discussed, followed by the documentation on fabrication, prototyping, and administrative consideration. Final conclusions and appendices containing technical references and permissions.

2 Introduction

2.1 Motivation & Background

Identifying and marking distant objects is a common task in recreational activities, engineering field work, and many professional applications. Hikers want to be able to identify and remember landmarks they saw from a trail, surveyors and field engineers may need to reference a target within a piece of land for planning layouts, or someone participating in outdoor recreation sports may want an efficient way to measure the distance from a point of interest (POI).

Binoculars are widely used for navigation, surveying, outdoor activities, and observation work, all of which require the user to locate distant targets. In all these situations, the user needs to have two important pieces of telemetry data: (1) an accurate distance measurement from where the user is pointing and (2) the orientation of where the user is looking. The process of acquiring the position of a point of interest and storing that position is defined as “pinging”, while the stored position of the POI being pinged and shown on the mapping interface is defined as a “waypoint”. Handheld rangefinders are currently available on the market, but they only provide a distance measurement. Users would need to use a separate device, such as a phone, to manually

map and store waypoints. This results in the user needing multiple processes, increasing overall usage time and the complexity of the pinging action.

Inspired by the Sheikah Slate Landmark pin systems from *The Legend of Zelda: Breath of the Wild*, our system ties together the viewing quality of binoculars, the precise distance acquisition of light detection and ranging (LiDAR), the information presentability of mobile phones, and mapping accuracy of global positioning systems (GPS) into a single portable device. Combined with a heads-up display (HUD) visualization and wireless connectivity to a phone-side companion application, our system also emphasizes a seamless user experience. The system also contains a pinging function that saves several relevant telemetry measurements from a sensor suite. When the user pings a POI, the system calculates a waypoint using the location, orientation, and distance information from the sensor suite. The waypoint is then wirelessly transmitted to the user's mobile phone, where it is stored locally and later synchronized to a backend server used by the mapping interface when connectivity is available. This allows the user to revisit the same location later without manually writing notes, coordinates, or switching between multiple devices.

2.2 Review of Prior Related Work

In preparation for the project, we researched several products and projects online to better understand ways to achieve our objectives and goals. Shown below are the ones we felt most aligned with our project needs.

2.2.1 Vector X by Vectronix

The most prominent example and inspiration was a product called Vector X by Vectronix. This product details the highest end of the market in terms of the price being approximately \$3,000. The binoculars outline several optical specifications, including a 42mm objective lens diameter, an 8x magnification, and a 7.5° field of view. While these specifications are standard across all price ranges for binoculars, the Vector X excels in the quality of its LiDAR and heads-up display systems. The LiDAR specifications detail a 7,000-meter maximum detection range for large targets, an accuracy of less than 0.5% x distance beyond 500 meters, a greater than 90% detection rate within 2,500 meters, and an operating wavelength of 905 nanometers. [1] The heads-up display has multiple display screens to choose from and changes between displaying information like exact distance, elevation angle, cardinal direction, and battery life. The heads-up display design is also clean and uncluttered in such a way that we will take inspiration when designing how we choose to display the information provided by the sensors in our sensor suite. These specifications are at the peak of what can be found component wise at their price range. We aim to replicate the premium feel of their HUD display and the quality of their LiDAR system for a fraction of the cost to remain affordable for personal and commercial use. Our project differentiates itself via its connectability features including Bluetooth connection to the user's personal device, waypoint pinging and storage, and cloud service.

2.2.2 F/A-18 Super Hornet Heads-Up Display

The heads-up display implemented in L.I.N.K. draws direct inspiration from the classical beamsplitter-based HUD architecture employed in aircraft such as the F/A-18 Hornet. In these systems, data information is output by a display source, collimated through an optical relay, and reflected off a partially reflective combiner so that the projected image appears at optical infinity while remaining superimposed on the real-world scene [2], [3]. This optical configuration eliminates the need for the user to refocus between the external environment and the displayed information, significantly improving situational awareness and response time. Our design adopts this same fundamental principle by integrating a compact beamsplitter within the binocular optical path, enabling a see-through augmented display directly aligned with the user's line of sight. However, several well-documented limitations of beamsplitter HUDs must be addressed in a handheld optical system. First, the partial reflectivity of the combiner inherently reduces scene transmission, which can degrade performance in low-light conditions. This issue can be mitigated through the use of high-efficiency dielectric coatings optimized for the operational spectral band [3]. Second, ghost reflections arising from multiple internal surfaces can introduce secondary images, though this can be reduced by introducing slight wedge angles and carefully controlling coating interfaces. Lastly, traditional HUDs assume a relatively fixed eye position, whereas binocular systems must accommodate a variable eye box. This challenge is addressed by designing for an expanded exit pupil and relaxed alignment tolerances. By adapting the proven collimated projection strategy of fighter aircraft HUDs while addressing these constraints, L.I.N.K. looks to achieve a compact and effective augmented viewing capability suitable for portable rangefinding applications.

2.2.3 Google Maps

The map pinging functionality is conceptually derived from widely adopted digital mapping platforms such as Google Maps, where users can intuitively define and store geographic waypoints through simple interactions like dropping pins. These systems rely on GPS-based coordinate storage, persistent data management, and real-time positional updates to provide a seamless navigation experience [4], [5]. Our system extends this feature by replacing screen-based selection with direct line-of-sight targeting through the binocular optics. Specifically, the user designates a point of interest by visually aligning the system with a target, after which the device computes the corresponding geospatial coordinates using rangefinding data and orientation sensing. This approach transforms passive map interaction into an active spatial measurement process, enabling more precise and contextual waypoint generation. However, this introduces several challenges not present in traditional mapping applications. Errors in angular alignment or range estimation can propagate into significant positional inaccuracies, particularly at long distances [6]. Additionally, unlike smartphone interfaces, the absence of a large interactive display limits direct user feedback; this is addressed by integrating simplified HUD-based indicators to confirm successful waypoint acquisition. By leveraging the intuitive interaction model of modern mapping software while extending it into an optically driven framework, the system provides a novel method for real-world spatial tagging.

2.2.4 Garmin Foretrex

The Bluetooth connectivity framework of this system is informed by rugged, field-oriented navigation devices such as the Garmin Foretrex series, which emphasize reliable, low-power wireless communication in environments where connectivity may be intermittent or constrained. These systems typically prioritize data synchronization over continuous streaming, allowing them to maintain functionality even when connections are temporarily lost. Our design adopts a similar philosophy by implementing Bluetooth as a means of periodic data exchange between the binocular system and external devices such as smartphones or companion modules. This enables the transfer of waypoint data, system status, and configuration parameters without requiring a persistent high-bandwidth connection. A key consideration in this approach is robustness to disconnection events, which are common in outdoor or dynamic use scenarios. To address this, the system incorporates local buffering of all critical data, ensuring that no information is lost during temporary communication interruptions. Power consumption is another critical constraint, as continuous wireless operation can significantly impact battery life. We mitigate this through low-duty-cycle communication protocols and event-driven synchronization [7]. While consumer-grade Bluetooth implementations often assume stable connectivity and frequent user interaction, this system adapts the more resilient strategies used in dedicated navigation hardware to ensure dependable performance in demanding operational conditions.

2.2.5 Strava

The cloud connectivity aspect of the system is inspired by platforms such as Strava, which demonstrate the effectiveness of cloud-based aggregation, storage, and analysis of spatial data collected from devices. In such systems, locally recorded data is periodically synchronized with remote servers, enabling long-term storage, cross-device accessibility, and enhanced data processing capabilities [8]. Our design incorporates a similar architecture, wherein waypoint and measurement data collected by the binocular system are transmitted to a cloud-based service via a paired mobile device. This allows users to access, review, and share collected spatial information across multiple platforms, extending the utility of the device beyond immediate field use. However, unlike continuous tracking applications, this system must accommodate sporadic data generation and potentially limited connectivity. To address this, all data is first stored locally and only uploaded when a stable connection becomes available, ensuring reliability in remote environments. Security and data integrity are also important considerations, particularly when handling location-based information. [8]. By adopting the scalable and persistent data management strategies demonstrated by modern cloud platforms while tailoring them to the constraints of a field-deployed optical system, the design enables seamless integration between physical measurements and digital data ecosystems.

2.2.6 3D Printed Telescopes

Another relevant project we referenced was a 3D printed telescope project in which a Galilean telescope with a 4x magnification was designed and built. The shortcomings of the cheaper and simpler Galilean configuration meant that apparent field of view was extremely low due to the nature of the concave eyepiece lens pulling in light from a much wider cone. Knowing this drawback incentivized the use of a Keplerian setup which allows for a significantly better apparent

field of view with the drawback of having an inverted image at the output. This drawback can be accounted for by using a pair of porro prisms that re-invert the image to the correct orientation. Furthermore, the telescope computer aided design (CAD) model contained a parameterized 3D design for the lens mounts which allowed for simple changes to be made to lens diameter and thickness without having to redesign a new 3D model with each change. This functionality will be exceptionally useful for the optical packaging of the L.I.N.K. device as tolerances affecting the fitting of the lens can easily be adjusted to improve the alignment and security of the fitted optical components.

2.2.7 MapBox Waypoint API

In addition, our group has some prior software experience that is relevant to this project's waypoint and mapping system. In a previous project for the COP 4331C class at UCF, one of our teammates helped implement a map-based interface using the MapBox Application Programming Interface (API) [9] that also supports placing waypoints on an interactive map. This prior project is applicable to the mapping system, because the binoculars require a way to visualize and store waypoints in an easy-to-understand format. Having already built a MapBox based waypoint system reduces the implementation risk for the website/application portion of the project and allows more emphasis on the embedded firmware side of things.

2.3 Goals & Objectives

Below are the goals and objectives we established for the L.I.N.K. device. The goals define the broader purpose and vision of the project while the objectives define specific and measurable deliverables that will outline the development and implementation of subsystems into the device. There are stretch goals and objectives that provide insight on how long term development of the product would occur, but is outside of the scope for this project.

2.3.1 Goals

2.3.1.1 Basic Goals

- Design a binocular optical design system that allows the user to clearly identify distant targets in various environmental conditions.
- Implement a LiDAR system that accurately measures the distance between the user and a desired target.
- Integrate a heads-up display system that provides the user with data acquired from various sensors in a sensor suite while looking through the binoculars.
- Measure latitude, longitude, altitude, and bearing using sensors integrated in a custom printed circuit board.
- Develop an end-to-end pinging system that stores and visualizes waypoints on a map within an application or website.
- Provide a custom PCB integrated with a rechargeable power system, a sensor suite, and a microcontroller.

2.3.1.2 Advanced Goals

- Implement an adjustable focusing subsystem that allows the user to control the focus of the binocular system based on the distance between the user and the desired target.
- Support capturing waypoints even during temporary internet or Bluetooth interruptions by locally storing pings on the phone and automatically synchronizing them to the backend server and map interface once connectivity is restored, without duplicating waypoints.
- Provide a multi-page heads-up display interface that supports multiple possible display pages so that the user can cycle between desired data visuals using on-device controls.

2.3.1.3 Stretch Goals

- Design a variable magnification system in which the user can manually move between different magnification levels.
- Expand mapping interface into a multi-user map which supports sharing waypoints with other users, so multiple devices can view the same pins in a shared map workspace.
- Implement a HUD mini map that displays a map view in the HUD that shows the user's position, heading, and direction of nearby waypoints.

2.3.2 Objectives

2.3.2.1 Basic Objectives

- Utilize multiple lens elements and prisms to produce an afocal binocular system delivering an upright image to the user with a $>5\times$ magnification and an apparent field of view of 22.5 degrees, while achieving sufficient image quality to resolve standard targets at 100 meters away.
- Integrate and align a LiDAR system capable of measuring target distance out to at least 100 meters with an accuracy of ± 5 meters, while maintaining less than 1° angular deviation from the optical axis of the binocular optical system.
- Mount and align a beamsplitter to the optical axis of the binoculars and an OLED display so measurement and status information is visible through the binoculars without significantly obstructing the target view. Combine with a HUD layout that displays core sensor data including range reading, cardinal direction, and ping confirmation with readable text or icons during normal operation.
- Implement sensor drivers for the LiDAR, GNSS, IMU, and barometer integrated with the custom PCB, and develop a telemetry routine that captures latitude, longitude, altitude, bearing/heading, range, and timestamp in a single measurement event.
- Define a waypoint payload schema and wirelessly transmit it to a phone or web interface using Bluetooth such that each ping event generates a new waypoint. Display each waypoint on a map interface with correct position data and store it locally or in a database, maintaining a record including timestamp, range, position, and heading for user review.
- Design, fabricate, and validate a custom PCB that integrates the microcontroller, sensor suite, HUD interface, and a rechargeable battery power system with voltage regulation sized for the peak current draw of all connected subsystems. Verify stable operation through interface checks, grounding review, and power/signal integrity testing.

2.3.2.2 Advanced Objectives

- Implement a manually adjustable eyepiece lens that allows the user to achieve clear focus of targets over a distance range of 5 meters to infinity, verified by maintaining acceptable image quality across that range.
- Support offline-first behavior by storing waypoint payloads locally during internet or BLE interruptions and automatically synchronizing all stored waypoints to the backend once connectivity is restored without data loss. Validate operation under normal use, no internet, temporary BLE disconnect, and invalid sensor conditions. Prevent duplicate waypoints using unique ping identifiers to ensure each ping is recorded exactly once.
- Implement a multi-page HUD interface with at least three distinct display pages displaying different sensor data, and enable user-controlled page cycling via on-device inputs such that page transitions occur reliably and the correct data is displayed on each page during operation.

2.3.2.3 Stretch Objectives

- Implement a variable magnification optical system that allows the user to manually switch between at least two distinct magnification levels, with each state verified by measured magnification and maintaining acceptable image quality at each setting.
- Implement a multi-user map system where multiple devices can connect to a shared workspace such that waypoints created by any user are transmitted, stored, and displayed consistently across multiple users and devices without duplication or data loss.
- Implement a HUD mini map that renders the user's position and heading along with nearby waypoints, such that waypoint directions update correctly with changes in user orientation and position using a live feedback communication between the device and the user's personal device during system operation.

2.4 Engineering Components

Below are the hardware components required for each engineering discipline involved in the device and the relevant specifications of each.

TABLE I: Optical Engineering Components

Discipline	Component	Parameter	Specification	Unit
Optics	Objective Lens Edmund Optics #32-923	A) Focal Length B) Diameter C) Coating D) Material	A) +160 B) 40 C) MgF2 D) N-BK7/N-SF5	A) mm B) mm C) - D) -
Optics	Focus Extender Lens Edmund Optics #45-027	A) Focal Length B) Diameter C) Coating D) Material	A) -100 B) 25 C) MgF2 D) N-BK7	A) mm B) mm C) - D) -

Continued on next page

TABLE I – continued from previous page

Discipline	Component	Parameter	Specification	Unit
Optics	Field Lens Edmund Optics #45-128	A) Focal Length B) Diameter C) Coating D) Material	A) +60 B) 25 C) MgF2 D) N-BK7	A) mm B) mm C) - D) -
Optics	Eyepiece Lens Edmund Optics #32-625	A) Focal Length B) Diameter C) Coating D) Material	A) +50 B) 25 C) MgF2 D) N-BK7	A) mm B) mm C) - D) -
Optics	Beamsplitter Edmund Optics #32-505	A) Construction B) Dimension C) Coating D) Material	A) Cube B) 25x25x25 C) MgF2 D) N-BK7	A) - B) mm C) - D) -
Optics	Porro Prism	A) Construction B) Dimension C) Coating D) Material	A) Right-Angle B) 2x1x1 C) None D) N-BK7	A) - B) in C) - D) -
Optics	LiDAR Module Chengdu JRT PTY5-12X	A) Wavelength B) Dimension C) Range D) Accuracy E) Data Frequency	A) 905 B) 22.5x16x9.5 C) 1200 D) ± 2 E) 1-4	A) nm B) mm C) m D) m E) Hz

TABLE II: Computer Engineering Components

Discipline	Component	Parameter	Specification	Unit
Computer	ESP32-S3 DevKitC-1	A) Processor Frequency B) Processor Cores C) Flash Memory D) PSRAM E) Wireless Communication F) Logic Voltage G) Power Input	A) 240 B) 2 C) 32 D) 16 E) Wi-Fi / BLE F) 3.3 G) 5	A) MHz B) count C) MB D) MB E) - F) V G) V

Continued on next page

TABLE II – continued from previous page

Discipline	Component	Parameter	Specification	Unit
Computer	BNO055 9-DoF IMU Adafruit Breakout	A) Degrees of Freedom B) Sensor Types C) Sensor Fusion D) Communication Interface E) Orientation Update Rate F) Operating Voltage	A) 9 B) Accel/Gyro/Mag C) Integrated D) I ² C E) 100 F) 3.3	A) count B) - C) - D) - E) Hz F) V
Computer	GT-U7 GNSS Receiver u-blox NEO-7M Class	A) Navigation System B) Communication Interface C) Default Baud Rate D) Position Update Rate E) Supply Voltage F) Antenna Type	A) GPS/GNSS B) UART C) 9600 D) 1–10 E) 3.3 F) External Patch	A) - B) - C) bps D) Hz E) V F) -
Computer	BMP280 Digital Pressure Sensor	A) Measured Quantity B) Pressure Range C) Communication Interface D) Supply Voltage E) Pull-up Resistance F) Decoupling Capacitance	A) Pressure/Altitude B) 300–1100 C) I ² C D) 3.3 E) 4.7 F) 100	A) - B) hPa C) - D) V E) kΩ F) nF
Computer	SSD1306 OLED HUD Display Adafruit OLED Module	A) Display Type B) Resolution C) Color Output D) Communication Interface E) Supply Voltage F) Use Case	A) OLED B) 128x64 C) Monochrome D) I ² C E) 3.3 F) HUD Telemetry	A) - B) pixels C) - D) - E) V F) -
Computer	Rotary Encoder with Pushbutton	A) Input Type B) Output Signals C) GPIO Lines Used D) Logic Voltage E) User Function	A) Incremental B) A/B/SW C) 3 D) 3.3 E) HUD Navigation/Acquire	A) - B) - C) count D) V E) -

TABLE III: Electrical Engineering Components

Discipline	Component	Parameter	Specification	Unit
Electrical	18650 Lithium-Ion Cell	A) Cell Chemistry B) Nominal Voltage C) Discharge Voltage Range D) Cell Configuration E) Charging Method F) System Role	A) Li-ion Cylindrical B) 3.7 C) 3.0–4.2 D) 1S E) External Charger F) Main Battery	A) - B) V C) V D) count E) - F) -
Electrical	LTC3113 Buck-Boost Regulator 3.3 V Rail	A) Regulator Type B) Input Voltage Range C) Output Voltage D) Typical Efficiency E) Inductor Value F) Output Rail Use	A) Buck-Boost B) 1.8–5.5 C) 3.3 D) Up to 96 E) 2.2 F) MCU/Sensors/HUD	A) - B) V C) V D) % E) μ H F) -
Electrical	LTC3539 Boost Regulator 5 V Rail	A) Regulator Type B) Input Voltage Range C) Output Voltage D) Peak Switch Current E) Inductor Value F) Output Rail Use	A) Boost B) 0.7–5.5 C) 5.0 D) 2.6 E) 2.2 F) LiDAR/5 V Loads	A) - B) V C) V D) A E) μ H F) -
Electrical	SMT-0440-T-R Piezo Buzzer Acquire Function	A) Supply Rail B) Driver Type C) Control Signal D) Base Resistor E) Noise Capacitor F) User Feedback	A) 3.3 B) NPN Low-Side Switch C) PWM D) 1 E) 0.1 F) Ping Queued	A) V B) - C) - D) $k\Omega$ E) μ F F) -
Electrical	Single-Color Discrete LEDs Power/Laser Indicators	A) LED Colors B) Indicator Functions C) Drive Method D) Typical Forward Current E) Current-Limiting Method	A) Green/Red B) Power/Laser On C) GPIO or Switch State D) 20 E) Series Resistor	A) - B) - C) - D) mA E) -

Continued on next page

TABLE III – continued from previous page

Discipline	Component	Parameter	Specification	Unit
Electrical	Power Switch and Regulator Headers	A) Input Source B) Input Voltage Range C) Switched Output D) Connected Regulators E) Rail Outputs	A) 18650 Battery B) 3.0–4.2 C) Battery Bus D) LTC3113/LTC3539 E) 3.3, 5.0	A) - B) V C) - D) - E) V

2.5 Project Features and Functionalities

The L.I.N.K device will feature several systems including a binocular system, an integrated HUD, a sensor suite, an on-board computing module, and an off-board connected device displaying relevant geolocational data on a mapping interface. The sensor suite contains a LiDAR module, a GNSS receiver, an IMU, and a barometer. The L.I.N.K. device will also feature a Power Supply Unit (PSU). The PSU contains three 3.7V (18650) lithium-ion battery, a 3.3V regulator, and a 5V regulator. These regulators power the sensor suite and other peripherals connected to the ESP32 [10]. The battery will be rechargeable and will use a commercial battery charger

A key functionality of the L.I.N.K. device is the ability to acquire waypoints and store them on a personal device. The waypoint acquisition sequence starts with the user powering on the L.I.N.K. device and confirming proper connection with the companion application. Once power and connection are established, the user will seek a POI. Once a desired POI is in view, the user will turn on the LiDAR system using a separate button to begin producing distance measurements. The user will then press an “acquire” button to begin the pinging sequence. The pinging sequence begins by recording a suite of data including POI distance, device orientation, pressure readings, and GPS location. Following the pinging sequence, the acquired data is parsed and averaged on the ESP32 computer system. The processed data is then used to calculate and generate a waypoint, which is wirelessly transferred to the user’s mobile device using a Bluetooth connection. The mobile device stores the waypoint locally and synchronizes it to the backend server used by the mapping interface when connectivity is available. The map interface will then show the user’s location and display the pinged waypoint containing relevant information such as altitude and coordinates.

The integrated HUD system constantly pulls data from the sensor suite and live updates on the display to show relevant data such as range, direction, altitude, and whether the acquisition sequence was able to successfully ping a waypoint. In cases where specific sensors are unable to define a specific measurement, the HUD will include an indicator of which sensors are failing. A manual brightness control system will also be included with the HUD display in which a fixed linear polarizer and a rotatable linear polarizer in series allow the user to define the HUD brightness based on the viewing conditions of the binocular system. The polarizer setup also reduces complexity on the software and hardware design while also being fully analog.

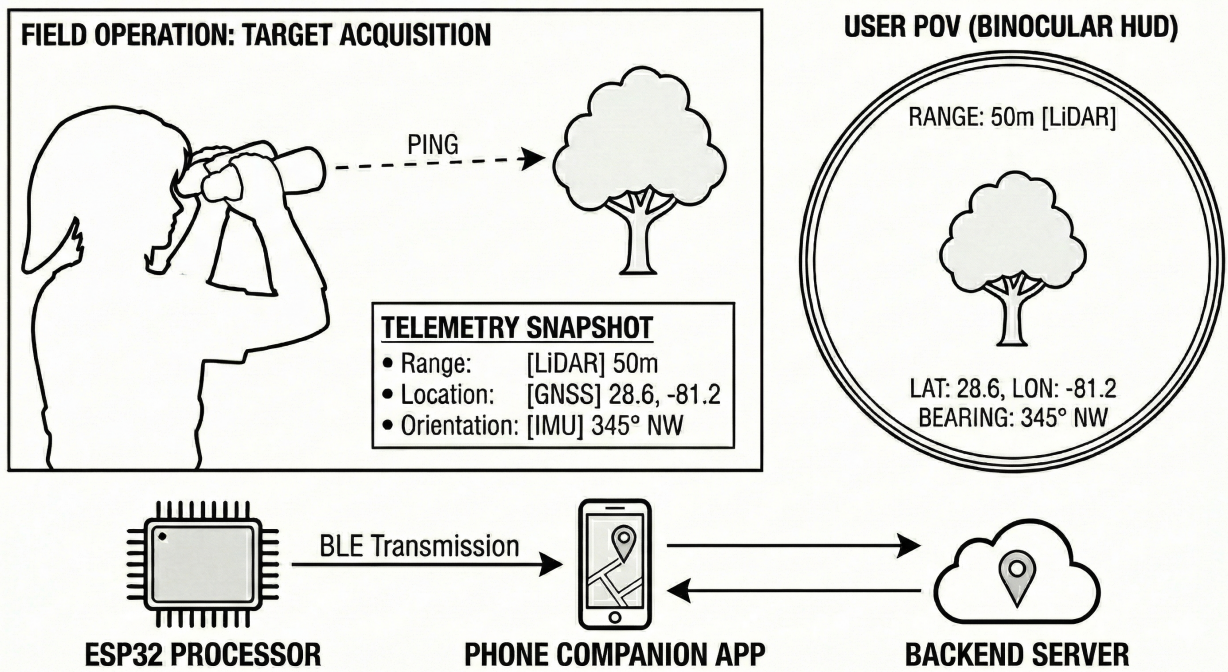


Fig. 1: System Overview

2.6 Requirement Specifications

TABLE IV: Customer Requirements

Category	Requirement
Affordability	The device will be competitively priced compared to similar devices on the market.
Ease of Use	The device will have simple operating procedures that are repeatable and easy to register.
HUD Quality	The device will contain a HUD system displaying range and bearing clearly without obstructing the binocular field of view.
Portability	The system must be easy to physically hold and store without external support.
Versatility	The device will operate in varied environmental conditions and ping different POIs including buildings, hills, and distant terrain.
Durability	The device can be stored, used, and transported without debilitating damage from standard travel and transport.

Continued on next page

TABLE IV – continued from previous page

Category	Requirement
Reliability	The device will provide concise, accurate, and low-error data consistently over many operating cycles.
Battery Life	The device will operate continuously for a minimum of 1 hour under normal electrical demand conditions.

TABLE V: System Specifications

Parameter	Value	Description
Optical Stacks		
Magnification	5×	Minimum angular magnification required for target observation.
Apparent Field-of-View	20°	Angular field visible to the user through the binocular optical system.
Entrance Pupil Diameter	40 mm	Objective aperture size defining the light collection capability.
Eye Relief	20 mm	Distance from the final eyepiece surface to the eye position.
Boresight Error	< 1°	Alignment deviation between the LiDAR axis and binocular optical axis.
H.U.D. Latency	≤ 50 ms	From pinging a target, data shall appear in the H.U.D. in 50 ms or less.
Sensor Suite		
Maximum Tested Range	200 m	Maximum experimentally verified ranging distance.
Operating Range	≤ 200 m	The LiDAR system shall measure any notable target within 200 meters.
Range Accuracy	±2 m	Measured range uncertainty under test conditions.
Mapping Error	±10 m	When considering the error from the LiDAR, IMU, and data processing, the targeted ping shall not exceed a 10 meter error radius from the true target.
Website Latency	≤ 1.5 s	Time from ping command to companion website ping indicator display shall not exceed 1.5 seconds.

Continued on next page

TABLE V – continued from previous page

Parameter	Value	Description
Packaging		
Size	200 mm × 200 mm × 100 mm	Maximum allowable package envelope for device portability.
System Weight	< 1500 g	Target total device mass shall not exceed 1500 grams.
Power Draw	≤ 3.32 W	Device operating at full power shall not draw more than 3.32 W.

In Table I, the Customer Requirements are listed and specified. These serve as goals for us to hit, that will be marketable for a potential consumer/ buyer. Next are Table II and III. These are divided into two categories: System Specifications and Component Specifications. System Specifications represent the primary performance characteristics and key marketing points of the L.I.N.K. device from a user perspective. Component Specifications define the individual engineering requirements that collectively enable the overall system to function as designed and are primarily used to guide the engineering design process.

The three most critical design specifications identified are Minimum Working Range, Mapping Error, and Range Accuracy. These specifications are classified as System Specifications, as they are indicative of the performance of the entire system rather than any single component. For the L.I.N.K. device to be a viable and practical product capable of fulfilling the objectives of this project, these requirements must be met. To verify compliance with these specifications, significant testing will be conducted using equipment such as calibrated distance targets for range validation, timing and logging tools for latency measurement, and controlled power-draw with runtime tests for battery life evaluation.

2.7 House of Quality

The House of Quality is used to translate user needs into quantifiable engineering design priorities for the L.I.N.K. device. As shown in Figure 2, the strongest correlations between customer requirements and engineering characteristics are cost, field of view, magnification, and dimensions. The House of Quality indicates that affordability, high HUD quality, and strong portability must be prioritized to create a viable consumer product.

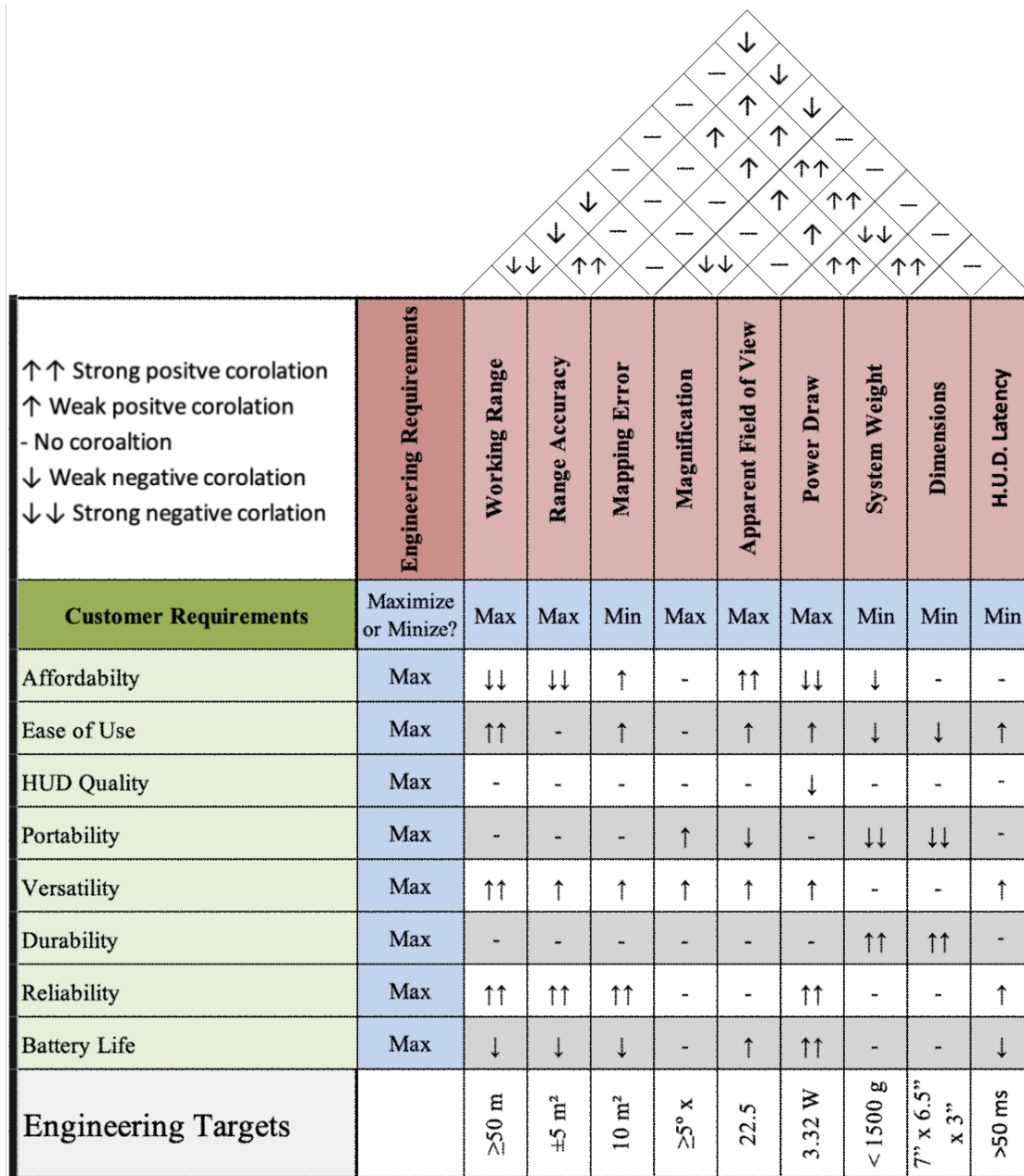


Fig. 2: House of Quality Diagram

2.8 Project Block Diagram

2.8.1 Block Diagrams

The project block diagrams summarize the architecture for several systems within the L.I.N.K. device. The user observes a distant target through the Optical Stacks while also being able to view important sensor information using an integrated HUD. The sensor data provided by the

sensor suite is sent by the MCU to the OLED display in which the image from the display transmits through the beamsplitter where the image is then sent down the optical path towards the user. Incoming light from the objective lens and HUD pass through a porro prism setup that corrects the image inversion caused by the Keplerian setup of the binocular system. The porro prism also serves to reduce the packing size by decreasing the physical distance required between the objective and eyepiece lenses to maintain an afocal system. The LiDAR system measures distance by sending out pulses of 905 nm light in which a time-of-flight sensor and system timer are used to compute a distance from the reflected pulses that output a signal when they hit an avalanche diode. The hardware is split into two PCBs. PCB #1 serves as the power and charging board containing the rechargeable battery interface, charging IC, and regulators that will generate stable power for the system. PCB #2 contains the ESP32 and the sensor suite (IMU, barometer, GNSS) and serves as the central controller for measurements, processing data, driving the HUD, and managing wireless communication with the phone-side companion application and backend mapping system. This separates noise and high-power loads from the sensitive sensing and logic sections and should simplify debugging while creating a clean path from prototyping to a full custom PCB.

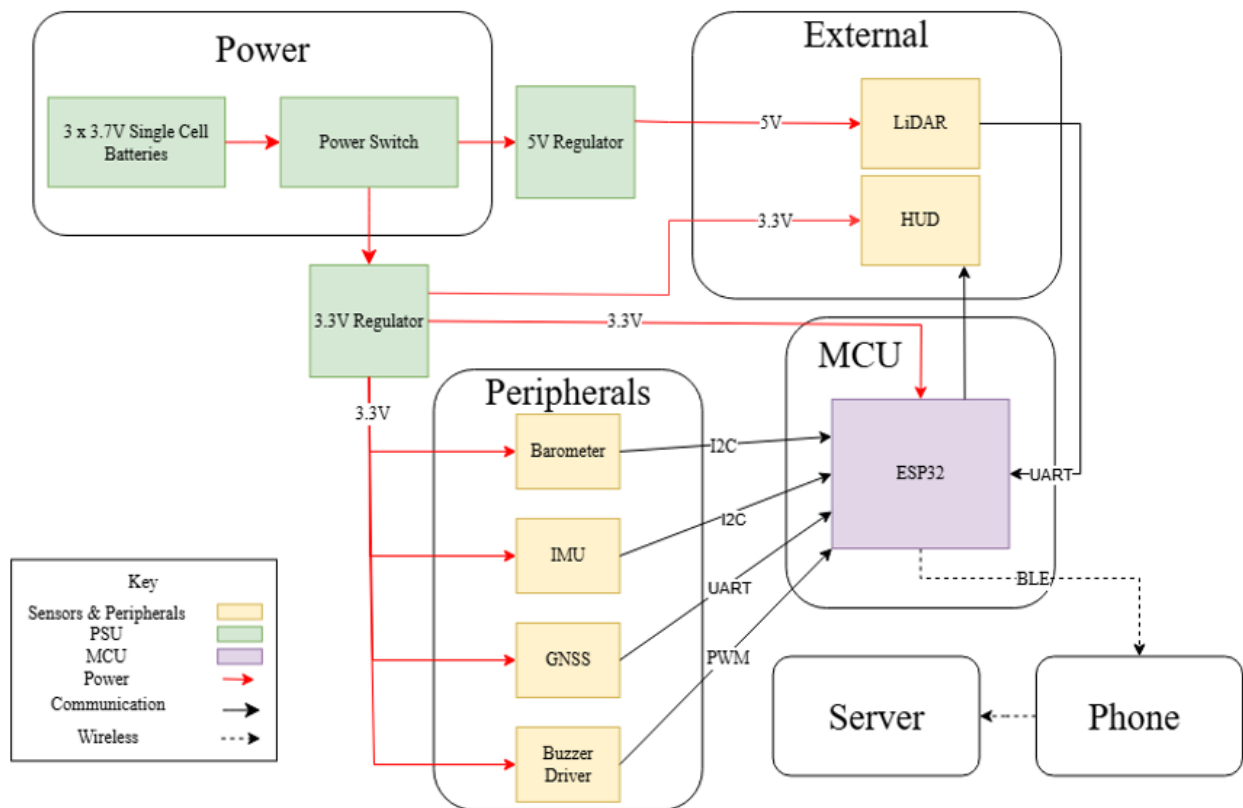


Fig. 3: System-Level Hardware Architecture Diagram for L.I.N.K.

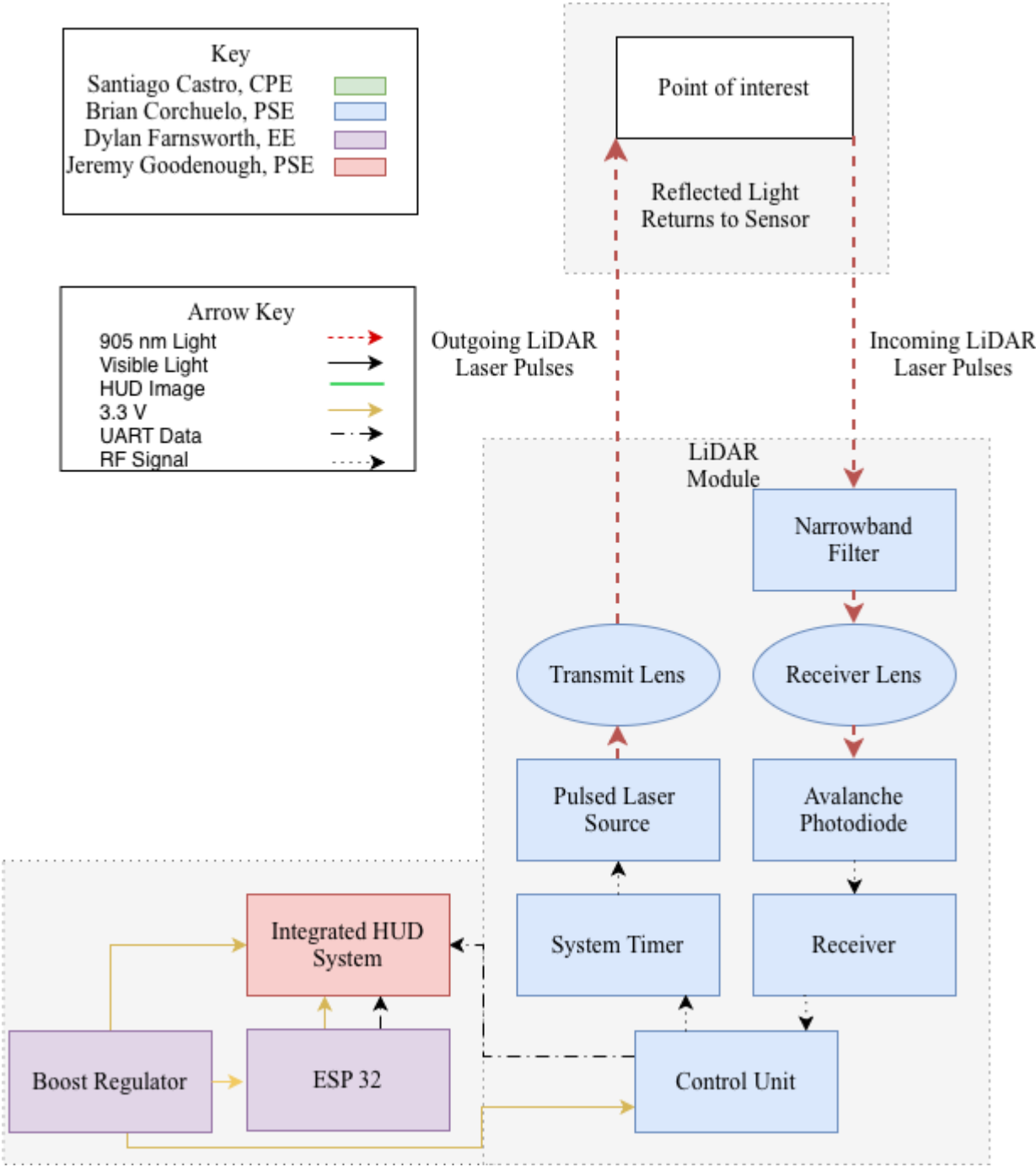


Fig. 4: LiDAR Block Diagram

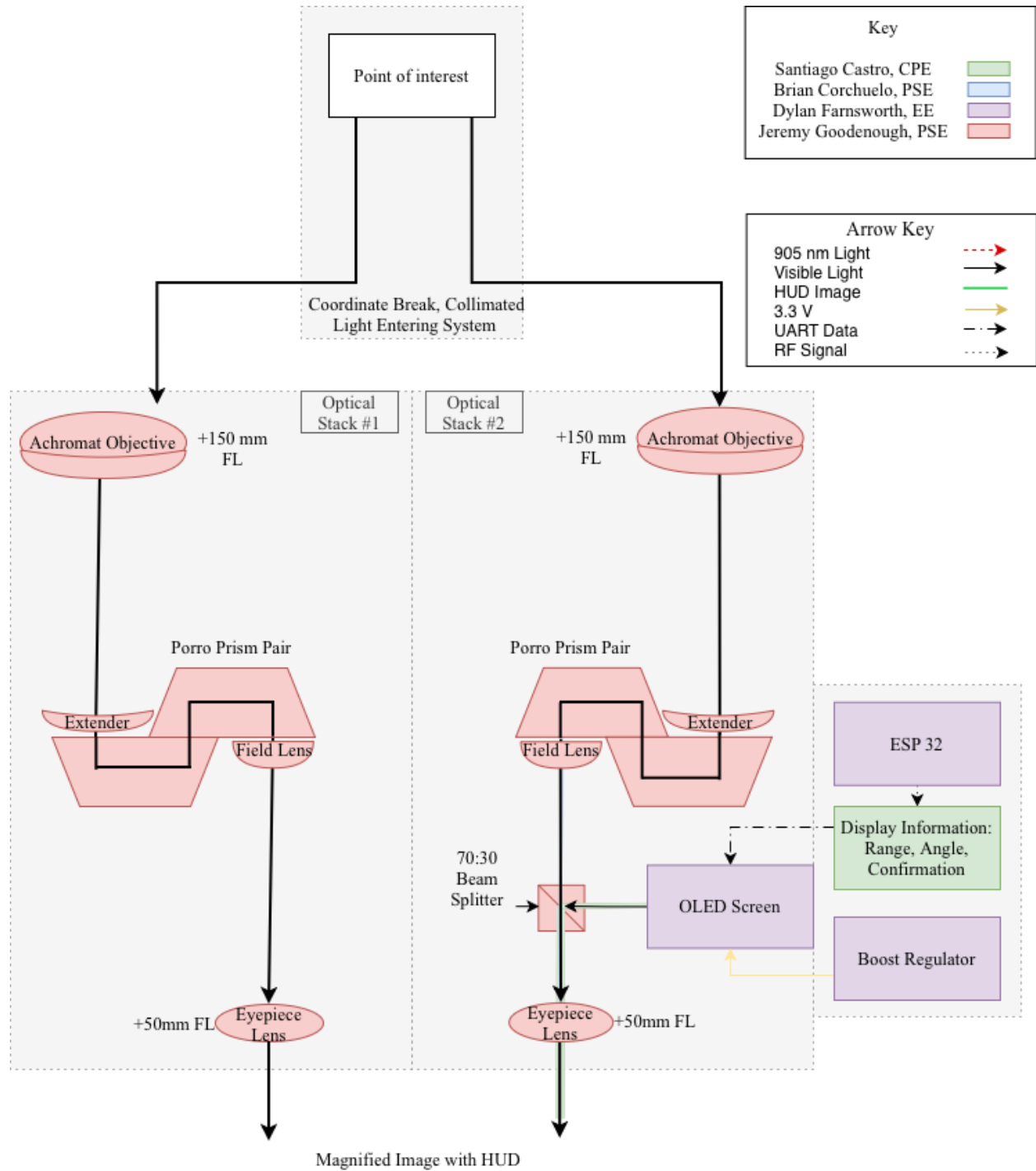


Fig. 5: Optical Stacks Block Diagram

2.9 Flow Charts

The software flowchart describes the complete waypoint workflow across all different aspects. When the device gets powered on the ESP32, they boot and initiate drivers and peripherals

(IMU, GNSS, barometer, LiDAR). Those peripherals then run self-tests to ensure they are up to order, and the ESP32 advertises a Generic Attribute Profile (GATT) in Bluetooth Low Energy (BLE) [11] service to make sure it can transmit a Bluetooth connection. If a module (sensor, HUD, etc.) fails, the device will degrade by disabling the failed module and warning the user via the HUD. When the user aims and presses the ping button, the device will capture a snapshot of the telemetry data, validate the measurements, and then compute an estimated location of the waypoint. This data is then packaged into a JavaScript Object Notation (JSON) [12] payload. If the phone is not connected or an acknowledgement (ACK) is not received, the payload can be locally queued on the device for a retry later. On the phone side, a Progressive Web App (PWA) [13], [14] maintains the BLE session, receives the payload, stores it locally, and attempts to send it to the backend server through an HTTP endpoint [15]. If internet connectivity is unavailable, the phone retains the payload locally and retries synchronization later. The server then validates the payload schema and bounds, stores the pin in the database, and responds with success. Finally, the phone updates the map UI to render the new waypoint and sends an ACK back to the ESP32 over BLE so that the device can mark the pin as delivered and remove it from its local queue, closing the loop with reliable delivery.

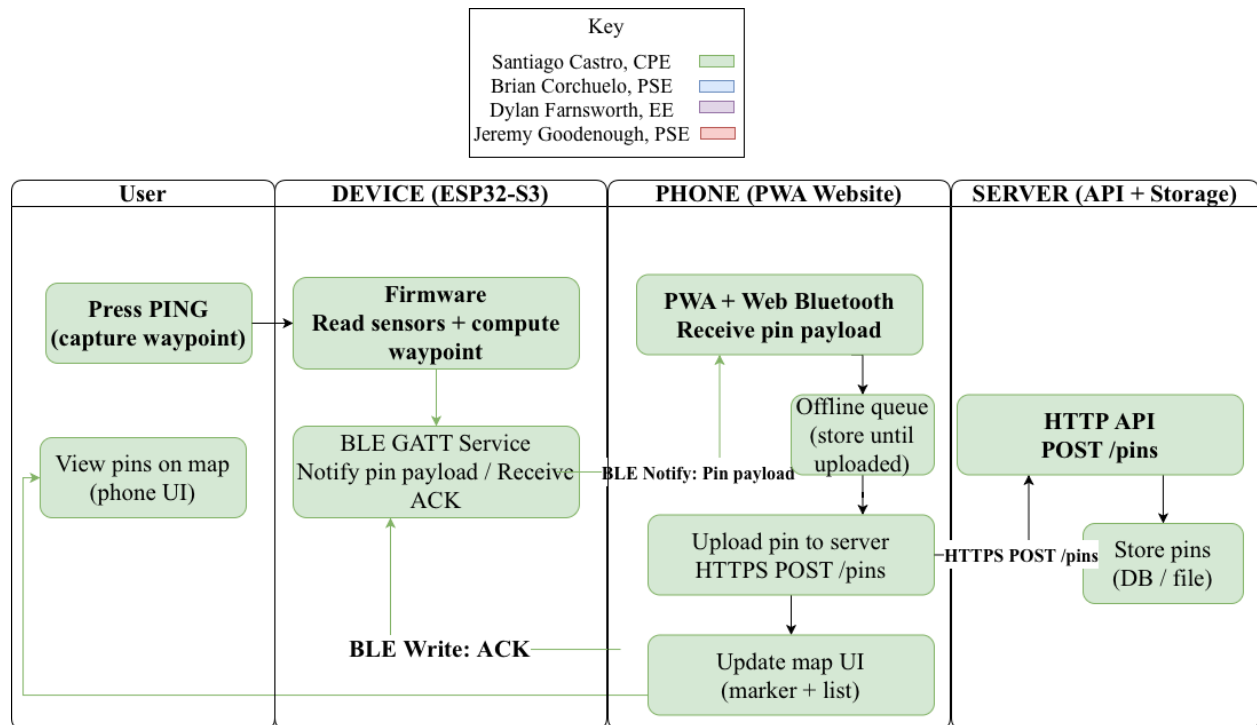


Fig. 6: High Level Software Flow Chart

3 Research and Investigation

3.1 Digital Processing and Control

The Digital Processing Subsystem (DPS) is responsible for the entire L.I.N.K. system workflow: sensor sampling, basic filtering, user input handling, HUD updates, and packaging telemetry into a message format suitable for a wireless transfer. Because the waypoint is captured in a "snapshot", the capture event must feel responsive to the user, and the processing platform must support seamless peripheral integration with a platform designed for uninterrupted timing accuracy.

In addition to timing, the processor must support multiple sensors using common communication protocols such as Inter-Integrated Circuit (I²C), Serial Peripheral Interface (SPI), and Universal Asynchronous Receiver-Transmitter (UART), and must also support wireless connectivity. These requirements narrow the selection to platforms that provide sufficient computational headroom, full peripheral support, and reliable software support.

3.1.1 Digital Processing Technology Comparison

Digital processing can be implemented using several types of hardware, each with different trade-offs, power consumption, complexity, and integration effort. The three most relevant categories are Microcontroller Units (MCUs), Field Programmable Gate Arrays (FPGAs), and Single Board Computers (SBCs). For L.I.N.K., the primary factors in the decision-making will come down to:

- Predictable real-time response for sensor reading and HUD updates
- Ability to interface with a sensor suit using standard communication protocols
- Integrated wireless support
- Low power consumption for portable use
- A development workflow that supports rapid changes during prototyping

3.1.1.1 Microcontroller (MCU)

A Microcontroller Unit (MCU) is designed for embedded control tasks, typically combining a microprocessor, memory (Flash (ROM) and SRAM (RAM)), and peripherals (GPIO, ADC, Timers/PWM, communication interfaces) on a single chip. MCUs are widely used in sensor-driven systems. This is because they can directly interface with I²C/SPI/UART peripherals, reliably handle interrupts, and provide deterministic execution timing. This determinism is valuable because the device must regularly poll sensors, refresh the HUD, and execute a consistent, reliable waypoint routine without interruption.

The main strengths of the MCU are its ease of integration and rapid development. The use of a proven toolchain, an abundance of code and libraries, and the availability of data all help lower the design risk of a multi-sensor design. In addition, MCUs typically operate within a power range suitable for portable devices, thereby helping preserve battery life and addressing thermal concerns. The main drawback of MCUs is that their performance is lower than that

of more complex platforms; however, many embedded applications do not require such high computational performance and instead benefit more from reliable timing and stable peripheral handling.

3.1.1.2 Field Programmable Gate Array (FPGA)

A Field-Programmable Gate Array (FPGA) is a reconfigurable digital logic device that implements custom hardware circuits defined in a Hardware Description Language (HDL) such as Verilog or VHDL. Because logic can be implemented in parallel, FPGAs excel in applications that demand extremely high throughput, custom timing, or simultaneous execution of multiple operations without sharing a single processor pipeline. In other domains, FPGAs are often chosen when a project requires extremely low latency or highly parallel signal processing.

Despite all the advantages, FPGAs still incur development and integration costs. HDL development requires careful verification and debugging, which can be quite demanding and require a more specialized workflow than firmware development on an MCU. Power consumption can also be much higher, which is important for portable designs. Additionally, connecting an FPGA to a mixed set of sensors is feasible. Still, it comes at the cost of introducing additional complexity (level shifting, bus bridging, etc.) compared to the straightforward peripheral model of an MCU. For L.I.N.K., an FPGA remains a strong conceptual option if future iterations require higher parallel computation.

3.1.1.3 Single Board Computer (SBC)

A Single Board Computer (SBC) integrates a microprocessor, memory, and I/O onto a small PCB and typically runs a full operating system such as Linux. SBCs provide ample compute headroom and facilitate the implementation of advanced software features, such as complex graphical user interfaces, local storage, and high-level networking stacks. Many SBCs are also supported by large developer communities, thereby reducing friction when implementing non-embedded features.

While operating systems on a single-board computer (SBC) help improve responsiveness, some operating system-level features can make the timing of system behavior less certain. All of these additional power-consuming components and the generally higher power consumption of an SBC relative to a typical Microcontroller (MCU) mean that, even when only a modest amount of additional processing capability of a system-on-a-chip (SoC) relative to an MCU is desired, SBCs may not be a good first choice for L.I.N.K.

3.1.1.4 Technology Comparison Summary (Table)

TABLE VI: Pros and Cons for Digital Processing Technology

Criteria	MCU	FPGA	SBC
Latency	Low (ms-level)	Very Low (sub-ms)	Higher / variable (OS jitter)
Development	Easy (libraries, examples)	Difficult (HDL, verification)	Moderate (Linux + drivers)
Cost	Low	Medium-High	Medium
Power	Low (portable-friendly)	Medium	High
Sensor/Peripheral Integration	Direct (I ² C/SPI/UART)	Possible, more effort	Possible, more overhead
Fits Project Constraints	Yes	With difficulty	Not ideal

3.1.1.5 Final Choice of Digital Processing Technology

After evaluating the tradeoffs, we chose a Microcontroller Unit (MCU) for L.I.N.K. The main reason is that the device's core jobs are sensor polling, HUD updates, waypoint snapshot collection, and wireless payload transfer. All rely heavily on deterministic timing and reliable peripheral connections rather than desktop-level processing power.

Using an MCU also makes the prototyping and testing phases easier. It enables the integration of sensors via standard communication buses, step-by-step testing of our code, and early refinement of the waypoint capture routine. Even though FPGAs or SBCs remain viable options for expanding the device's capabilities later, starting with an MCU provides a practical, reliable foundation that meets the embedded control needs of the L.I.N.K.

3.1.2 MCU Part Selection

With the MCU technology selected, it is time to examine real-world MCU options for the L.I.N.K. sensor suite and the HUD workflow. Strong candidates must provide on-board wireless, ease of adding a separate wireless device, sufficient GPIOs for all peripheral devices, ample memory for sensor drivers and data buffers, and an easy-to-use development environment.

L.I.N.K. is to include a Sensor Interface section and a Wireless Transmitter section. A board with strong peripherals and good documentation will help mitigate risks associated with integrating various components. Most projects will start with a development board for initial prototyping. Development boards facilitate initial board bring-up and can speed up firmware iterations.

3.1.2.1 ESP32

The ESP32 class of boards [10], [16] is designed for many embedded applications that require capabilities and wireless connectivity. It is commonly used in IoT (Internet of Things) devices that integrate sensors, displays, and communication capabilities. At the system architecture level, the ESP32 can read sensors over I²C [17] or UART, update a small display frequently, and handle wireless data transfer without requiring additional processing power.

The main advantage of the ESP32 is its integrated wireless capabilities; the design avoids adding a

separate radio module and the additional firmware complexity that can arise from multi-processor communication. This comes with a software tradeoff: with wireless integration and sensing on the MCU, the firmware must be structured to prevent wireless events from disrupting sensor timing. This can be mitigated with careful drive design, buffering, and predictable scheduling.

3.1.2.2 STM32

An STM32-class MCU development board could also be used. STM32 microcontrollers are widely used in embedded systems due to their extensive set of peripherals and a wide range of performance options. All common development tools are available for the STM32 family, enabling a professional-grade development and debugging experience. Also, these boards are a good option for sensor-heavy systems due to the large number of I²C/SPI/UART interfaces available and the board's high clock-cycle consistency.

The control-first architecture offers advantages, primarily by decoupling wireless operations from the core system logic. By maintaining ST control-first philosophy, peripheral control and debugging remain consistent with a traditional MCU workflow. This allows driver development to proceed as it would for a regular peripheral, while the radio stack is independently adapted for the external module (Bluetooth or Wi-Fi). Although this approach requires more complex hardware and careful consideration of RF design—including antennas and matching networks—it ensures a robust user experience by maintaining a reliable wireless L.I.N.K. whenever the device is within range of a paired smartphone

3.1.2.3 Nordic nRF52 / RP2040

Nordic nRF52-class boards (focused on Bluetooth) and RP2040-based boards (simple, dual-core MCUs) are both relatively affordable, well-documented platforms. Their comprehensive libraries and community examples make them ideal for prototyping. In particular, the nRF52 is well-suited to designs where low-power wireless communication is the priority, and it offers extensive reference material.

However, a significant tradeoff exists between these options: while the nRF52 integrates wireless at the expense of some computational headroom, the RP2040 offers higher performance but requires external hardware for connectivity. For the L.I.N.K. project, the processor must simultaneously handle sensor fusion, GNSS parsing, display updates, and wireless data packaging. Therefore, the chosen platform must provide a sufficient performance margin and interface flexibility to support these integrated features as the design scales.

3.1.2.4 MCU Comparison Summary (Table)

TABLE VII: Pros and Cons for Candidate MCU Platforms

Criteria	ESP32	STM32	nRF52 / RP2040
Wireless (BLE/Wi-Fi)	Built-in (strong fit)	External module needed	nRF52: BLE built-in / RP2040: external
Development	Easy (large community support)	Moderate (toolchain learning)	Easy–Moderate (good docs/libraries)
Sensor/Peripheral Integration	Strong (I ² C/SPI/UART)	Very strong (embedded-focused)	Strong, varies by board
Performance	High (fits sensing + HUD + BLE)	High (good real-time control)	Moderate–High (platform dependent)
Power Consumption	Low–Moderate (radio use matters)	Low (excellent for portable)	nRF52: very low / RP2040: low–moderate
Cost	Low–Medium	Low–Medium	Low
Fits Project Constraints	Yes (single-board solution)	Yes (more integration effort)	Yes (depends on wireless needs)

3.1.2.5 Final Choice of MCU Part

The selected MCU will be the ESP32. The reason is the need to combine sensor interfacing, HUD updates, and wireless communication within a single, well-supported embedded platform. Using an ESP32 class development board reduces hardware integration overhead during early prototyping while providing a practical path to a custom PCB later.

In addition, the development ecosystem supports rapid iteration, which is important for early validation of the waypoint capture workflow. Because L.I.N.K. must reliably capture a telemetry snapshot and deliver it to a phone, choosing a platform that can host the sensor drivers and the wireless stack together strikes a balance between capability and implementation risk.

3.2 Rangefinding LiDAR

3.2.1 Rangefinding Technology Comparison

Precision rangefinding technology encompass various methods to measure distance with extremely high accuracy by leveraging either timing, phase, or geometric principles. [18] Systems like interferometric laser rangefinders, phase-shift LiDAR, and high-resolution triangulation sensors are all viable methods for our rangefinding purposes.

3.2.1.1 Time-of-Flight (ToF) LiDAR

Time-of-Flight LiDAR determines distance by emitting short laser pulses and measuring the roundtrip-trip travel time of photons reflected from a target. The fundamental relation is shown below.

$$d = \frac{c \cdot \Delta t}{2} \quad (1)$$

The d is the measured distance, c is constant representing the speed of light which we can round to 300,000,000 meters per second, and Δt is the roundtrip time that it takes for the laser pulse to hit the target and return back to the module. [19] These laser pulses come out one at a time, meaning that the laser pulse reaches the target and returns to the sensor before the next laser pulse is release. For extremely long ranges, this can limit the repetition rate of the laser used in the LiDAR and therefore the number of data point the system is capable of acquiring per second. Given that we know our desired range of up to 1 km, we can calculate the delay time of a single pulse traveling 1 km. In this case we plug in our values and see that the delay time for a pulse traveling 1 km is 6.66 microseconds, which equates to a max repetition rate of 150 kHz.

In practice, ToF LiDAR systems use fast photodetectors like avalanche photodiodes (APD) or single-photon avalanche detectors (SPAD) combined with precise timing electronics to achieve precise distance measurements over ranges from meters to kilometers. [20] The strengths of ToF LiDAR are its direct measurement of absolute distance and robustness to varying surface reflectivities. In contrast, the performance of ToF LiDAR depends heavily on environmental factors like fog and rain, as well as ambient lighting conditions. [21] Often times, these system rely heavily on statistical computation of many data points to produce a single distance measurement. For example, a ToF LiDAR module often produces 10,000 measurements per second, but statistically analyzes 3,333 of those measurements at a time to accurately produce a single distance. Hence, the modules are only capable of producing a distance measurement 3 times per second.

3.2.1.2 Phase-Shift (Continuous Wave) LiDAR

Phase-shift LiDAR measures distance by emitting a continuous laser beam whose intensity is sinusoidally modulated and then comparing the phase difference between the emitted and received signals. The distance is proportional to the phase delay, which is shown below.

$$d = \frac{\lambda_m}{4\pi} \cdot \Delta\phi \quad (2)$$

The modulation wavelength is given by λ_m and $\Delta\phi$ is the measured phase shift. [22] Because phase can be measured with very high precision, this method achieves excellent short to mid range accuracy. However, it is inherently ambiguous over distances exceeding one modulation length, so multiple modulation frequencies or unwrapping techniques are required for extended ranging. [23] Compared to pulsed ToF systems, phase-based LiDAR tends to offer higher precision and faster update rates but at the cost of reduced maximum ranges and increased sensitivity to multi path interference.

3.2.1.3 Frequency-Modulated Continuous Wave (FMCW) LiDAR

FMCW LiDAR measures distance by emitting a laser whose frequency is linearly chirped over time and then coherently mixing the returned signal with a reference copy of the transmitted light. The key observable is the beat frequency between the transmitted and received signals, which is directly proportional to the round-trip delay and therefore the distance shown below.

[24]

$$d = \frac{cf_b}{2S} \quad (3)$$

In this case, d is the target distance, c is the speed of light, f_b is the measured beat frequency, and $S = \frac{df}{dt}$ is the chirp rate (the rate at which the laser frequency is swept). [25] Physically, the chirped laser means that a time delay Δt becomes a frequency difference given by $f_b = S\Delta t$. Substituting $\Delta t = \frac{2d}{c}$ gives expression (3).

Unlike conventional ToF, which relies on precise timing of short pulses, FMCW converts the distance measurement into a frequency measurement, which can be resolved with extremely high precision using standard signal processing techniques. A major advantage is that it inherently provides simultaneous velocity information via Doppler shifting of light, allowing separation of moving and stationary targets. [26] Because it uses coherent detection, FMCW systems also exhibit strong immunity to ambient light and can achieve excellent sensitivity at ranges of hundreds of meters to kilometers, especially with narrow-linewidth lasers. However, this performance comes at the cost of significantly higher system complexity, requiring stable laser chirps, precise frequency control, and coherent receivers. In practice, FMCW LiDAR is emerging as a leading architecture for long-range, high-precision sensing in applications like autonomous navigation and defense, where both distance accuracy and velocity discrimination are critical.

3.2.1.4 Technology Comparison Summary (Table)

TABLE VIII: Pros and Cons for Different LiDAR Techniques

Criteria	Time-of-Flight LiDAR	Phase-Shift LiDAR	Frequency-Modulated Continuous Wave LiDAR
Range Capability	Very Far	Short	Very Far
Implementation Complexity	Simple	Complex	More Complex
Cost	Low	Medium	High
Fit for L.I.N.K.	Best fit	Poor fit	Possible

3.2.1.5 Final Choice of Ranging Technique

After comparing the different ranging techniques that LiDAR has to offer, the technique we will use is Time-of-Flight. While FMCW LiDAR is the most technically advanced of the three, offering very high range precision, direct velocity measurement, and high immunity to ambient noise due to its coherent detection, the complexity required for those benefits directly leads to a monetary cost that immediately puts it out of competition with similar products to the L.I.N.K. device. Furthermore, the benefit of velocity measurements is negligible in this case as we are only looking to ping stationary targets onto a map that can be referenced later on. The 1 km range requirement also falls well within the capabilities of ToF LiDAR. Since L.I.N.K. simply needs to produce an accurate distance measurement, the benefits provided by FMCW LiDAR become less relevant and make the cost unjustifiable for our purposes.

3.2.2 Rangefinder Component Selection

ToF LiDAR has complex optics and electronics that is difficult to build and produce from scratch. Without buying bulk materials, the cost is also well beyond a reasonable for the applications we are using it for. Hence, we chose to buy an already existing LiDAR module that can be directly implemented into device without the inclusion of complex software and electronic components that would make integration significantly more complex. Furthermore, ToF LiDAR modules often operate at 905nm or 1535nm due to their low absorption in air. [27]

3.2.2.1 PTYS-12X-251028 905nm Module

The PTYS-12X is an affordable ToF LiDAR module that operates at a wavelength of 905 nm and is capable of ranging large targets at up to 1200 meters during the day and 2000 meters at night. With built in UART connection, it pairs well with our chosen ESP32 MCU development board. Furthermore, it has a compact design with dimensions 22.5mm x 16mm x 9.5mm. The voltage draw is 3.3V. It has an operating range of -20°C to 60°C. Lastly, it operates at a Class 1 power level which is considered eye safe and is great for field use. [28]

3.2.2.2 PTYS-15X-A260317 905nm Module

The PTYS-15X is another affordable ToF LiDAR module that operates at a wavelength of 905 nm and is capable of ranging large targets at up to 1500 meters during the day and undefined range at night. It also comes with built in UART connection, which pairs well with our chosen ESP32 MCU development board. Furthermore, it has a less compact design with dimensions 26mm x 25mm x 13.5mm. The voltage draw is also 3.3V. It has also has an operating range of -20°C to 60°C. Lastly, it also operates at a Class 1 power level. The added range comes at a slightly higher cost and larger dimensions.

3.2.2.3 PTF-LD30X 1535nm Module

The PTF-LD30X is an entry level 1535 nanometer wavelength LiDAR module that has significant ranging capabilities of up to 3000 meters undefined during day or night. The 1535nm wavelength makes it more resistant to environmental factors like fog and humidity. It uses an RS422 Serial Port which is not directly compatible with our chosen ESP32 MCU board. It has the least compact dimensions of 48mm x 21mm x 31mm. It draws a higher voltage of 6V which also draws more electrical power. It has the best operating range of -40°C to 65°C and is still considered a Class 1 eye safe laser. The much better range capabilities and resistance to environmental factors comes at a significantly higher cost and much larger dimensions.

3.2.2.4 Part Comparison Summary (Table)

TABLE IX: Pros and Cons for LiDAR Module Selection

Criteria	PTYS-12X	PTYS-15X	PTF-LD30X
Range Capability	1200 meters	1500 meters	3000 meters
Dimensions	22.5mm x 16mm x 9.5mm	26mm x 25mm x 13.5mm	48mm x 21mm x 31mm
Connection Type	UART/TTL	UART/TTL	RS422
Cost	\$100	\$110	\$420
Operating Temperature	-20°C to 60°C	-20°C to 60°C	-40°C to 65°C
Fit for L.I.N.K.	Best fit	Possible	Poor fit

3.2.2.5 Final Choice of Rangefinder Part

After comparing these three different LiDAR modules, the PTYS-12X is the the best option for our product. The range extends beyond what we are looking to consistently achieve and is the only module that defines the varied performance between day and night operation. What it lacks in absolute range, it makes up for in cost and dimensions which were the most important factors when considering which LiDAR module to buy. Its UART/TTL connection works seamlessly with our chosen ESP32. The PTSY-15X was a tempting option due its better range which would improve the confidence intervals of distance measurements at shorter distances. However, the added complexity caused implementing a larger dimension size outweighed the benefits provided by longer ranges. The PTF-LD30X vastly outperforms in terms of absolute range. however, given the rather limited magnification of the L.I.N.K. device, objects at ranges beyond 1 km would be difficult to define through the binoculars and would serve as poor targets.

3.3 Orientation and Navigation Sensors

The orientation and navigation subsystem of the L.I.N.K. device determines the device's position and orientation when the user presses the button to select a waypoint. The GNSS receiver component of the subsystem is used to determine the device's global position and time. The IMU subsystem is used to determine the device's heading and tilt. The rangefinding measurement, together with the device's global position and time, and its heading and tilt, is used to estimate the selected waypoint's position relative to the device.

The orientation error impacts the error in the target waypoint. Thus, this high-level subsystem is not only about sensor performance but also about system stability. A very accurate sensor does not automatically make it a good sensor for this subsystem. Interference from other magnetic fields, calibration drift, or poor sampling quality can all cause problems. This is a trade-off between accuracy and system stability.

3.3.1 IMU Technology Comparison

An IMU (Inertial Measurement Unit) sensor is used to track an object's movement and its orientation in space by measuring linear acceleration and angular velocity. Some IMUs may

also include a magnetometer that provides an absolute heading, which is a key requirement for our L.I.N.K. application, as we need to stabilise the device's heading while it tracks waypoints. The most commonly used technologies in this field are 6-degree-of-freedom (6-DoF) IMUs and 9-degree-of-freedom (9-DoF) IMUs with magnetometers.

The main evaluation criteria were: heading stability, number of calibration cycles required, resistance to magnetic-field interference, processing time for sensor data, availability of robust libraries, and sufficiently accurate calibration with an adequate update rate.

3.3.1.1 6-DoF IMU

A 6-DoF IMU combines a 3-axis accelerometer and a 3-axis gyroscope. This sensor type can estimate tilt and angular motion very effectively, and short-term orientation tracking can be extremely smooth. In many embedded systems, 6-DoF IMUs are chosen for their low cost, wide availability, and reduced sensitivity to magnetic interference compared with magnetometers.

The limitation is the absolute heading. Without a magnetometer, yaw/heading estimation must rely on integrating gyro measurements over time, which tends to drift. While drift can be mitigated with filtering and periodic correction, L.I.N.K. benefits from a heading estimate that remains stable over time without requiring frequent manual recalibration. For that reason, 6-DoF IMUs are attractive for stabilization and tilt measurement, but may require additional strategy to achieve consistent compass-like heading.

3.3.1.2 9-DoF IMU (w/ Magnetometer)

A 9-DoF IMU (Integrated Motion Unit) is a sensor that contains an accelerometer, gyroscope, and magnetometer. The magnetometer provides absolute heading to the sensor fusion algorithm, along with data from the other sensors. In L.I.N.K., we require the heading of the 9-DoF IMU at the time a waypoint is captured to provide the vector that points in the direction of the intended waypoint.

The downsides are that they are highly sensitive to magnetic fields, so other ferromagnetic objects, voltage spikes, and magnetic fields from magnets in speaker or motor drivers can be problematic. This means that 9-DoF must be evaluated not only by their spec sheet but also by how well they maintain calibration within the actual device packaging. If the system uses a 9-DoF IMU, mechanical placement, wiring strategy, and calibration procedures are key to successful integration.

3.3.1.3 Comparison Summary (Table)

TABLE X: Pros and Cons for IMU Technology Selection

Criteria	6-DoF IMU (Accel + Gyro)	9-DoF IMU (Accel + Gyro + Mag)
Heading / Compass	No true heading (yaw drifts)	Absolute heading possible (mag reference)
Tilt / Motion Tracking	Strong (smooth short-term tracking)	Strong (plus heading support)
Calibration	Usually simple	Requires mag calibration
Magnetic Interference	Less sensitive	Sensitive to nearby metals/currents
Software Complexity	Lower (basic filtering)	Higher (fusion + validation)
Cost / Availability	Low, widely available	Medium, varies by part
Fit for L.I.N.K. (waypoint pointing)	OK (needs heading strategy)	Best (supports pointing + heading)

3.3.1.4 Final Choice of IMU Technology

For L.I.N.K., the 9-DoF IMU with magnetometer category has been chosen. The 9-DoF IMU with magnetometer provides all the necessary information for an absolute heading. An absolute heading is needed for waypoint calculation. An IMU with a magnetometer provides a compass heading without requiring alignment to anything else, but it must be calibrated and is highly susceptible to magnetic-field interference.

This problem, along with others, will be addressed in the design. The design will treat the placement and calibration of the IMU as part of the system integration rather than an afterthought. This will involve ensuring that the IMU is not in the vicinity of any high-current traces and that the firmware includes a method to alert the user if the IMU calibration is deemed inadequate.

3.3.2 IMU Part Selection

The 9-DoF IMU (accelerometer, gyro, and magnetometer) was the best technology category for providing an absolute heading for L.I.N.K. We have decided to use one of the development/breakout boards of commercial IMUs. This approach is also suggested in the guidance given to us by Professor Weeks. The IMU affects waypoint accuracy through the heading, and it is important to test and validate this aspect early in the correct test environment. The use of a module-based design allows flexibility in component placement, wiring, and calibration, which will be determined later in the development process and may not be suitable for a full-board design.

IMU selection criteria: A stable, fused orientation output is required to power a HUD. An understanding of the calibration workflow and quality indicators associated with the IMU. A simple interface, preferably I²C, to ease integration. Availability of libraries and documentation for the preferred programming environment. The proximity of other electronic components and how the IMU will perform in the presence of strong magnetic fields. All of these considerations should focus on overall system reliability rather than on the sensor's characteristics.

3.3.2.1 Bosch BNO055

BOSCH BNO055 9-Axis Absolute Orientation Sensor: The BNO055 is a 9-axis absolute orientation sensor featuring a 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer, along with a digital Direction Motion Operation (DMO) output processor based on an ARM Cortex-M7. In addition to providing the usual three-dimensional raw data, the microcontroller in the sensor can perform sensor fusion to provide output in aligned Euler angles or quaternions, therefore reducing the burden of the main application processor to perform complex mathematical calculations to obtain the sensor orientation from raw data from the 3 individual channels of each sensor. Data communication is via the I²C (in some cases, UART) interface, enabling the sensor to be used in multi-sensor systems.

3.3.2.2 STMicroelectronics LSM9DS1

The STMicroelectronics LSM9DS1 is a 9-DoF (degrees of freedom) inertial measurement unit (IMU) that includes a 3-axis accelerometer, a 3-axis gyroscope, and a 3-axis magnetometer. While some IMU's, and even some other orientation modes, report the "absolute orientation" of the device, this IMU reports the measurements from each sensor. This means that the sensor data must be fused into a heading and orientation by the main microcontroller. This is one of the more common methods for implementing an IMU, as it allows the user to fine-tune sensor data filtering and orientation calculations to the device's specific motion and design.

3.3.2.3 TDK InvenSense ICM-20948

This is the TDK InvenSense ICM-20948, a 9-axis inertial module that includes a 3-axis accelerometer, a 3-axis gyroscope, and an internal magnetometer. Like most other IMUs of this type, the purpose is to transmit acceleration and magnetic field data in real time via I²C or SPI protocols for analysis by the main MCU (microcontroller unit), which then determines the element's orientation using a sensor fusion algorithm. The ICM-20948 and other IMUs like it are typically used for motion tracking. These devices are reported to offer high performance in handheld devices and a wide range of sampling options.

3.3.2.4 Part Comparison Summary (Table)

TABLE XI: Pros and Cons for Candidate IMU IC Options (Dev Board Integration)

Criteria	Bosch BNO055	ST LSM9DS1	TDK ICM-20948
Fusion Output	Onboard fusion (Euler/Quat)	Raw sensor data only	Raw sensor data only
Firmware Effort	Low (use driver + read orientation)	Medium-High (implement fusion)	Medium-High (implement fusion)
Heading Stability (Compass)	Good when calibrated	Depends on fusion + calibration	Depends on fusion + calibration
Calibration Requirement	Required (mag + system calib)	Required (mag + fusion tuning)	Required (mag + fusion tuning)
Integration Approach	Easy via breakout/dev board	Easy via breakout/dev board	Easy via breakout/dev board
Documentation / Libraries	Strong community + libraries	Moderate (varies by library)	Moderate (varies by library)
Tuning / Transparency	Lower (fusion is "black-box")	High (raw data, full control)	High (raw data, full control)
Fit for Prototype Schedule	Best (fast bring-up)	Good, more dev time	Good, more dev time

3.3.2.5 Final Choice of IMU Part

The IMU used on the L.I.N.K. prototype was the Bosch Sensortec BNO055. The BNO055 was used via a breakout/development board rather than being defined as a custom PCB footprint. The development board provided an easy means of connecting the IMU and made it as quick as possible to get it up and running. The BNO055 has built-in sensor fusion, which outputs Euler angles or quaternions. This data could be read directly from the IMU via the GPIO pins on the ESP32-S3; at the time, the team had not yet implemented or tested a full sensor fusion algorithm. Although the IMU was not fabricated on the team's PCB, it was still possible to use a dev board for the IMU, test it in potential locations of the final enclosure, and verify that the calibration is sufficient.

A raw-sensor IMU, such as the LSM9DS1 or ICM-20948, allows us to tweak the characteristics of the sensors individually, which may be useful. However, implementing this from scratch is non-trivial (new code and tests must be written) and may further slow our progress toward end-to-end testing of the full system. We will proceed with the BNO055 for now. It provides the orientation data we need for now, and we'll focus on implementing the remaining dataset (e.g., range, orientation, and GNSS) while ensuring that all BLE data from the board reaches the phone. Issues such as handling stray magnetic fields, calibrating the IMU, and determining the optimal location for the IMU will be addressed later in the project, when we have time.

3.3.3 GNSS Technology Comparison

The GNSS subsystem provides a global position and time reference for the waypoints' anchor points. The GNSS receiver can range from a simple single-frequency GPS receiver for coarse navigation to a more complex multi-constellation and multi-band receiver for high-accuracy positioning. A balance must be struck between being stable enough to place waypoints in a stable position and being cost-effective for a small project.

Factors in the selection include fix reliability, sensitivity, time-to-first-fix behavior, interface simplicity, and antenna requirements. Because L.I.N.K. is intended for outdoor use, antenna placement and sky visibility are as important as module selection, and the GNSS evaluation must consider both.

3.3.3.1 GNSS Module Types / Accuracy Classes

Consumer GNSS modules can be categorized into several real-world use cases. Basic navigation enables the module to provide latitude/longitude readings for applications such as mapping or orienteering/hiking. More advanced modules may support multiple constellations such as GPS, GLONASS, Galileo, or BeiDou. Higher-accuracy positioning can also be achieved with more advanced modules by using dual-frequency signals and additional data from SBAS (Satellite-Based Augmentation Systems) or RTK (Real-Time Kinematic). However, these features often come at a higher cost and with more complexity.

For a portable waypoint device, the GNSS method to employ depends on the required accuracy to meet the desired user experience. If the L.I.N.K. system is only to serve as a general area-return feature, standard GNSS operation will suffice. If the user is required to place waypoints

accurately, it will be necessary to determine whether higher GNSS accuracy is needed and, if so, whether this is feasible in the proposed prototype, given the limitations imposed by adding higher-accuracy GNSS and by antenna size constraints.

3.3.3.2 Antenna Considerations

GNSS performance is very dependent on the antenna design and location. Many small-form-factor GNSS designs use patch antennas. They are mounted over a ground plane and positioned away from obtrusive switching regulators and high-speed digital bus lines. Active antennas have also been implemented to enhance GNSS performance, but they require more DC power and can complicate the RF path to the antenna.

Blocks that may obscure the RF antenna's view of the sky include device geometry and internal electronics packaging. Handheld devices are commonly used in various orientations. Consider what is typically done with the device and how the antenna configuration will be affected. Also consider the effect of the device material on GPS signal loss.

3.3.3.3 Comparison Summary (Table)

TABLE XII: Pros and Cons for GNSS Technology Selection

Criteria	Basic GPS (Single-Band)	Multi-Constellation GNSS	High-Precision (Dual-Band / RTK)
Typical Accuracy	Medium (navigation-grade)	Better stability / consistency	Highest (cm–dm possible)
Fix Reliability (partial sky view)	Lower (fewer satellites)	Higher (more satellites)	High (but depends on corrections)
Time-To-First-Fix (TTFF)	Medium	Often faster	Medium (can be longer)
Antenna Requirements	Simple patch antenna	Simple patch antenna	Better antenna / placement needed
Cost	Low	Medium	High
Power Consumption	Low	Low–Medium	Medium–High
Integration Complexity	Easy (UART NMEA)	Moderate (more config options)	Higher (corrections + tuning)
Needs External Corrections	No	No	Often yes (RTK/SBAS)
Fit for Prototype Goals	Good (quick bring-up)	Very good (robust field use)	Best accuracy, higher effort/-cost

3.3.3.4 Final Choice of GNSS Technology

For the initial prototype, the GNSS approach prioritizes a navigation-grade, single-band receiver that provides a standard UART NMEA-style output and supports an external patch antenna. This provides a predictable embedded integration path while maintaining flexibility to upgrade to higher-precision GNSS later if field testing shows that waypoint accuracy is insufficient.

This approach keeps the GNSS subsystem aligned with the prototype's goals: first, validate the reliability of the waypoint workflow; then refine positioning performance through module upgrades and antenna optimization as needed.

3.3.4 GNSS Part Selection

From Section 3.33, choose a GNSS navigation-grade approach for this project. Next, we select a receiver module or chipset to embed in our L.I.N.K. prototype. At this time of our project, the key performance parameters for the chosen receiver module/chipset are: Reliable field fixes, a simple interface to our embedded board (most are UART compatible and provide NMEA formatted data), and the module package is suitably small to provide an unobstructed antenna within the confines of our housing.

We suspected that performance would vary significantly with antenna location and with prevailing RF noise levels across locations. Several factors for the practical integration of GNSS into our system were also important, including the presence of an external active antenna connector, the level of filtering and amplification built into the module, and the ability to determine when a good fix is obtained and that the satellites are locked on. Our main aim was to obtain the simplest system possible and the shortest possible commissioning time, enabling a sufficient number of repeatable tests before more complex systems would be required.

3.3.4.1 u-blox 7 Family (NEO-7 Class) Module

One candidate option is a GNSS module based on the u-blox 7 (NEO-7 family) receiver platform. This class is widely used in embedded navigation projects and is designed for compact integration while still providing standard GNSS outputs over UART. u-blox's NEO-7 family is also backward compatible with older NEO product workflows, which is useful when leveraging existing tools, wiring conventions, and library examples during early prototype work.

In the low-cost prototyping market, modules such as the GT-U7 are commonly advertised as using a "7th generation" GNSS chipset and as being software-compatible with NEO-6M-style operation. Many of these modules include a UART output and an IPEX-style active-antenna interface (and sometimes a USB-to-serial convenience), which reduces integration effort when validating data output and verifying status early in development. The primary trade-off is that these boards can vary across vendor implementations, so verification testing (cold-start behavior, sustained lock, and noise susceptibility inside the enclosure) is important.

3.3.4.2 Quectel L76-L (Multi-GNSS) Module

The Quectel L76-L GNSS module family is an alternative. The L76-L GPS/GNSS location module (using GPS/GLONASS/BeiDou, etc.) can be configured to use GPS only, GPS/GLONASS, or GPS/BeiDou, and can be set to use or exclude the Galileo/QZSS system. This family of modules is designed for small-form-factor embedded designs and is very compact. The module can track multiple GNSS systems (GPS, GLONASS, BeiDou, Galileo/QZSS, or any combination). All the common embedded interfaces are provided, including UART and I²C.

Reliable in typical operating environments. The multi-constellation capability increases the likelihood of satellite sightings, thereby providing more reliable position estimates and enabling a faster initial fix in urban environments. The drawback is that it costs more than a basic GPS receiver, and that adjustments to operating parameters may be needed to maximize the available word bits and achieve the best update rates. Definitely still an option to consider, especially

where field tests do not show a requirement for precision tracking so much as for stability.

3.3.4.3 MediaTek MT3339-Based GPS Modules

One GPS module that we have not yet examined is a MediaTek MT3339-based GPS receiver. This module uses a widely used GPS chipset found in many "Arduino GPS" modules and other products. It receives NMEA data strings in the standard format, such as that provided by the U-BLOX 6M, and can be configured to update at the GPS rate, matching any data rate required by the application. There is a large amount of information online about these products, how they can be used, and the parsed data strings, so they are likely to be "plug and play" in many cases.

They are easy to obtain and very easy to get working, and provide a stable UART interface that can be used to verify that waypoints are reached within the first few minutes after power-on. The only drawback to using a "navigation grade" module designed for GPS rather than GNSS applications is that it may not perform as well in poor viewing conditions as newer multi-GNSS devices. So they are a good way to bring up the basics of a system, but not a good choice for a permanent module.

3.3.4.4 Part Comparison Summary (Table)

TABLE XIII: Pros and Cons for Candidate GNSS Module/Chipset Options

Criteria	u-blox 7 (NEO-7 class)	Quectel L76-L	MediaTek MT3339
Integration (UART NMEA)	Easy	Easy	Easy
Fix Reliability (field use)	Good	Very good (multi-GNSS)	Good (GPS-focused)
Antenna Support	External active antenna options	External antenna support	Varies by module
Configuration Effort	Low	Moderate	Low
Cost	Low	Medium	Low
Prototype Schedule Fit	Best	Good	Very good

3.3.4.5 Final Choice of GNSS Part

The selected GNSS module for the L.I.N.K. prototype is the GT-U7 GPS module, based on a u-blox 7th-generation/NEO-7 class receiver. This choice supports quick system recall by providing a standard UART GNSS data output compatible with common embedded parsing workflows and offering practical integration features commonly associated with these modules, such as an active antenna interface.

Although the Quectel L76-L offers slightly better multi-GNSS robustness and MT3339-based solutions are widely available for simple GPS bring-up, we have chosen the GT-U7 for its early availability, lower cost, and ease of integration for the first end-to-end waypoint validation. The system design is flexible enough to accommodate a more stable GPS module in partially obstructed environments. In this case, the GNSS module can be replaced with a multi-GNSS module.

3.3.5 Barometer Technology Comparison

Barometers measure atmospheric pressure in units of Pascals (Pa) or hectopascals (hPa). Pressure is greater at lower altitudes than at higher altitudes. We can use pressure changes to estimate altitude changes. This feature is used on some devices to compute a smoothed relative altitude change, thereby filtering out noise in GPS altitude readings. The GPS altitude can vary rapidly and large amounts, depending on the satellite configuration and the quality of the satellite signal.

For L.I.N.K., the barometer isn't used for location but rather as an auxiliary measurement to confirm waypoints. In Telemetry mode, the device's barometric altitude can be used as a reasonably accurate height at the time of a "ping," and, when combined with the waypoint location, yields a much more accurate 3D location. The LiDAR works by emitting a laser and measuring the distance to a target. The IMU measures the altitude and the angle relative to the horizon. Using the trigonometry of the angle between the target and the horizon ($\sin/\cos$ of the angle), the vertical component of the range can be calculated. The barometer serves as an "anchor" for height, improving the stability and accuracy of the location, since GNSS height can be highly variable over short time periods.

3.3.5.1 Barometric Sensing Technologies

One approach is a digital MEMS barometric pressure sensor, which is the most common option in modern embedded designs. These sensors use a micro-machined pressure element and provide a digitized pressure value over a standard embedded interface such as Inter-Integrated Circuit (I²C) or Serial Peripheral Interface (SPI). Digital MEMS barometers are popular because they are compact, low-power, and typically include built-in temperature compensation, thereby reducing the analog design and calibration burden on the main PCB.

A second approach is an analog pressure transducer paired with an external Analog-to-Digital Converter (ADC). This method is less common in consumer embedded projects today, but it is used in industrial sensing because analog sensors can be rugged and sometimes easier to interpret in simple systems. The downside is that it pushes complexity onto the PCB design: signal conditioning, filtering, ADC selection, and temperature compensation become the team's responsibility. For a prototype that already integrates multiple sensors and a display, the analog approach can introduce more integration risk than necessary.

A third approach is using an integrated environmental sensor that combines pressure with additional measurements such as humidity (and temperature). This can be attractive when the project benefits from collecting extra context data for the UI or logs. However, it can also introduce "feature creep," where the system collects more data than it uses. If humidity is not among the device's goals, incurring additional cost and complexity to measure it may not be justified, even if the sensor is technically capable.

3.3.5.2 Comparison Summary (Table)

TABLE XIV: Pros and Cons for Barometer Technology Selection

Criteria	Digital MEMS (I ² C/SPI)	Analog + ADC	Environmental (Pressure+Humidity)
Integration Effort	Easy	Hard	Moderate
Calibration / Compensation	Mostly built-in	Mostly external work	Mostly built-in
PCB Complexity	Low	High	Moderate
Extra Telemetry Features	Pressure + temp	Depends on design	Adds humidity data
Fit for L.I.N.K. Prototype	Best	Not ideal	Optional

3.3.5.3 Final Choice of Barometer Technology

For the L.I.N.K. project, we will be using a digital MEMS barometer. Our barometer will be a digital MEMS barometric pressure sensor with a standard embedded interface (I²C or SPI). The pressure-sensing part of the system should be as simple and as uniform as possible. We are already using a GNSS receiver, an IMU, a display, and wireless communication in this project. Using digital MEMS components enables us to verify the sensor's behavior early in the project. Using pre-existing libraries and the microcontroller's serial debug interface also allows us to test sensor functionality before developing more complex system-level code.

This approach aligns with expectations for a barometer's role in modern systems. The sensor provides stable pressure measurements that yield a reasonable estimate of relative altitude above ground level, with only an 8-pin connection and minimal analog board-level design effort. If additional environmental information were required in the future (e.g., humidity), this sensor family could be extended without departing from this approach.

3.3.6 Barometer Part Selection

Once the digital MEMS barometer approach is chosen, the next decision is to select an IC/module that is well-supported and meets the needs of the HUD telemetry. At this stage, the most important considerations are integration reliability, library support, and stable readings, rather than pursuing extreme precision, which may be limited by enclosure effects and field conditions.

Because barometers are sensitive to airflow, thermal gradients, and enclosure pressure equalization, module selection is only part of the solution. The project must also consider placement inside the enclosure and whether the sensor has a reasonable path to "sense" ambient pressure. Selecting a well-known barometer IC reduces software risk and enables a focus on system-level testing.

3.3.6.1 Bosch BMP280

One option is the Bosch BMP280 barometer, which measures temperature and pressure. It is available to measure via I²C or SPI. It is a widely used sensor among makers, and a wide range of breakout boards is available for it. It can be used to measure temperature and atmospheric pressure to determine altitude. However, the number of samples required to determine altitude

should be as low as possible.

The BMP280 option is appealing because it keeps the subsystem focused on pressure-based altitude trends and basic environmental context, without adding additional channels. In addition, widespread adoption implies strong library support and numerous reference designs. The main limitation is that absolute altitude accuracy will still depend on calibration and ambient atmospheric pressure, but this is true of nearly all navigation-grade barometer use cases.

3.3.6.2 BME280

Another option is the Bosch BME280, which adds humidity sensing to its pressure and temperature measurements. From an integration standpoint, it behaves similarly to the BMP280 and is also widely available on breakout boards with solid library support. If the project were to add environmental-awareness features, humidity data could be used to provide richer telemetry or to support a future “weather context” UI.

However, humidity is not a core requirement for L.I.N.K.’s waypoint workflow. Adding it increases cost and may complicate the software/UI side because it introduces another measurement that the system must either display meaningfully or ignore. This option is technically strong, but its value depends on whether the additional data is actually used in the device’s feature set.

3.3.6.3 TE Connectivity MS5611

Another option could be the TE Connectivity MS5611 barometric pressure sensor. This is a high-accuracy pressure sensor commonly used in applications requiring altitude tracking. We use this sensor when high-accuracy altitude tracking is required.

A more accurate sensor will always be a trade-off against cost and ease of use. A better sensor than the average breakout board for the standard Bosch module will be more expensive and have less functionality. A high-resolution barometer does not necessarily imply high accuracy. The enclosure environment, airflow, and thermal rise can make high-resolution less useful. In the case of L.I.N.K., it is not certain whether the higher sensor resolution will actually result in a better product for the end user, or merely a standard barometer with a fast integration time.

3.3.6.4 Part Comparison Summary (Table)

TABLE XV: Pros and Cons for Candidate Barometer IC Options

Criteria	Bosch BMP280	Bosch BME280	TE MS5611
Measurements	Pressure + temp	Pressure + temp + humidity	Pressure + temp
Integration Effort	Easy (common libraries)	Easy (common libraries)	Moderate
Cost	Low	Medium	Higher
Value for L.I.N.K.	Best fit	Optional feature	Strong, may be overkill
Prototype Schedule Fit	Best	Good	Good

3.3.6.5 Final Choice of Barometer Part

For the L.I.N.K. prototype, the Bosch BMP280 barometer will be used. A single sensor to determine altitude above ground level, with temperature reported alongside the pressure readings, provides more than sufficient detail to give a good idea of the location without overexplaining. The BMP280 includes a large library of sample code that is easier to use than starting from scratch, thereby allowing waypoints and the overall system's functionality to be tested and verified without uncertainty about whether the barometer was functioning correctly.

For now, the MS5611 yields better readings, but we will assess its performance within the air enclosure, accounting for air composition, board placement, and thermal characteristics. For now, we will primarily be using the BMP280. It's easy to hook up and test. If there are any stability issues, we can later replace it with a barometer upgrade using a higher-performance sensor, again using a digital MEMS core.

3.4 HUD Display and User Input

The HUD (Head-Up Display) subsystem is part of the L.I.N.K. user interface. While the user uses binocular optics to range a target, the HUD will display the measured parameters and system status. The HUD requires high contrast for optimal visibility and stable light intensity to ensure a comfortable image and legible text when projected into the viewer's field of view. A small HUD display requires low power consumption. This must be accommodated in a very small space and mounted in the system's handpiece.

The input section of the optical device according to the invention furthermore comprises input means by which the operation of the HUD pages can be carried out in a simple and safe manner, by a user being able to switch between the pages in a simple manner, entering the ranging mode, and being able to click on a waypoint. The input means should enable safe and efficient handling outdoors, should not inadvertently activate incorrect or unintended functions, and should be easily integrated mechanically into the binocular system.

3.4.1 Display Screen Technology Comparison

A selection of display technologies will affect display readability, power consumption, and the complexity of display integration. When considering display technologies for portable devices, the options typically include Organic Light-Emitting Diode (OLED), Liquid Crystal Display (LCD), electronic paper (ePaper), and multi-segment light-emitting diode (LED).

For L.I.N.K., the following criteria were considered important: high optical contrast for optical injection, stable legibility across a variety of lighting conditions, compact size, low power consumption relative to the rest of the system, ease of interfacing, and the use of existing libraries to enable rapid development.

3.4.1.1 OLED (Organic Light Emitting Diode)

Organic Light-Emitting Diode (OLED) displays are emissive. Each pixel on the display emits light. OLED is suitable for HUD-style overlays. It is possible to render highly visible text and graphics on a true-black screen background without a backlight. In an optical combiner (mirror/beam-splitter) setup, high contrast is very important. Thin lines and small fonts remain readable because the “black” pixels contribute almost no stray light.

OLEDs are well-suited for small embedded systems. There are several small OLED panels suitable for HUD display modules that can be driven via I²C or SPI, have a low pin count, and provide update times fast enough for a smooth HUD refresh. OLEDs are not as bright as an LCD in bright sunlight, and it is possible for images to burn-in if they are static and are displayed for a long time at a high brightness, which is a consideration for L.I.N.K. in order to avoid these issues, we have chosen to use a minimal user interface that does not include any static full-bright elements, and implemented an efficient display update strategy.

3.4.1.2 LCD (Liquid Crystal Display)

Liquid Crystal Displays (LCDs) are transmissive displays. On an LCD, the pixels modulate the light from the backlight (or reflect ambient light in reflective LCDs). LCDs are ubiquitous and inexpensive, and they can provide an excellent reading experience when adequately backlit. They are also available in a wide range of sizes and resolutions. LCDs could potentially be used for a relatively simple HUD implementation. The system could be designed with this in mind, knowing that it could later be advanced to more complex layouts.

An LCD HUD is slightly easier to design in, but it is not recommended. LCDs lack the true blacks of OLEDs, which is a significant contrast factor. This could be an issue in a small, low-power HUD module. An LCD would be suitable if the design requires color or a brighter display, but it should also assess the display’s legibility within the system, as the datasheet may not reflect HUD performance.

3.4.1.3 Electronic Paper Display

Electronic paper (ePaper/E-Ink) displays are increasingly resembling printed paper and are generally considered reflective displays. As reflective displays, they achieve high outdoor readability because they require little power to remain readable in sunlight. They also exhibit low idle power consumption, as power is drawn only when pages or text change, not when displaying static images.

One of the problems with a live HUD is the refresh behaviour. Most of today’s ePaper panels refresh very slowly and exhibit significant ghosting when refreshed at high speed. We would like the distance and heading information to flow in smoothly, and the HUD should react immediately during ranging and ping capture. Regardless of the basic design of a HUD, the ePaper refresh rate leads to poor user experience due to lag and ghosting, which is undesirable for fast-moving information, as in a HUD application.

3.4.1.4 Multi-Segment LED Displays

Multi-segment displays (7-segment, 14-segment, or small dot-matrix displays) consist of individual LEDs that are switched on and off to display numbers and basic characters. These displays are highly rugged, easy to drive, and capable of displaying numbers and characters clearly. In a system like L.I.N.K., they could be used, for example, as status indicators or to display a primary value in the user interface when the UI is intentionally kept simple, such as when displaying only the distance.

The technology is limited by the constraints of information density and presentation flexibility. For example, we will display a variety of information fields on the L.I.N.K. HUD, including range, bearing, fix acquisition, and page selectors. Multi-segment displays become difficult to manage when displaying more than 4 or 5 digits. They can also become very cluttered when too many indicators are displayed within discrete segments of the display. Therefore, multi-segment LEDs appear to be a display technology more suited to an auxiliary role than to the primary HUD overlay for multiple information fields.

3.4.1.5 Technology Comparison Summary (Table)

TABLE XVI: Pros and Cons for HUD Display Technology Selection

Criteria	OLED	LCD	ePaper	Seg. LED
Contrast (HUD)	High	Medium	Medium	Medium
Refresh rate	Fast	Fast	Slow	Fast
Power	Low-Med	Med-High	Very low (static)	Low-Med
Outdoor	Good (brightness)	Good (backlight)	Excellent	Excellent (digits)
UI flexibility	High	High	Medium	Low
Fit (L.I.N.K.)	Best	Possible	Not ideal	Limited

3.4.1.6 Final Choice of Display Technology

For the L.I.N.K. project, we will be using an OLED (Organic Light-Emitting Diode) display for the HUD. The OLED was chosen for several key reasons. One of the most important reasons is that the HUD display will be viewable through the optical injection path. The OLED consists of individual emitters (pixels) that can be turned on and off with ease, producing a high-contrast image free of backlight reflections. This creates a very clear display of text and icons. When a user uses the HUD to range and lead a target without looking away, this greatly improves situational awareness and increases system effectiveness.

OLED also fits our system integration goals. Small OLED modules are widely used in embedded designs and offer robust libraries and fast response times. They are fast enough to show live real-time telemetry. LCD and ePaper displays are suitable for some outdoor applications, but for the initial implementation of our L.I.N.K. multi-field HUD display, OLED was the best choice for readability, sufficient update rate, and system miniaturization.

3.4.2 Display Part Selection

With OLED as the HUD display technology, the next step for the L.I.N.K. project is to select a display controller family and the overall configuration of the display module. With the previous component choices in mind, we can review our options and priorities for selecting a display. In particular, the following criteria will be relevant: Display legibility. We aim to select a display that is visually clear, offers high readability for drivers, has low reflectivity, and provides optimal brightness and contrast. Update rate: A fast refresh rate enables a smooth display without trailing artifacts, reducing visual fatigue and discomfort. It is worth noting that we expect a HUD to primarily display a few real-time status widgets rather than many graphics-based widgets. Accordingly, we prefer a display that accurately and reliably renders static text and icons without overburdening the software's processing.

Another factor is the ease of integrating the OLED into the optical design. While all the OLEDs we've examined are roughly the same size, the controllers differ, which affects the memory map, resolution handling, startup code, and libraries used. A "good" display part isn't necessarily one with the highest specification, but one that achieves a good balance between legibility in the optical design, ease of use in the firmware, and a minimum risk when designing a first full prototype.

3.4.2.1 SSD1306

The SSD1306 OLED display-based module is one of the most popular options in the industry. The SSD1306 is an IC (chip) that controls an OLED (Organic Light-Emitting Diode) display. It is actually a very small display controller IC and one of the most popular for small OLED displays in embedded systems. Monochrome small displays are widely used in instrumentation. Hence, it is favorable for the L.I.N.K. Existing libraries, examples, and development tools are also mature, so the risk in developing the firmware of the display will be reduced in the early stage of the integration of the HUD, which can enable the team to focus more on the layout design and the application of telemetry data.

Another advantage of the SSD1306 family of displays is their suitability for a simple, high-contrast HUD (Head-Up Display) interface. Since the main part of the interface will be showing the bearing to the target, the range to the target, and some system status, the display can be a simple monochrome OLED driven by the SSD1306 chip. Although the displays are relatively low-resolution, those in this family are designed to display characters and graphics rather than high-resolution images, so this isn't a problem. In any case, a clean, simple design is far more useful than cramming in lots of detail, and this is especially true for L.I.N.K.

3.4.2.2 SH1106

Another option to consider is a module that uses the SH1106 controller family. These displays are common in embedded projects and resemble SSD1306-based modules in both size and appearance. The SH1106 is similar to the SSD1306 in that it's used for small monochrome OLEDs, and it can provide high contrast and update quickly enough for basic heads-up display telemetry. This makes it a strong choice for certain projects, especially when a simple, effective display solution is required.

The key difference between SH1106 and SSD1306 modules lies in their memory addressing and initialization behavior. This distinction can lead to software compatibility issues, as a library that works seamlessly with one controller may need to be modified to work with the other. As a result, the SH1106 can be a viable option, but it may introduce additional software complexity, particularly if the project's codebase, examples, or preferred libraries rely heavily on SSD1306 support. For projects aiming to minimize unexpected implementation issues, this difference is significant, even if the visual hardware is similar. In essence, the choice between SH1106 and SSD1306 modules depends on the project's specific needs and the team's familiarity with the respective controllers.

3.4.2.3 SSD1327

Another option is to use an OLED module driven by an SSD1327 controller. The SSD1327 controller family is used in some SSD1306- and SH1106-based monochrome OLEDs, but it is more commonly used with grayscale OLEDs. If we want to display more than just on/off and want to use smooth icons or more detailed graphics in the user interface, an SSD1327-based OLED could be a good option. From a purely display-capability standpoint, the SSD1327-based display can provide a more complex HUD than a simple 1-bit monochrome display.

Complexity with questionable value for the current version of the project. The grayscale OLED requires more processing of relatively unimportant data, more complex graphics processing, and more load on the application side. Currently, the first version of L.I.N.K. shows only a few of the most important fields, and the SSD1327-class display's capabilities are not fully utilized. Perhaps in future iterations, this feature will be used, but in the first iteration, it provides sufficient complexity.

3.4.2.4 Part Comparison Summary (Table)

TABLE XVII: Pros and Cons for Candidate OLED Display Controller Families

Criteria	SSD1306	SH1106	SSD1327
Display Type	Monochrome OLED	Monochrome OLED	Grayscale OLED
Library Support	Strong	Good	Moderate
Firmware Complexity	Low	Moderate	Higher
HUD Readability	High	High	High
Graphics Flexibility	Moderate	Moderate	High
Prototype Schedule Fit	Best	Good	Less ideal
Fit for L.I.N.K.	Best fit	Possible	Future upgrade path

3.4.2.5 Final Choice of Display Part

For the L.I.N.K. project, we will use an SSD1306 OLED for the display. We chose it because it appears to be the most readable, is easy to implement in electronics, and does not significantly affect the software. The HUD will display only a few data points, such as range, direction, and status lights, and the SSD1306 family of displays is more than sufficient without adding several lines or more complex elements.

The SH1106 and SSD1327 are the only alternative display controller families that might be needed in the future. The SSD1306 is a simple module to use as a placeholder for our first iteration of the HUD, and it should be straightforward to replace it with a more complex graphics module or a different optical design in later iterations.

3.4.3 Buttons / Encoders / Indicators

The User Input Subsystem provides the means by which the operator can interact with the L.I.N.K. system while focusing on the object of interest. To maximize the system's effectiveness for the operator, the device controls must be straightforward, easily accessible, and highly tactile. Given the device's binocular form factor, the controls must be easily accessible and not require the operator to look away from the optic. As a result, touch-based input is ruled out in favor of physical input devices.

Input actions for this project include switching between HUD elements and pressing the capture button to mark locations of interest. As they are navigation-based input actions, they are easily reproducible with a single input device. Rather than scattering many dedicated buttons across the device's exterior, a multi-function control can be used in the first version.

3.4.3.1 Input Method Comparison

A standard momentary pushbutton can be used as an input switch. They are easy to build, cheap, and simple to interface to the microcontroller. This makes them well-suited for a single event type, such as a switch to turn on power, a switch to confirm an action, or a switch to record information. They are not well-suited to a multi-page HUD, where one must navigate to the displayed page. Single-key chording can be used with multiple-button inputs to reduce the number of keys that need to be pressed, but a multiple-button key chord needs a multiple-button key switch, and these can be quite large and even more complicated to drive than a single-button switch.

Rotary Encoders with Pushbuttons. To pack more functionality into a smaller area, a rotary encoder can be used with a built-in pushbutton. The rotary motion to change pages in the HUD, then use the button press to choose items, mark a Waypoint, etc. This reduces the complexity of navigation for the user; they don't have to remember a sequence of turns to reach a different page, for example. The code to manage the encoder motion and debounce the button is slightly more complex.

Capacitive touch controls can be used, but we do not recommend them as the primary input method. While they enable a very compact enclosure without moving parts, they offer limited tactile feedback and can be problematic in the harsher environment of an outdoor device, especially when operating with gloves. Good tactile feedback is essential to allow for fast and accurate operation in the field.

3.4.3.2 Input Method Comparison Summary (Table)

TABLE XVIII: Pros and Cons for User Input Method Selection

Criteria	Pushbutton	Rotary Encoder + Push-button	Capacitive Touch
Tactile Feedback	High	High	Low
HUD Navigation	Limited	Best	Possible
Capture / Confirm Input	Easy	Easy	Possible
Ease of Integration	Easy	Moderate	Moderate
Outdoor Usability	Good	Good	Less reliable
Fit for L.I.N.K.	Good	Best fit	Not ideal

3.4.3.3 Final Choice of User Input Method

The chosen user interface element for the L.I.N.K. system prototype is a rotary encoder with a pushbutton. The rotary encoder allows the user to scroll through the HUD pages or views. The pushbutton can be used as a single click “trigger” for location capture or confirmation. The chosen user interface elements allow the fewest number of buttons to be placed on the exterior of the housing while remaining functional and easy to use. The interface design is designed with the device workflow in mind and is as non-disruptive to the binocular view as possible.

The input method for the device was chosen to keep the first prototype functional while remaining simple. A standard push-button interface with a large number of buttons would not be practical for providing the required level of control to the user. Capacitive touch buttons can eliminate tactile feedback and can be unreliable in harsh field environments. A rotary encoder provides a good balance among functionality, tactility, and form factor and is well-suited to the L.I.N.K.

3.4.4 Indicator LED Part Selection

Indicator LEDs in the L.I.N.K. system allow the operator to obtain system status information without relying solely on the HUD. Although the HUD contains all the information concerning the system, it would be valuable to have indicator LEDs that light or extinguish when the system is “ON,” and the laser/ranging subsystem is “ON”. These LEDs would be particularly valuable for power-on/off checks and troubleshooting when the HUD is not readily visible.

For this project, the indicator LEDs must be visible to the user as status LEDs rather than merely as PCB debug LEDs. Thus, the PCB may include a power LED for debugging, but the onboard power LED is insufficient for the external visible status LEDs of the final system. For the final system, we will require status LEDs visible to the operator to indicate when system power and the system laser are on.

3.4.4.1 Single-Color Discrete LED

One option is to use basic discrete single-color LEDs and assign a meaning to each. For example, we could keep the LED always on green to indicate that power is on, and have a red LED light up when we switch on the laser/ranging subsystem. This method is very simple, very cheap,

very simple to understand from a user perspective, and does not require the microcontroller to have any more than a basic state or the power control system to have more than a simple switch, since the LED only needs to be driven as an on/off signal.

The user does not want to manage a multi-state lighting system that indicates the status of all subsystems. They just want to easily check whether the device and the laser are on. Using two dedicated LEDs yields a clearer indication than using a single light with multiple meanings. In this case, two LEDs are far clearer than trying to show multiple system states with a single light.

3.4.4.2 RGB LED

Another approach is to use a bi-color (RGB) LED to represent multiple states. One color could be for power-on, one for laser-on, and one for fault or standby. The bi-color (RGB) approach reduces the number of LEDs on the enclosure's outer surface, thereby making the design more elegant.

This makes interpretation and control of the system more complicated. The user must remember what the colours represent, and the microcontroller must manage multiple states with a single device. For this first generation of prototype, this method is not as straightforward as using dedicated indicator lights. However, since we are also using the L.I.N.K. for the HUD, the indicator lights need not display a large amount of information, and a multi-colour approach may be overkill for the required function.

3.4.4.3 Part Comparison Summary (Table)

TABLE XIX: Pros and Cons for Indicator LED Selection

Criteria	Single-Color Discrete LEDs	Bi-Color / RGB LED
Ease of understanding	Very easy	Moderate
Circuit simplicity	Simple	More complex
Cost	Low	Low-Medium
Number of states supported	Limited	High
Fit for L.I.N.K.	Best fit	Possible, but more than needed

3.4.4.4 Final Choice of Indicator Part

For the L.I.N.K. prototype, we chose to use the selected-indicator approach with simple single-color discrete LEDs. A green LED would illuminate to indicate that the device was powered on, and a red LED would illuminate to indicate that the laser/ranging subsystem was enabled. We chose this method as it was the simplest to implement. There was an inherent understanding that the device was on and usable, and no learning was required to interpret the color codes or understand what the LED was attempting to communicate to the user.

This selection also isolates the user-visible LEDs from the debug LEDs on the PCB. The PCB may still include a small onboard power LED for development use, but the external LEDs are intended to be visible to the user during normal product operation. Specifically, the dedicated laser-enabled LED provides a highly visible indication that the ranging section is active. A very

important feature for user visibility and safety.

3.5 Power Supply

When selecting components for the Power Supply Unit (PSU), it is important to identify the number of electrical components and the voltages required to support them. The system will include an array of components, including an ESP32, a HUD, IMU, Barometer, Buzzer, LiDAR, and GPS. These components require a low-ripple voltage rail at 3.3 V, implemented using a high-efficiency switching regulator.

L.I.N.K. must fulfill certain requirements to maintain its identity as a portable binocular rangefinder with an integrated HUD; therefore, the components chosen for the PSU must be compact, lightweight, rechargeable, and efficient.

Due to the battery voltage ranging from 3.0 to 4.2 V during discharge, the use of voltage regulation is required and must be selected accordingly. The regulator must have a minimum efficiency of 90% to meet the constraints of L.I.N.K. [29], [30].

TABLE XX: Power Distribution Table for L.I.N.K.

Component	Voltage (V)	Current Draw	Qty	Power Consumption (W)	Total Power Consumption (W)
ESP32-WROOM-32	3.3	130 mA	1	0.429	0.429
u-blox NEO-7M GNSS	3.3	67 mA	1	0.221	0.221
Bosch BNO055 IMU	3.3	12.3 mA	1	0.041	0.041
Bosch BMP280 Barometer	3.3	1.12 mA	1	0.004	0.004
SMT-0440-T-R Buzzer	3.3	90 mA	1	0.297	0.297
SSD1306 OLED HUD Module	3.3	25.05 mA	1	0.083	0.083
PTYS-12X LiDAR Module	3.3	450 mA	1	1.485	1.485
Total System Power Consumption					2.560 W

3.5.1 Energy Source Selection

3.5.1.1 Wall Outlet

Wall outlet power sources offer a very consistent and reliable source of power, with the advantage of continuous operation without the need to be recharged. This makes them especially applicable to systems that remain in one place for an extended period of time.

3.5.1.2 Batteries

Battery-based power sources offer a variety of different benefits, including compactness and portability, which makes them extremely useful in handheld systems. Various battery chemistries exist, each with different energy densities, environmental concerns, costs, and safety considerations [29].

3.5.1.3 Power Banks

Power banks provide a fully integrated design with already existing battery protection and charging circuitry; therefore, there is minimal safety concern. They tend to have high energy density, with most being about 10,000 mAh, allowing for extended use without having to be recharged. Power banks are also widely commercially available, relatively low cost, and easily implementable due to regulated outputs, and they already have in-house USB standards.

3.5.1.4 Solid-state Batteries

Solid-state batteries utilize solid electrolytes instead of liquid electrolytes, which reduces thermal constraints and produces higher energy densities. compared to the traditional battery [31]. However, this solution poses many limitations, such as price, availability, and lack of options.

3.5.1.5 Energy Source Comparison Summary (Table)

TABLE XXI: Pros and Cons for Energy Source Selection

Criteria	Wall Outlet	Batteries	Power Bank	Solid-State
Portability	Not portable	Highly portable	Portable but bulky	Portable
Runtime	Unlimited	Limited	High capacity	High potential
Safety / Protection	Very safe	Requires protection	Integrated protection	Improved safety
Thermal Performance	Minimal	Moderate	Higher heat	Lower thermal risk
Cost	Low	Low	Low	High
Fit for L.I.N.K.	Not feasible	Best fit	Not ideal	Not feasible

3.5.1.6 Final Choice of Energy Source

Based on the analysis presented in Table XXI, battery-based power was selected as the energy source for L.I.N.K. due to its compactness, portability, and ability to meet the system's voltage and efficiency requirements.

The power supply to the wall outlet was eliminated due to its lack of portability, while the power banks were not selected due to concerns about weight, ergonomics, and thermal performance within the enclosure. Solid-state batteries were also not selected due to limitations in cost, availability, and technological maturity.

Therefore, a rechargeable lithium-ion battery was chosen as the optimal energy source for L.I.N.K., because it satisfies the criteria of system size, cost, and ease of use.

3.5.2 Battery Selection

The battery provides the only source of energy to L.I.N.K., allowing it to be fully operational with all electronic peripherals being powered in a portable fashion. In order to maintain its use as a handheld binocular rangefinder, the following conditions must be met: compact, lightweight, rechargeable, and providing stable power during operation.

The PSU requires a power source capable of supplying a 3.3 V regulated rail, while maintaining a nominal voltage suitable for the regulated voltage value. The battery selected will provide proper regulation along its discharge curve.

3.5.2.1 Lithium-Ion Cylindrical (18650)

Lithium-Ion cylindrical batteries are rechargeable, with a nominal voltage of 3.7 V. They have a high energy density and a nominal capacity of 3000 mAh, providing sufficient capacity for a variety of load requirements [29]. Due to cost considerations, safety characteristics, system ergonomics, and battery life, this is a highly suitable option for many portable electronic systems.

3.5.2.2 Alkaline Batteries (Primary Cells)

Alkaline batteries are non-rechargeable, with a nominal voltage of 1.5 V per cell [32]. Compared to an 18650 cell, they are considered to have low energy density and very minimal capacity. Far more cells would be required to meet higher power and capacity requirements. Due to constraints regarding rechargeability, low energy density, and increased space requirements, this option is generally less suitable for modern portable electronic systems.

3.5.2.3 Nickel-Metal Hydride (NiMH)

Nickel-Metal Hydride batteries are rechargeable and have a nominal voltage of 1.2 V per cell, with a moderate energy density compared to lithium-based batteries [29]. They have fewer safety constraints and are less prone to overcharging, but they discharge more quickly. To meet higher capacity requirements, multiple cells are often required. Due to the limited energy density and inefficiencies compared to lithium-based batteries, this option is less optimal for compact, high-performance systems.

3.5.2.4 Lithium Polymer (LiPo)

Lithium Polymer batteries are rechargeable and have a nominal voltage of 3.7 V per cell, with a high energy density contained in a lightweight form factor [29]. LiPo batteries have one major constraint, which is the requirement for strict charging and protection circuitry due to thermal swelling and potential explosion risks. They provide a strong option for lightweight and compact applications; however, their safety considerations must be carefully managed.

3.5.2.5 Lithium-Ion Prismatic Cells

Lithium-ion prismatic batteries are rechargeable, have a nominal voltage of 3.6 V per cell, and have a high energy density contained in a very small form factor, which is ideal for applications with space constraints [33]. One major constraint is the strict charging and protection circuitry required to ensure proper thermal management, which could lead to fire or explosion under fault conditions. While they are highly effective in terms of size and efficiency, safety considerations associated with custom charging circuitry must be addressed.

3.5.2.6 Battery Comparison Summary (Table)

TABLE XXII: Pros and Cons for Battery Selection

Criteria	18650 Li-Ion	Alkaline	NiMH	LiPo	Prismatic
Rechargeability	Rechargeable	Not rechargeable	Rechargeable	Rechargeable	Rechargeable
Energy Density	High	Low	Moderate	Very high	High
Voltage per Cell	3.7 V	1.5 V	1.2 V	3.7 V	3.6 V
Form Factor	Compact	Bulky	Bulky	Flexible, light	Compact flat
Safety Complexity	Moderate	Low	Low	High	High
Cost	Low	Very low	low	Moderate	Moderate
Overall Suitability	Best overall	Poor	Moderate	Strong but risky	Strong but complex

3.5.2.7 Final Choice of Battery Selection

Based on the comparison shown in Table XXII, the Lithium-Ion cylindrical 18650 battery was selected as the power source for L.I.N.K. The requirements outlined in this section specify that L.I.N.K. must remain compact, lightweight, rechargeable, and maintain stable power for the entire system. The 18650 battery is the most suitable option because of its high energy density, and its capacity meets the battery life requirement under the load of the peripherals for extended use. The 18650 battery also fits the safety requirement and is able to be charged by a commercially available charger, reducing the risk of fire or explosion.

In comparison, the other options, such as alkaline batteries, require recharging and multiple cells, which reduces compactness and is highly inefficient for reuse. Nickel-Metal Hydride batteries are rechargeable, but have a lower energy density, requiring many cells, which is not ideal for compactness of the system. Lithium Polymer and prismatic cells offer high energy densities and reduced form factors, but require strict charging and protection circuitry to be safe to use. The complexities of the implementation and safety concerns make them unsuitable options for L.I.N.K. 18650 batteries are the most balanced solution in terms of safety, performance, cost, and availability, they offer the most practical solution.

3.5.3 Voltage Regulation Selection

Each L.I.N.K. subsystem requires a 3.3 V operating voltage. Since the ESP32 and the required sensors all operate at 3.3 V, a tightly regulated voltage, choosing a regulator with proper efficiency, cost, and overall suitability should be considered so that L.I.N.K. is operational in normal use cases and all components are properly powered. This section will explore different regulator types and weigh out the pros and cons related to L.I.N.K.

3.5.3.1 Linear Regulation

A linear regulator dissipates heat as a means to ensure a stable output voltage. They are very simple in architecture and are extremely useful in analog circuits [30]. The use case is hindered when the difference between the input and output voltage is large or when the load current is high, since battery life can be reduced. The use of a linear regulator in L.I.N.K. would not be ideal due to the more complex nature of the subsystem architecture and power requirements.

3.5.3.2 Switching Regulation

A switching regulator switches energy on and off very quickly, which regulates the output voltage. This uses inductive and capacitive components, which results in high efficiency [30], [34]. While being more complex in nature, juxtaposed to a linear regulator, they are very useful in systems requiring power-efficient applications. A large difference between the input and output voltage allows this option to minimize power losses. The use of switching regulators is required for this use case due to the need for high efficiency; L.I.N.K. needs to minimize power use due to the handheld package it is contained in.

3.5.3.3 Voltage Regulation Comparison Summary (Table)

TABLE XXIII: Pros and Cons for Voltage Regulation Selection

Criteria	Linear Regulation	Switching Regulation
Efficiency	Low (power loss as heat)	High efficiency
Thermal Performance	High heat dissipation	Lower heat generation
Complexity	Simple	More Complex
Suitability for Battery Systems	Poor	Excellent
Voltage Conversion Capability	Limited (step-down only, inefficient)	Step-up and step-down capable
Fit for L.I.N.K.	Not suitable	Required

3.5.3.4 Final Choice of Voltage Regulation Selection

Based on the system-level requirements for L.I.N.K., involving a compact, battery-powered handheld system, battery life is a crucial design factor that needs to be considered. Therefore, switching regulation is the type of voltage regulation being used in L.I.N.K. A single 18650 lithium-ion cell will require the high efficiency of the switching regulator to maximize battery life.

In Table XXIII, linear regulation is limited in efficiency because of its power dissipation in the form of heat, which would also be an issue due to the form factor of L.I.N.K. This option has limitations regarding portability, thermal limits, and energy efficiency.

Switching regulation provides high efficiency with limited heat dissipation. L.I.N.K. requires a regulated voltage rail at 3.3 V; this can not be achieved using linear regulation. Therefore, switching regulation is required for the scope of this project.

3.5.4 Switching Regulator Selection

L.I.N.K. requires a switching regulator, which is crucial to ensure high-efficiency power delivery to the MCU, all sensors, and peripherals, requiring both a 3.3V rail. The handheld compactness constraints of the system prioritize heat dissipation minimization as a major design constraint. To address the requirements of L.I.N.K., both buck, boost, and buck-boost regulators are evaluated.

3.5.4.1 Buck Regulator

A buck regulator is a type of switching regulator that steps down voltage, which brings the output voltage down to a lower level than the input [30]. This component uses both inductive and capacitive components to maintain high efficiency.

3.5.4.2 Boost Regulator

A boost regulator is a type of switching regulator that steps up voltage, which brings the output voltage up to a higher level than the input [34]. This component uses inductive and capacitive components to maintain high efficiency.

3.5.4.3 Buck-Boost Regulator

A buck-boost regulator is a type of voltage regulator that can step and step down a voltage to a regulated output [30]. They are useful in situations where batteries are required due to their varying discharge voltages. This regulator has high efficiency, reduced heat dissipation and holds a stable output, while the voltage of the battery drops.

3.5.4.4 Switching Regulator Comparison Summary (Table)

TABLE XXIV: Pros and Cons for Switching Regulator Selection

Criteria	Buck Regulator	Boost Regulator	Buck-Boost Regulator
Voltage Conversion	Steps down voltage	Steps up voltage	Steps up or down voltage
Efficiency	High	High	Moderately High
Complexity	Moderate	Moderate	Higher complexity
Application Use	3.3V	5 V	3.3V & 5V
Input vs Output Relationship	$V_{out} < V_{in}$	$V_{out} > V_{in}$	$V_{out} \leq V_{in}$
Fit for L.I.N.K.	Limited	Can be used for 5V	Best fit for 3.3V and 5V rails

3.5.4.5 Final Choice of Switching Regulators

L.I.N.K. requires an efficient switching regulator: buck-boost regulator. Due to the compact form factor of the package of L.I.N.K., there are strict battery constraints; therefore, minimizing power consumption is important for the product.

The buck-boost regulators must regulate the battery voltage to 3.3V for all components. Using a buck-boost regulator ensures all components are fed the correct voltage, regardless of battery voltage variations. The regulator needs to have a minimum of 90% efficiency to extend battery life and ensure accurate powering of all components. This is important to help with heat dissipation in the small enclosure.

In addition to efficiency, cost and complexity must be considered in regulator selection. One of the selling points of L.I.N.K. is its pricing relative to competitors, so reducing unnecessary costs is crucial. The complexity of the regulator goes hand in hand with cost and efficiency, so it needs to fit within those constraints. Tools such as TI Webench[®] Power Designer [35] were

used to evaluate and select appropriate regulator components based on these requirements.

The selection of a high-efficiency switching regulator is crucial in the overall functionality of the L.I.N.K., ensuring stable power delivery, strong battery life, and reliable performance.

3.5.5 3.3V Regulator Selection

For L.I.N.K., the regulator needs maintain a 3.3V rail supplied from the 18650 lithium-ion battery, as highlighted earlier. L.I.N.K. requires a regulated 3.3V to power the MCU and low-power peripherals. The regulator must consider efficiency, cost, footprint, and overall performance as applied to an embedded battery-powered system.

3.5.5.1 LTC3113

The LTC3113 is a high efficiency synchronous buck-boost regulator designed to step up or step down the voltage, making it especially suitable for battery applications [30].

3.5.5.2 TPSM863253

The TPSM863253 is a buck regulator that combines the controller, power switches, and inductor into one package. This provides a simplified design process and reduces the external components needed for the overall design. This regulator has high efficiency and built-in shielding, which reduces electromagnetic interference. This device has a relatively small footprint.

3.5.5.3 3.3 V Regulator Comparison Summary (Table)

TABLE XXV: Pros and Cons for 3.3V Regulator Selection

Criteria	LTC3113	TPSM863253
Efficiency	Up to ~96% (High)	93.1% (Moderate)
Cost (BOM)	\$12.00 (High)	\$0.85 (Very Low IC, higher module cost context)
Footprint	4 x 4 mm	53 mm ² (Slightly Larger)
Design Complexity	More complex	Simple
Current Capability	900mA	3 A (High headroom)
Fit for L.I.N.K.	Best for full battery range regulation and robust performance	Good for rapid prototyping, less flexible

3.5.5.4 Final Choice for 3.3 V Regulator

In Table XXV, the LTC3113 buck-boost regulator was chosen to facilitate the 3.3 V rail due to its ability to buck and boost simultaneously. L.I.N.K. is a battery powered system, meaning as the battery discharges, it is advantageous to have a buck-boost, so that it maintains a strong 3.3 V rail.

3.6 Communication

3.6.1 Wired Communication

The wired communication architecture inside L.I.N.K. is responsible for short-range data exchange between the ESP32-S3 and the on-device peripherals. Unlike the wireless BLE link used to communicate with the phone, these wired interfaces exist entirely within the binocular hardware and must be selected based on pin count, ease of integration, throughput, electrical simplicity, and availability of software support. Because the system includes both low-bandwidth configuration/status peripherals and serial-output sensor modules, more than one wired protocol category must be considered.

3.6.1.1 I²C

Inter-Integrated Circuit (I²C) is a two-wire synchronous serial bus commonly used for low-to-moderate bandwidth embedded peripherals. One of the main advantages of I²C is that multiple devices can share the same clock and data lines, provided each device has a unique address. This makes I²C attractive for compact embedded systems where minimizing wiring complexity and GPIO usage is important.

For L.I.N.K., I²C is well-suited to peripherals such as the IMU, barometer, and OLED HUD display. These devices exchange relatively small packets of configuration and telemetry data, and they benefit more from simple wiring and library support than from extremely high bus throughput. The main drawbacks of I²C are that it is a shared bus, so a single malfunctioning device can affect the entire bus, and that its overall throughput is lower than with point-to-point interfaces such as SPI. In addition, address conflicts can occur if two candidate peripherals use the same fixed address.

3.6.1.2 UART

Universal Asynchronous Receiver-Transmitter (UART) is a point-to-point serial communication method that does not require a shared clock line. UART is widely used by sensor modules that internally handle their own processing and expose formatted data frames to the host microcontroller. This makes UART especially useful for modules such as GNSS receivers and certain LiDAR rangefinders.

In L.I.N.K., UART is attractive because both GNSS and long-range LiDAR modules commonly ship with UART interfaces and example code. UART also provides isolation between devices because each module occupies its own serial channel rather than sharing a bus. The main disadvantages are increased pin usage compared to I²C and reduced flexibility in attaching multiple devices, since each UART peripheral typically requires its own receive/transmit path.

3.6.1.3 SPI

Serial Peripheral Interface (SPI) is a synchronous serial interface that typically offers higher data throughput than I²C. SPI is commonly chosen when a design requires fast sensor readout,

display refresh, or streaming of larger data blocks. Its main strength is performance, since it uses separate clock and data lines and avoids the addressing overhead of I²C.

However, SPI also increases wiring complexity because each peripheral generally requires its own chip-select line in addition to shared clock and data lines. For L.I.N.K., SPI was considered a possible option for high-speed peripherals, but the current prototype does not require the additional bandwidth of SPI for the selected sensor set. The present device prioritizes integration speed, low pin count, and simple library support over maximum internal data throughput.

3.6.1.4 Comparison Summary (Table)

TABLE XXVI: Pros and Cons for Wired Communication Interface Selection

Criteria	I ² C	UART	SPI
Wiring Complexity	Very low (2-wire shared bus)	Low (point-to-point)	Moderate (clock, data, chip select)
GPIO Usage	Very low	Moderate	Higher
Multi-Device Support	Excellent shared-bus support	Poor (one link per device)	Moderate
Throughput	Moderate	Moderate	High
Typical L.I.N.K. Use	IMU, barometer, OLED HUD	GNSS and LiDAR modules	Future high-speed peripherals
Software / Library Support	Strong	Strong	Strong
Fit for L.I.N.K.	Best for shared low-speed peripherals	Best for smart serial modules	Possible, but not required now

3.6.1.5 Final Choice of Wired Communication Architecture

The wired communication architecture chosen for L.I.N.K. is a hybrid approach using I²C for shared low-bandwidth peripherals and UART for self-contained sensor modules. The IMU, barometer, and OLED HUD display are grouped onto the I²C bus because they benefit from the low pin count and straightforward shared-bus topology. The GNSS receiver and the PTYS-12X LiDAR rangefinder use UART because both modules expose serial data streams and are easier to integrate as independent point-to-point devices. SPI remains a viable future expansion option, but for the current prototype, it adds wiring and firmware complexity without providing a meaningful system-level benefit.

3.6.2 Wireless Communication

One of the major components of the L.I.N.K. unit is the binocular system's wireless communication subsystem. This subsystem would be capable of transmitting the coordinates of a selected waypoint to an external device, such as a mobile phone, via an associated application. The purpose of the binocular system is to store measurements of objects of interest and to view them later on a map or use them for navigation. The wireless link required for this purpose must reliably transmit data such as measured range, bearing, and position, as well as the measurement date and time, without overloading the binocular system's electronics.

The first iteration of the L.I.N.K. project will use the wireless link to communicate between the

L.I.N.K. system and a nearby mobile phone. Thus, wide area networking is not a priority for this application. What is required is a short-range, low-bandwidth link that consumes as little power as possible, is easy to pair, and has reliable data transfer characteristics for the small amount of data being sent. As the data to be sent consists of waypoints and telemetry information, rather than video feeds or large files, the choice of communication technology for this application can be based on practical considerations of available embedded hardware rather than on theoretical communication bandwidth.

3.6.2.1 Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE) [11] is a technology designed for ultra-low-power transmission and reception. It is common in embedded systems and other mobile devices such as smartphones and tablets. We believe BLE is a suitable protocol for short-range, low-power communication between devices that transmit small amounts of data. In this use case, the transmitted data rarely changes and is typically short, a perfect fit for the L.I.N.K. system. A waypoint packet will, on average, carry only a few lines of data, such as Latitude, Longitude, Heading, Range, and altitude/status, which is more than enough for BLE to successfully transmit the information.

We can set the L.I.N.K. to advertise itself as a peripheral device near the Waypoint location and have a smartphone app that directly connects to it. No internet or router is required. This is very useful in the field and also helps to conserve battery life. The downside of BLE is that it is really only suited to small packets of information. We must handle packet connection, acknowledgment, and retransmission in our firmware to ensure that the Waypoint packet is actually sent rather than merely attempted.

3.6.2.2 Wi-Fi

WiFi [36] can also be used for communication. WiFi offers higher bandwidth than BLE and is available on many modern embedded boards. If large amounts of data need to be sent or if data needs to be sent to the cloud frequently, then WiFi is a viable solution. It would also allow for a more in-depth webpage to be hosted locally on the device. This is a possibility for the future if the system is to be made more complex.

Wi-Fi introduces several disadvantages for the first prototype. There are several reasons for this. For example, battery life is longer with BLE than with Wi-Fi. Additionally, the user must address Wi-Fi issues, including network connectivity, passwords, and mobile hotspot sharing, which are not necessary in this case. The main goal of the first product in this project is to provide a reliable point-to-point connection between the device and the phone.

3.6.2.3 Comparison Summary (Table)

TABLE XXVII: Pros and Cons for Wireless Communication Protocol Selection

Criteria	Bluetooth Low Energy (BLE)	Wi-Fi
Power Consumption	Low	Higher
Data Rate	Moderate	High
Best Use Case	Small telemetry / waypoint packets	Large data transfer / web services
Phone Integration	Direct and simple	Possible, more setup required
Infrastructure Required	None	Often needs network or hotspot
Prototype Complexity	Lower	Higher
Fit for L.I.N.K.	Best fit	Future expansion option

3.6.2.4 Final Choice of Communication Protocol

The L.I.N.K. prototype will use Bluetooth Low Energy (BLE) technology. It appears that this technology is best suited to this first version. The amount of data that the device needs to send is low: coordinates, radius, direction, and state variables. BLE seems to have sufficient bandwidth for this purpose, and it is much more power-efficient than WiFi, which would require more resources to transmit the same amount of data.

BLE is more in line with the field's intended workflow. Rather than connecting and syncing the device before going out, you would power it on and pair it with your phone before recording GPS waypoints. This removes the need to search for a Wi-Fi hotspot or sit near a router to establish an internet connection before heading out to get work done.

3.7 Embedded Firmware

The firmware of L.I.N.K. handles most of the components on the board, including sensors, the HUD display, and the BLE wireless link. It also handles the ping button during the waypoint capture sequence. Reliable firmware is needed under tight real-time constraints. Examples of such constraints are: Sensor readings need to be collected at 20 Hz (samples per second); Telemetry needs to be sent to the smartphone at 5 Hz; The device needs to handle commands and BLE events. In addition to the various peripherals included on the board (a number of them are connected on shared buses such as I²C and UART) as well as the wireless stack, the firmware, the application code and the drivers they use have a great impact on the reliability of the system, the productivity of the development process and also on meeting the tight time schedules.

3.7.1 Embedded Programming Environment and Language Comparison

The embedded software environment for L.I.N.K. determines how quickly the firmware can be developed, tested, and integrated with the selected sensor suite and wireless subsystem. In addition to raw execution performance, the chosen environment must support reliable access to common embedded peripherals such as I²C, UART, GPIO, timers, and Bluetooth Low Energy (BLE). The choice of programming language is closely tied to this decision, since the firmware

must manage real-time sensor polling, waypoint capture logic, HUD rendering, and wireless data transfer while remaining maintainable throughout prototyping.

For L.I.N.K., the main evaluation criteria were development speed, availability of existing libraries for the selected peripherals, BLE support, debugging simplicity, performance for real-time embedded control, and compatibility with the ESP32-S3 platform.

3.7.1.1 Arduino Framework with C++

The Arduino framework [37] is one of the most widely used embedded development environments for rapid prototyping. It provides a simplified hardware abstraction layer for common interfaces such as I²C, UART, SPI, GPIO, and timers, while allowing firmware to be written primarily in C++. One major advantage of the Arduino ecosystem is its large library base. Many of the peripherals relevant to L.I.N.K., such as the BNO055 IMU [38], BMP280 barometer [39], SSD1306 OLED display [40], [41], and common GNSS modules, already have mature and well-documented Arduino libraries available. Builds were managed with PlatformIO [42] so that toolchain pinning and library versioning remained reproducible across the team.

For L.I.N.K., this environment is attractive because it minimizes bring-up time and allows each subsystem to be validated quickly during early prototyping. The C++ language also provides structured code organization through classes, reusable drivers, and typed data structures, which is useful for managing telemetry snapshots and waypoint payloads. The main drawback is that the Arduino framework abstracts away some lower-level control, so it is not always the most optimal choice when absolute control over timing, memory allocation, and peripheral configuration is required.

3.7.1.2 ESP-IDF with C/C++

The Espressif IoT Development Framework (ESP-IDF) [43] is the native software development framework for the ESP32 family. It provides low-level access to the ESP32-S3 hardware, including FreeRTOS [44] integration, hardware timers, interrupt management, DMA, and the full BLE software stack [11]. Firmware in ESP-IDF is typically written in C, although C++ can also be used where appropriate.

The main strength of ESP-IDF is control. It allows the developer to manage timing, task priorities, memory behavior, and peripheral interactions more directly than the Arduino framework. This makes it well-suited for production-grade firmware or systems in which strict real-time behavior must be guaranteed. However, ESP-IDF generally requires more boilerplate code and more driver-level development effort. Many convenient Arduino libraries are not directly reusable in ESP-IDF without modification, which increases the implementation burden. For L.I.N.K., this added complexity can slow down early system integration when the priority is validating the complete waypoint workflow rather than optimizing every low-level detail.

3.7.1.3 MicroPython

MicroPython [45] is a lightweight implementation of Python designed for microcontrollers and other resource-constrained systems. It offers a highly accessible embedded programming model

and can significantly speed up simple experimentation because firmware changes can often be tested interactively. This makes it useful for educational contexts and early proof-of-concept work.

Despite this convenience, MicroPython is less suitable for L.I.N.K. because the system requires predictable timing, stable BLE behavior, efficient sensor polling, and a relatively complex multi-peripheral firmware structure. Compared with C or C++, MicroPython introduces more runtime overhead and provides less direct control over low-level hardware behavior. It also has a smaller ecosystem for performance-critical embedded integration on the ESP32-S3 than Arduino or ESP-IDF. As a result, MicroPython is attractive for quick demonstrations, but not ideal for a portable embedded instrument that must reliably capture and transmit waypoint data.

3.7.1.4 Comparison Summary (Table)

TABLE XXVIII: Pros and Cons for Embedded Programming Environment and Language Selection

Criteria	Arduino + C++	ESP-IDF + C/C++	MicroPython
Development Speed	Fast prototyping	Moderate	Fast for simple tests
Library Availability	Excellent for common sensors and displays	Moderate, more low-level work	More limited for full system integration
Hardware Control	Good	Best	Limited
Timing Predictability	Good for prototype scale	Strong	Weaker
BLE Integration	Strong and accessible	Strong but lower-level	Possible, but less practical
Debugging Simplicity	Easy	Moderate	Easy for basic scripts
Performance Efficiency	Good	Best	Lower
Fit for L.I.N.K.	Best fit for prototype development	Strong future upgrade path	Not ideal for final embedded workflow

3.7.1.5 Final Choice of Embedded Programming Environment and Language

For the L.I.N.K. prototype, the selected embedded programming environment is the Arduino framework running on the ESP32-S3, with firmware written primarily in C++. This combination provides the best balance between development speed, peripheral library availability, and sufficient low-level control for the current scope of the project.

The main reason for this selection is that L.I.N.K. depends on a diverse set of peripherals, including an IMU, a barometer, a GNSS receiver, an OLED HUD display, and a BLE communication stack. Using Arduino with C++ allows these components to be integrated quickly via mature libraries while still supporting structured firmware design with typed data structures, modular drivers, and reusable classes. Although ESP-IDF offers greater low-level control and remains a strong option for future refinement, the Arduino-plus-C++ approach better supports rapid end-to-end validation of the waypoint capture-and-transmission workflow during the prototype stage.

3.7.2 Firmware Architecture Selection

After selecting the embedded programming environment, the next decision for L.I.N.K. is the overall firmware architecture. The firmware must repeatedly sample multiple sensors, update the

HUD, handle button input, transmit telemetry over BLE, and manage waypoint delivery without allowing one subsystem to block the rest of the system for an unacceptable amount of time. Because the ESP32-S3 already supports multitasking internally and allows a simpler single-loop design, the main architectural decision is whether to use a cooperative state-machine approach or a task-based RTOS architecture.

The most important evaluation criteria for this decision are timing predictability, implementation complexity, debugging effort, watchdog behavior, and suitability for the current prototype scale. In the current L.I.N.K. design, the firmware is expected to sample sensors at approximately 20 Hz, transmit telemetry to the phone at approximately 5 Hz, update the OLED at a slower human-visible refresh rate, and retain waypoint reliability through local queueing and retry logic.

3.7.2.1 Cooperative State Machine Architecture

In a cooperative state machine architecture, the firmware runs primarily inside a single non-blocking main loop. Each subsystem is serviced when its timing condition is met. For example, the main loop may read sensors every 50 ms, transmit telemetry every 200 ms, debounce button input on its own interval, retry queued waypoint transmissions periodically, and refresh the HUD on a slower update schedule. Rather than preempting tasks, the system progresses by checking timestamps and calling short, non-blocking routines in sequence.

This approach is attractive for L.I.N.K. because it is relatively simple to implement and debug. Shared data structures such as the telemetry snapshot, waypoint payload buffer, and module status flags can be accessed sequentially without mutexes, queues, or other synchronization primitives. This reduces software complexity and makes serial debugging more straightforward during prototyping. In addition, most peripheral transactions in the current design are short relative to the 50 ms sensor polling interval, so a cooperative loop can still meet the required timing behavior without excessive scheduling overhead.

The main limitation is that all subsystems share one execution path. If one routine begins to take significantly longer than expected, such as a blocking sensor read, a long display update, or an unexpected BLE stall, the delay can affect the responsiveness of the rest of the firmware. For small embedded systems, this tradeoff is often acceptable, but it becomes less attractive as subsystem count, timing pressure, and concurrency demands increase.

3.7.2.2 RTOS-Based Task Architecture

An RTOS-based architecture divides the firmware into multiple scheduled tasks, each with its own priority, stack, and execution timing. Under this model, sensor polling, BLE telemetry transmission, HUD updates, button handling, and queue retry behavior can be assigned to separate FreeRTOS [44] tasks. The operating system scheduler then determines when each task runs, allowing time-critical tasks to preempt lower-priority work when necessary.

The main advantage of this approach is better isolation between subsystems. A longer-than-expected display update does not necessarily delay BLE telemetry transmission, and a sensor read that stalls temporarily does not prevent other tasks from running if they are structured correctly. This architecture also provides a clearer upgrade path if future versions of L.I.N.K.

require more complex concurrency, background processing, or tighter timing guarantees.

However, an RTOS-based design introduces additional complexity. Shared firmware data must be protected using synchronization mechanisms such as mutexes, queues, semaphores, or task notifications. This increases the likelihood of concurrency-related bugs, such as race conditions, priority inversion, deadlocks, or stack-sizing problems. For the current L.I.N.K. prototype, these costs are high because the present workload is still relatively modest: the sensor polling period is on the order of 50 ms, while individual peripheral transactions are much shorter. As a result, the advantages of a full RTOS task architecture are real, but they are not yet large enough to clearly outweigh the added implementation and debugging burden for the first prototype.

3.7.2.3 Comparison Summary(Table)

TABLE XXIX: Pros and Cons for Firmware Architecture Selection

Criteria	Cooperative State Machine	RTOS-Based Task Architecture
Implementation Complexity	Low	High
Debugging Simplicity	Straightforward single-threaded flow	More difficult due to concurrency
Shared Data Handling	Simple sequential access	Requires mutexes, queues, or semaphores
Timing Isolation	Moderate	Strong
Scalability for Future Features	Moderate	Strong
Watchdog / Scheduling Complexity	Lower	Higher
Fit for Current Prototype	Best fit	Possible, but not yet justified

3.7.2.4 Final Choice of Firmware Architecture

For the first L.I.N.K. prototype, the selected firmware architecture is a cooperative state-machine design implemented around a non-blocking main loop. This architecture provides the best balance between timing adequacy, implementation simplicity, and rapid debugging during integration of the sensor suite, HUD, BLE link, and waypoint queue logic.

The main reason for this choice is that the current firmware workload remains well within the timing limits of a cooperative architecture. Sensor polling occurs at approximately 20 Hz, telemetry transmission occurs at approximately 5 Hz, and the remaining periodic tasks such as display refresh and queue retry do not require hard real-time isolation at this stage. Because individual peripheral transactions are short relative to the system polling intervals, a carefully structured single-loop architecture can serve all subsystems without requiring the added synchronization and debugging complexity of a full RTOS task model.

Although a full RTOS-based architecture was considered, it was not selected for the current prototype because its benefits are not yet large enough to justify the added software complexity. However, it remains a strong upgrade path for the future if later versions of L.I.N.K. require tighter subsystem isolation, more demanding concurrent processing, or expanded firmware functionality.

3.7.3 Sensor Driver Strategy

The L.I.N.K. sensor suite uses two different types of buses: I²C for the IMU, barometer, and OLED display, and UART for the GNSS and time-of-flight LiDAR rangefinder. The choice of driver strategy significantly impacts how the firmware interacts with peripherals and enables easy substitution/replacement of individual sensors.

3.7.3.1 Vendor Libraries vs. Custom Register-Level Drivers

The first option is to use vendor-provided Arduino libraries (Adafruit BNO055 [38], Adafruit BMP280 [39], TinyGPS++ [46], Adafruit SSD1306 [41]) that contain highly optimized I²C and UART code, greatly simplifying the development process. The libraries handle turning the sensor on and off, as well as calibrating it. They also include functions to access sensor data in engineering units (for example, degrees, hPa, and meters). The cost is losing control over the I²C protocol and transaction order.

The second option is to write low-level register drivers for each sensor that communicate with its I²C interface to read/write internal registers, or with its UART interface to parse the sensor's binary frames. This method allows for greater control over how the sensor is accessed (i.e., more precise timing, control over the sensor's power-down features, and proper handling of error conditions in the code), but it requires much more code to write and test. For example, with the BNO055, we'd need to reimplement calibration and fusion modes, which are currently handled internally in this library.

3.7.3.2 Bus Protocol Considerations (I²C / UART / SPI)

The I²C bus can accommodate the IMU, barometer, and OLED on a shared two-wire interface, since each device has a unique address. The 400 kHz Fast-mode clock provides sufficient bandwidth for the combined traffic at the target polling rate. I²C is the natural choice for these three peripherals because all three vendor libraries default to I²C.

UART is required for the GNSS module, which outputs NMEA sentences [47] at 9600 baud. The ESP32-S3 [16] provides three hardware UART controllers, allowing the GNSS module to occupy a dedicated serial port. The selected ToF LiDAR rangefinder for L.I.N.K. is the PTYS-12X module, which also interfaces over UART/TTL. As a result, the firmware reserves a second UART interface for the LiDAR subsystem. This arrangement keeps both serial-output sensors on dedicated point-to-point links, simplifies integration, and avoids placing the rangefinder on the shared I²C bus.

SPI is supported by the BMP280 as an alternative to I²C, but since the barometer already shares the I²C bus with the IMU and OLED, adding a separate SPI interface would require additional GPIO pins without providing a performance benefit at the current polling rate.

3.7.3.3 Final Choice of Driver Strategy

We decided to use vendor-provided libraries for the BNO055, BMP280, GT-U7, and SSD1306 in the current hardware revision, giving higher priority to development speed and verification of

correct behavior than to custom low-level implementations. For the PTYS-12X LiDAR module, the firmware uses a dedicated UART-based driver integrated into the same modular driver structure as the other peripherals. Each driver is placed into its own source-file pair, and the main loop interacts with the subsystem through typed read functions. This approach keeps the firmware modular and makes future sensor replacement possible without requiring major changes to the rest of the application. Compile-time feature flags are also included so that individual modules can be disabled during bring-up or hardware debugging, with stub data provided when needed.

3.7.4 Data Packaging and Messaging

The firmware should send sensor telemetry and waypoint data back to the phone over BLE. The serialization and transmission strategy will affect the payload size and the complexity of parsing the data on the phone side. These considerations will also affect the ease of debugging the system and its effectiveness at capturing waypoints.

3.7.4.1 JSON vs. Binary Serialization

JSON (JavaScript Object Notation) [12] is a text format that is human-readable and is natively parsed by the JavaScript `JSON.parse()` function. We can also serialize it to the ESP32 using the `ArduinoJson` library. JSON is easy to see in the debugger, and there is no need to negotiate a schema with the phone. But the payload is larger than with a binary format.

Binary encoding formats such as Protocol Buffers (protobuf) or CBOR encode data in a more compact form than JSON. Unfortunately, this means that one has to write the .proto file and then generate the C++ and TypeScript bindings using the protocol compiler, which is quite a non-trivial step in the build process, and somewhat of a synchronization issue. CBOR is more compact than JSON but less user-friendly, which also makes it harder to perform packet inspection on BLE during development. The size savings from switching to a more compact encoding format are not large enough to be relevant for most L.I.N.K. telemetry payloads smaller than 500 bytes, nor for pin payloads smaller than 1 K.

3.7.4.2 Delivery Strategy: Fire-and-Forget vs. ACK/Retry

Continuous Telemetry (heading, position, altitude) should be sent as fire-and-forget BLE notifications. The fact that a frame might be missed matters less than in other use cases (where the receiver might actually care to receive the missing data point), since the next frame is sent immediately at 5 Hz.

Waypoint pin payload reliability is important here because each pin represents a single instant in time captured when the ping button is pressed. There are a couple of approaches to achieve this: fire-and-forget with client-side retry. If the phone detects that any pins have been lost, it will resend them. Firmware side queue with acknowledgment. The latter approach is a better option. It is necessary for the phone to be connected to measure a pin, but it should not be necessary to be connected to capture the pin. The user still has to press the ping button to actually trigger the measurement, but if the BLE connection is lost at that time, the ping is simply held locally in the ESP32 queue and transmitted when the connection comes back up.

3.7.4.3 Final Choice of Data Packaging Strategy

JSON-over-BLE was selected for all payloads, with a firmware-side queue-ACK-retry pipeline for waypoint pins. JSON was chosen over binary protocols for its ease of debugging and zero-configuration parsing on the phone. The ACK mechanism ensures end-to-end reliability: the firmware retains each pin in a local queue until the phone writes back a confirmation, closing the loop from capture to server storage.

3.8 Software Stack (Web + App)

The software stack of the L.I.N.K. project is composed of a phone-side mapping interface, a server-side API, and a data storage layer. The phone application receives waypoints and telemetry data from the ESP32 via BLE, displays them on an interactive map, and sends the received waypoints to the server for long-term storage. If the phone user is in a place with no mobile coverage, or the BLE link between the binoculars and the phone drops, the system should first rely on offline functionality using local storage and sync data only when connectivity is re-established.

3.8.1 Development Language Comparison

The choice of development language affects both the phone-side application and the server-side API. The phone application must run in a mobile web browser with access to the Web Bluetooth API [48], geolocation [49], and IndexedDB [50] for local storage. The server must accept HTTP requests [15], validate incoming data, and write to a database. The main evaluation criteria were: compatibility with the Web Bluetooth API, availability of mapping libraries, type safety for complex payload structures, and the ability to share data type definitions between the phone and server.

3.8.1.1 JavaScript / TypeScript

JavaScript is the programming language of the web and is run on the client side by modern web browsers. TypeScript [51] is a typed superset of JavaScript that adds static type checking and additional features, such as interfaces and enums. Our L.I.N.K. project uses a complex JSON payload between the ESP32 and the phone that can include nested objects, lists, or primitive types, and it should be validated for each type (int, string, etc.) to ensure it is used correctly. The payloads have been defined as TypeScript interfaces (TelemetrySnapshot, PinPayload, ModuleStatus) to have validation done by the firmware compiler so that the unexpected JSON values that are emitted by the firmware will not be unexpected values that are expected by the phone app, and thus reducing the number of bugs in the application.

TypeScript also supports the full Web Bluetooth API thanks to the @types/web-bluetooth type definitions. We can also use all major mapping libraries (MapLibre GL [52], Leaflet [53], Google Maps) with TypeScript. So, we can develop fully-fledged applications with TypeScript on the client side (React [54] application in the browser) and on the server side (Node.js [55] with

Express [56]). This way, we can share type definitions and validation logic between both sides of the application.

3.8.1.2 Python

Python is widely used for backend web development with frameworks such as Flask and Django. It provides excellent support for data processing, scientific computing, and rapid API prototyping. For the server-side component of L.I.N.K., Python would be a viable choice, as it can easily handle incoming HTTP requests, validate JSON payloads, and interact with databases.

However, Python cannot run in the browser. The phone-side application must be written in JavaScript or TypeScript, regardless, meaning that choosing Python for the server would result in two different languages across the stack. This introduces the overhead of maintaining separate type definitions for the waypoint payload on both sides, increasing the risk of desynchronization between the firmware's JSON output, the phone's parsing logic, and the server's validation rules. For a small team with one primary software developer, maintaining a single language across the entire stack reduces this risk.

3.8.1.3 PHP

PHP is a server-side scripting language commonly used with the LAMP stack (Linux, Apache, MySQL, PHP). It has a large ecosystem of frameworks and is well-suited for traditional server-rendered web applications. PHP can handle HTTP request processing and database interactions for the L.I.N.K. server.

The limitations of PHP for this project are similar to those of Python: it cannot run in the browser, it does not share type definitions with the client, and its ecosystem is less aligned with modern real-time web applications. Additionally, the PHP ecosystem has less community support for BLE-related web development patterns and progressive web application (PWA) tooling compared to the JavaScript/Node.js ecosystem.

3.8.1.4 Comparison Summary (Table)

TABLE XXX: Pros and Cons for Development Language Selection

Criteria	JavaScript / TypeScript	Python	PHP
Browser Compatibility	Native (runs in browser)	Not supported	Not supported
Type Safety	Strong (TypeScript)	Optional (type hints)	Weak
Web Bluetooth Support	Full	N/A (server only)	N/A (server only)
Server Capability	Yes (Node.js + Express)	Yes (Flask / Django)	Yes (Laravel / Slim)
Shared Type Definitions	Yes (same language both sides)	No (separate definitions)	No (separate definitions)
Mapping Library Support	Excellent (MapLibre, etc.)	N/A (server only)	N/A (server only)
Community / PWA Tooling	Excellent	Good (server)	Moderate
Fit for L.I.N.K.	Best fit	Viable for server only	Less ideal

3.8.1.5 Final Choice of Development Languages

The selected software language ecosystem for L.I.N.K. is JavaScript/TypeScript. The phone-side application is implemented in TypeScript so that complex waypoint and telemetry payloads can be described with explicit interfaces and validated consistently throughout the user interface. This is especially useful for Web Bluetooth data, local storage objects, and map-facing records, where structured payloads reduce the chance of integration bugs.

On the backend, the server is built in Node.js with modern JavaScript modules and Express. This still preserves a single web-language ecosystem across the stack while keeping the backend lightweight and easy to iterate on during prototyping. As a result, L.I.N.K. benefits from a unified JavaScript-based workflow while still using TypeScript, where its type safety provides the greatest benefit on the client side.

3.8.2 Application Type Comparison

Before selecting the web stack for the L.I.N.K. software, it is necessary to determine what type of user-facing application best fits the system requirements. The companion application must receive waypoint and telemetry data from the ESP32, display that data on an interactive map, store records locally when connectivity is unavailable, and continue functioning in the field without depending entirely on constant cellular service. At a high level, the main options considered were a traditional web application, a native mobile application, and a Progressive Web Application (PWA). Each option offers different tradeoffs in deployment, device integration, offline capability, and development complexity.

3.8.2.1 Traditional Web Application

A traditional web application runs entirely in the browser and is typically accessed through a standard website URL. One major advantage of this approach is accessibility. Users do not need to install software through an app store, and updates can be deployed centrally without requiring manual downloads. Development is also relatively streamlined because the same application can run across many devices with minimal platform-specific work.

The main drawback is that a traditional web application is generally less integrated with mobile-device features than a more app-like solution. While it can still provide a responsive interface and mapping functionality, a standard browser-based website may offer weaker offline behavior and a less seamless field experience. For L.I.N.K., this is a significant limitation because waypoint capture must remain usable even when internet connectivity is poor or unavailable.

3.8.2.2 Native Mobile Application

A native mobile application is developed specifically for a mobile operating system such as Android or iOS. The main advantage of this approach is strong hardware and operating-system integration. Native applications typically offer the most direct access to device features and can provide a polished user experience with strong performance and background behavior.

However, native development introduces greater implementation complexity. Separate develop-

ment workflows may be required for different mobile platforms, increasing the overall software burden on the project team. Distribution is also more involved, since native applications usually depend on installation and platform-specific packaging. For a senior design prototype, this can introduce more overhead than is warranted by the application's scale.

3.8.2.3 Progressive Web Application (PWA)

A Progressive Web Application (PWA) [13], [14] combines characteristics of both web and mobile applications. Like a traditional web application, it is built with standard web technologies and can be accessed through a browser without requiring full app store deployment. At the same time, it also provides a more app-like experience through features such as installability, offline caching, local data storage, and persistent application behavior.

For L.I.N.K., the PWA model is especially attractive because it provides a hybrid balance between accessibility and functionality. The application can function as a mobile companion app while remaining easier to develop and deploy than a fully native solution. In addition, offline-first behavior can be supported via service workers and local storage, allowing waypoint records to remain available when the phone temporarily loses connectivity. This aligns closely with the system's field-use requirements.

3.8.2.4 Comparison Summary (Table)

TABLE XXXI: Pros and Cons for Application Type Selection

Criteria	Traditional Web Application	Native Mobile Application	Progressive Web Application
Accessibility	High	Moderate	High
Installation Requirement	None	Required	Optional install
Offline Capability	Limited	Strong	Strong
Device Integration	Moderate	Best	Good
Cross-Platform Development	Strong	Weaker	Strong
Development Complexity	Low	High	Moderate
Fit for L.I.N.K.	Acceptable	Strong but heavy	Best fit

3.8.2.5 Final Choice of Application Type

After comparing the available application models, the Progressive Web Application (PWA) approach was selected for L.I.N.K. The main reason is that it provides a hybrid balance between the accessibility of a web application and the practical usability of a mobile application. A PWA avoids the added complexity of maintaining a fully native mobile app while still supporting app-like behavior that is valuable for field operations.

This choice is especially important because the L.I.N.K. companion application must continue operating when connectivity is limited. A PWA can support offline caching, local storage of waypoint records, and synchronization upon reconnection, directly supporting the project's way-

point workflow. For these reasons, the PWA model provides the best overall balance of usability, portability, and implementation effort for the first prototype.

3.8.3 Web Stack Technology Comparison

The web stack determines the combination of tools used to build the phone application, the server, and the interface between them. Because L.I.N.K. requires a responsive, map-centric mobile interface with BLE connectivity, the phone application must be either a native mobile app or a progressive web application (PWA). A PWA was chosen over a native app because it runs on any device with a Chromium-based browser (which supports Web Bluetooth), requires no app store distribution, and can be developed with standard web technologies. The server must expose a REST API for waypoint storage and retrieval.

3.8.3.1 LAMP Stack

The LAMP stack (Linux, Apache, MySQL, PHP) is one of the oldest and most established web development stacks. It is well-documented, widely deployed, and suitable for server-rendered applications. Apache serves static files and proxies dynamic requests to PHP, which handles business logic and database queries.

For L.I.N.K., the LAMP stack presents several limitations. The phone application is a single-page application (SPA) with client-side rendering, real-time BLE data processing, and interactive map rendering, none of which benefit from PHP's server-side rendering model. Additionally, the LAMP stack does not share a runtime with the client, meaning payload types must be maintained separately in PHP and TypeScript/JavaScript.

3.8.3.2 MERN Stack

A MERN stack (MongoDB, Express, React, Node.js) is a full JavaScript stack for web development. In this environment, React is used to build User Interfaces from reusable UI components, using a declarative programming paradigm. REST APIs are built on top of Node.js and Express. MongoDB is a document-oriented database that natively stores JSON-like documents.

The MERN stack seems to be a good fit for our project. We'll use the React library to implement the map and real-time telemetry views and to manage BLE connections between the phone and the GPS device. The API server will be a basic Express application. Everything is JavaScript/TypeScript, so we'll be able to share type definitions between the front and back ends. The MongoDB document database model is also very close to our waypoint payload format: we already have all the data in JSON objects.

3.8.3.3 React + Express + SQLite Stack

A React + Express + SQLite stack combines a modern single-page frontend with a lightweight REST backend and an embedded relational database. In this architecture, React builds the phone-side user interface, Node.js with Express exposes the HTTP API, and SQLite stores waypoint records in a structured relational format without requiring a separate database server process.

For L.I.N.K., this approach is attractive because the waypoint schema is relatively stable, the expected number of users is small, and the prototype benefits more from simple deployment and direct debugging access than from enterprise-scale database features. Compared with MongoDB, SQLite preserves a more structured schema for waypoint records. Compared with PostgreSQL, SQLite reduces bring-up complexity by removing the need to manage a separate database service. The primary tradeoff is that SQLite offers less scalability and fewer advanced geospatial features than a full database server, but those limitations are acceptable for the prototype scope of L.I.N.K.

3.8.3.4 Comparison Summary (Table)

TABLE XXXII: Pros and Cons for Web Stack Technology Selection

Criteria	LAMP Stack	MERN Stack	React + Express + SQLite
Client Framework	None (server-rendered)	React (SPA)	React (SPA)
Server Runtime	PHP on Apache	Node.js + Express	Node.js + Express
Database	MySQL (relational)	MongoDB (document)	SQLite (embedded relational)
Language Consistency	PHP server / JS client	Full JavaScript/TypeScript	Shared web-language ecosystem
Geospatial Queries	Limited	Limited (manual)	Limited / manual
Schema Flexibility	Rigid (SQL)	Flexible (schemaless)	Structured, lightweight
Real-time / BLE Fit	Poor (server-rendered)	Good (SPA + shared types)	Good (SPA + shared web stack)
Fit for L.I.N.K.	Not ideal	Good fit	Best practical fit

3.8.3.5 Final Choice of Web Stack

The selected web stack for L.I.N.K. consists of a React [54] and TypeScript [51] frontend built with Vite [57], paired with a Node.js [55] and Express [56] backend and a SQLite [58] database. This combination preserves the advantages of a modern single-page application while keeping the backend lightweight and easy to deploy for a prototype-oriented system.

React was selected for the phone-side interface because it supports a responsive map-centric layout, live telemetry display, and BLE-driven UI updates. Vite provides a fast frontend development workflow and a straightforward production build process. On the backend, Express exposes the REST API used for waypoint ingestion and retrieval, while SQLite provides a structured relational store for waypoint records without requiring a separate database server. This architecture better fits the current scope of L.I.N.K. than heavier database-backed stacks, as it minimizes infrastructure complexity while still supporting reliable waypoint storage and retrieval.

3.8.4 Server Hosting

The server hosting architecture determines where the L.I.N.K. backend API and database are deployed, how the phone application reaches them, and how difficult the system is to maintain during development and testing. At a high level, the backend can be deployed either on a cloud-hosted server or on a locally self-hosted server. Each option has tradeoffs in accessibility, cost, setup effort, scalability, debugging convenience, and long-term maintainability. Because L.I.N.K.

is a prototype-oriented system with a relatively small user base and a narrow application scope, the server hosting decision should prioritize practical deployment considerations over enterprise-scale requirements.

3.8.4.1 Cloud-Hosted Server

A cloud-hosted server places the application backend on rented or managed infrastructure accessible via the public internet. In this model, the backend API and database run on remote hardware rather than on a machine directly controlled by the development team. One major advantage of cloud hosting is accessibility. The server is generally reachable from anywhere with internet access, which makes demonstrations, testing, and remote use more straightforward. In addition, cloud-hosted systems are often easier to scale if the number of users, devices, or stored waypoints increases significantly.

Cloud hosting can also simplify some aspects of public deployment. The hosting provider typically handles core infrastructure concerns such as external reachability, public DNS association, and basic uptime management. This can reduce the amount of manual networking configuration the development team must perform.

The main drawbacks of cloud hosting are recurring costs, greater deployment overhead, and reduced direct control over the hardware environment. For a project whose primary goal is validating the waypoint pipeline rather than serving a large production user base, that added infrastructure can increase system complexity without improving core functionality.

3.8.4.2 Local Self-Hosted Server

A local self-hosted server runs the backend API and database on hardware directly managed by the team. In this model, the server can be hosted on a desktop computer, laptop, mini-PC, or other local machine. One of the main benefits of this approach is control. The team has direct access to the backend environment, which simplifies logging, debugging, file inspection, database resets, and rapid configuration changes during development. This is especially useful for an iterative prototype in which the software stack, payload schema, and synchronization behavior may still evolve over time.

A self-hosted design can also reduce recurring costs by avoiding the need for paid cloud infrastructure during the prototype stage. For a system like L.I.N.K., where the expected number of users is small and the volume of stored data is relatively modest, a local backend can be sufficient from a performance standpoint.

The disadvantages are primarily related to public accessibility and deployment effort. Unlike a cloud-hosted system, a local server is not automatically available to external users over the internet. Making a self-hosted server accessible from outside the local network may require additional setup steps, such as configuring firewalls, using reverse proxies, tunneling, or other networking tools. In addition, server availability depends on the local machine remaining powered on, connected to the internet, and properly configured. This makes self-hosting less convenient from an infrastructure standpoint, even if it remains attractive for direct control over development.

3.8.4.3 Comparison Summary (Table)

TABLE XXXIII: Pros and Cons for Server Hosting / Deployment Selection

Criteria	Cloud-Hosted Server	Local Self-Hosted Server
Public Accessibility	Strong, internet-ready by default	Weaker by default, requires added setup
Infrastructure Cost	Recurring / ongoing	Low or none
Initial Deployment Effort	Moderate	Moderate to high for external access
Hardware Control	Limited	Full control
Debugging Convenience	More indirect	Very direct
Scalability	Better for growth	More limited
Uptime Dependence	Provider-managed	Depends on local host
Fit for Prototype Development	Good	Very good

3.8.4.4 Final Choice of Server Architecture

After comparing the two approaches, the selected hosting architecture for L.I.N.K. is a locally self-hosted server. Although this approach may require more manual setup to make the backend externally accessible, it better aligns with the project's priorities. The L.I.N.K. system is a prototype-focused design with a limited user base, relatively small waypoint payloads, and a strong need for rapid debugging and direct backend control during development.

A cloud-hosted server would provide stronger public accessibility and a cleaner path toward large-scale deployment, but those advantages are less critical for the current scope of the project. In contrast, self-hosting allows the team to work more directly with the backend environment, inspect and modify the data layer more easily, and avoid the recurring cost and infrastructure overhead associated with cloud deployment. For these reasons, the local self-hosted approach provides the best balance between practicality, control, and prototype-stage system requirements. In the implemented prototype, the self-hosted server runs the Node.js/Express API together with the SQLite data layer, and it can also serve the built frontend assets in production from the same host.

3.9 Optical Stack

3.9.1 Optical Stack Overview

The optical stack of the L.I.N.K. system consists of a compact binocular-style telescope combined with an optical injection path for the HUD display. The telescope portion provides magnification and target acquisition, while the HUD subsystem overlays telemetry information such as range, bearing, and system status into the user's field of view. The optical stack, therefore, serves two main functions: providing a clear magnified image of the target and injecting a readable display into the optical path without significantly degrading image quality.[59], [60], [61]

The optical layout includes an objective lens, an erecting prism system, and an eyepiece assembly. Light from the target enters the objective lens and is focused into an intermediate image plane. The prism assembly both folds the optical path and erects the image so that the user perceives a correctly oriented scene. The eyepiece then magnifies this intermediate image for comfortable viewing. A partially reflective optical combiner or beamsplitter is used to inject the HUD display into the optical path, allowing the user to simultaneously observe the environment and the system telemetry.

3.9.2 Optical Stack Requirements

The optical design must satisfy several requirements to ensure that the device functions effectively as a handheld ranging instrument. The system is designed to achieve approximately 5× magnification with a true field of view of approximately 4.5 degrees, corresponding to an apparent field of view of roughly 22.5 degrees. These values provide a balance between magnification and situational awareness,[60], [62] allowing the user to observe distant targets while still maintaining sufficient context of the surrounding environment.

In addition to magnification and field-of-view constraints, the optical system must remain compact and lightweight to maintain the ergonomics of a handheld device. The optical stack must also accommodate the injection of the HUD display without introducing excessive optical losses or ghost reflections. The design must therefore consider optical alignment tolerances, prism size, and component placement to maintain image clarity and brightness while supporting the additional HUD functionality.[59]

3.9.3 Telescope Type Selection

3.9.3.1 Galilean Telescope

A Galilean telescope consists of a positive objective lens and a negative eyepiece lens. This configuration produces an upright image without requiring additional image-erecting optics. Because of this simplicity, Galilean telescopes are compact and lightweight,[59] which makes them attractive for applications such as opera glasses and small handheld optical devices.

However, Galilean systems suffer from a limited field of view and relatively poor edge performance. The exit pupil is located inside the optical system rather than outside it, which can make eye placement more difficult and reduce viewing comfort. Additionally, the lack of an internal focal plane makes it difficult to integrate additional optical elements such as reticles or HUD overlays. For a system that must integrate a display and maintain good usability, these limitations make the Galilean configuration less attractive.

3.9.3.2 Newtonian Telescope

A Newtonian telescope uses a concave primary mirror and a flat secondary mirror to redirect light to an eyepiece located at the side of the optical tube. This design is commonly used in astronomical telescopes [59] due to its ability to achieve large apertures with relatively simple optical surfaces.

Despite its advantages in large telescopes, the Newtonian configuration is not well suited for compact handheld optical devices. The geometry of the system requires a relatively long optical path and an offset eyepiece position, which would make the device bulky and ergonomically difficult to use. Additionally, integrating a HUD or compact binocular form factor into a Newtonian configuration would introduce significant mechanical complexity. For these reasons, a Newtonian telescope is not a suitable option for the L.I.N.K. optical system.

3.9.3.3 Keplerian Telescope

A Keplerian telescope uses two positive lenses: an objective lens that forms a real image and an eyepiece that magnifies this image. Unlike the Galilean system, the intermediate focal plane exists within the optical path[59], [60], which allows additional optical components such as reticles, prisms, or display combiners to be inserted. This configuration generally provides a wider field of view and better optical performance compared to a Galilean system.

The primary drawback of a Keplerian telescope is that the image formed by the objective lens is inverted. As a result, an additional optical element, typically a prism system, must be used to erect the image. While this adds complexity, prism systems are commonly used in binocular optics and also provide the benefit of folding the optical path into a more compact form. Because the Keplerian design provides good optical performance [60] and allows integration of additional optical elements, it is well suited for the L.I.N.K. system.

3.9.3.4 Comparison Summary (Table)

TABLE XXXIV: Pros and Cons for Telescope Type Selection

Criteria	Galilean Telescope	Newtonian Telescope	Keplerian Telescope
Image Orientation	Upright image (no prisms needed)	Inverted without additional optics	Inverted (requires prisms)
Field of View	Narrow FOV	Moderate FOV	Wide FOV
Optical Complexity	Very simple (two lenses)	Complex mirror alignment	Moderate (requires prisms)
Compactness	Very compact design	Long optical tube required	Compact with prism folding
Intermediate Image Plane	None (hard to add reticle/HUD)	Possible but inconvenient	Real image plane available
HUD / Reticle Integration	Difficult	Difficult for compact system	Easy integration at image plane
Mechanical Integration	Simple	Bulky configuration	Compatible with binocular optics
Fit for L.I.N.K.	Acceptable but limited	Not suitable	Best fit

3.9.3.5 Final Choice of Telescope Type

The selected telescope architecture for the L.I.N.K. optical system is the Keplerian telescope configuration. This choice provides the best balance between optical performance, field of view, and system integration. The presence of a real intermediate image plane allows additional optical components, such as the HUD injection optics, to be incorporated into the design. Although the inverted image produced by the Keplerian system requires the use of an image-erecting prism assembly, this requirement aligns well with standard binocular optical architectures. As a result,

the Keplerian configuration provides the flexibility and performance needed for the L.I.N.K. device.[60], [63]

3.9.4 Binocular Type Selection

3.9.4.1 Roof System

Roof prism binocular systems use a set of prisms arranged so that the optical paths remain nearly collinear with the objective lenses. This configuration results in a straight, compact form factor, which is commonly seen in modern consumer binoculars. Roof systems are mechanically robust and allow the optical barrels to remain aligned in a narrow configuration, which can make the device easier to hold and transport.

However, roof prism systems require very precise manufacturing [64], [65] and often require phase-correcting coatings to maintain image quality. These coatings and alignment requirements increase cost and complexity. Additionally, the optical path within a roof prism system is less accessible for integrating additional optical elements such as display combiners or beam splitters. These factors can make the integration of a HUD more difficult.

3.9.4.2 Porro System

Porro prism binocular systems use two right-angle prisms arranged to offset the optical path. This configuration produces an erect image while also folding the optical path [64], [65] into a compact layout. One advantage of the Porro design is that it generally provides excellent image brightness and depth perception [64] due to the wider spacing between the objective lenses.

Porro systems are also simpler to manufacture and typically require fewer specialized coatings compared to roof prism systems. Because the optical path is more accessible and the prism geometry is straightforward, Porro systems are often easier to modify or integrate with additional optical components. For a prototype device that requires integration of a HUD element and optical testing flexibility, this configuration provides practical advantages.

3.9.4.3 Comparison Summary (Table)

TABLE XXXV: Pros and Cons for Binocular Prism System Selection

Criteria	Roof Prism System	Porro Prism System
Form Factor	Straight, compact, inline design	Offset optical path, wider body
Image Quality	Good, requires phase coatings	Excellent brightness and contrast
Manufacturing Complexity	High precision required	Simpler prism geometry
Cost	Higher (coatings + alignment)	Lower cost
Optical Path Accessibility	Difficult to access/modify	Easy to access and modify
HUD Integration	Possible but constrained	Easier integration of combiner optics
Alignment Tolerance	Tight tolerances required	More forgiving alignment
Depth Perception	Standard binocular spacing	Enhanced depth perception (wider spacing)
Fit for L.I.N.K.	Good for compact final product	Best for prototype and integration

3.9.4.4 Final Choice of Binocular Type

For the L.I.N.K. prototype, a Porro prism binocular architecture was selected. The Porro system offers strong optical performance, relatively simple manufacturing requirements, and greater flexibility for integrating additional optical elements such as the HUD combiner. The wider spacing between objective lenses also improves depth perception and overall viewing comfort.[64]

Additionally, the Porro configuration provides a practical mechanical structure for a prototype device, allowing easier alignment and modification during testing. While roof prism systems may provide a more compact final form factor, the Porro design offers the optical simplicity and integration flexibility needed for the initial development and testing of the L.I.N.K. system.

3.9.5 Optics Selection

The optical subsystem of L.I.N.K. is designed to provide magnified observation of a point of interest while simultaneously overlaying system telemetry through the HUD. The optical path shown in Figure X begins with visible light entering the system through the primary objective lenses and proceeds through a sequence of magnification and image-conditioning elements before reaching the user's eye. Along this path, several optical components perform distinct roles including image formation, image erection, field conditioning, and display injection.

3.9.5.1 Objective Lens System

The first elements in the optical path is the achromatic objective lens. The objective collects incoming light from the external scene and forms an image that will be processed [59], [63] by the remainder of the optical stack. In the L.I.N.K. system, the objective focal length is selected to support the required 5× magnification [60] when paired with the eyepiece optics. This was designed to be +160 mm. The objective must also provide sufficient aperture to maintain brightness while remaining compact enough for a handheld optical device. The lense is 40 mm in diameter to accomplish the necessary brightness.

This element establishes the initial image geometry and prepares the light for propagation through the internal prism system.

3.9.5.2 Focus Extender

An plano-concave optical extender is placed before the prism system. The extender lens has focal length of -100 mm, and diameter of 25 mm. This element maintains the appropriate spacing between components [60] so that the intermediate focus is not inside of the Porro Prism system, and falls passed it. This will allow for the inclusion of a field lens.

3.9.5.3 Porro Prism Assembly

Following the Focus Extender, the light passes through a Porro prism pair. The prism assembly performs two important functions within the optical system. First, it erects the image that would otherwise appear inverted in a Keplerian telescope configuration. Second, it folds the optical

path [64], [65] into a more compact geometry, allowing the magnification system to fit within the physical constraints of the binocular housing, which is 25 mm diameter of prism surface area.

The Porro prism configuration also increases the effective separation between the objective lenses, which improves depth perception [64] and overall viewing comfort. Because the prism geometry is relatively simple compared to alternative prism systems, it provides strong optical performance while maintaining alignment stability within the device enclosure.

3.9.5.4 Field Lens

After the prism assembly, a field lens is used to stabilize the intermediate image and control the system's field of view. [62] This plano-convex lens brings in far field angles, is 25 mm diameter, and has focal length of +60 mm. The field lens ensures that the magnified image is properly centered within the eyepiece and helps maintain consistent illumination across the viewing field. This component also improves the optical efficiency [60] of the system by guiding rays toward the eyepiece pupil.

3.9.5.5 HUD Optical Injection

In one of the optical channels, a partially reflective beam splitter [61] is introduced to inject the HUD image into the optical path. This 35 mm x 35 mm beam splitter transmits the majority of the incoming scene light while reflecting light originating from the OLED display subsystem. [66]The result is a combined image consisting of the external environment and the projected telemetry information.

Light from the OLED display is directly incident onto the beam splitter. Since the distance is fairly short, the eye piece acts as a 1 F lens system magnifying the OLED display into the optical path. The OLED appears aligned with the external scene when viewed through the eyepiece. The geometry of this injection path is designed so that the display overlay remains stable and readable without significantly reducing scene brightness.

3.9.5.6 Eyepiece Optics

The final optical element in the viewing path is the eyepiece lens. The eyepiece magnifies the intermediate image [59], [63] formed earlier in the optical stack and presents it to the user with the desired apparent field of view. We have picked a biconvex, +50 mm focal length the eyepiece focal length is chosen so that the combined objective–eyepiece system achieves the required magnification [60] while maintaining comfortable eye relief.

The eyepiece must also accommodate the combined image produced by the scene and HUD injection system. Proper optical alignment ensures that the display overlay appears fixed relative to the external environment, allowing the user to observe telemetry data without losing focus on the target scene.

Together, these optical elements form the complete optical stack for the L.I.N.K. system. The sequence of objective optics, prism assembly, field conditioning elements, display injection

components, and eyepiece optics produces a magnified image of the external scene with an integrated heads-up display that presents ranging and navigation data directly within the user's field of view.

3.10 Implemented Component and Software Revisions

The previous sections document the components and software approaches selected during the initial research phase. As L.I.N.K. moved into PCB fabrication, firmware bring-up, and full-system integration, several parts were changed because of component availability, interface requirements, and results observed during testing. The original comparisons remain in this chapter as a record of the design process. The following paragraphs document the revisions used in the current prototype.

3.10.1 Digital Processing Revision

After the original ESP32 selection, the processing platform was finalized as the ESP32-S3-WROOM-2-N32R16V rather than a generic ESP32 module. The selected module provides 32 MB of flash memory, 16 MB of PSRAM, integrated Bluetooth Low Energy, and enough hardware interfaces to operate the sensor suite, two UART peripherals, the SPI display, the user controls, the buzzer, and battery monitoring. The final implementation places the ESP32-S3 module on the custom MCU PCB instead of retaining a complete development board inside the prototype.

This revision does not change the reasoning behind selecting an MCU over an FPGA or single-board computer. It narrows the original ESP32 choice to the exact device used by the firmware and PCB. The additional memory and peripheral flexibility provide more room for the BLE stack, display buffers, sensor drivers, waypoint queue, and future firmware expansion without changing the overall embedded architecture.

3.10.2 GNSS Revision

The GNSS subsystem was changed from the GT-U7 and NEO-7-class module described earlier to the u-blox NEO-M9N. The NEO-M9N provides a more modern multi-constellation receiver while maintaining a straightforward UART interface for the ESP32-S3. In the current firmware, the receiver operates on its own hardware UART at 38,400 baud and is parsed using the same NMEA-based workflow used elsewhere in the system.

The updated GNSS PCB also uses an external U.FL antenna connection rather than depending on the antenna arrangement of a generic breakout board. This allows the antenna to be positioned where it has a clearer view of the sky and can be kept farther away from the switching regulators and high-speed digital traces. The custom board includes the supporting RF path, filtering, and backup-power components required by the selected receiver. This change gives the team more control over antenna placement and makes the GNSS subsystem consistent with the custom-PCB requirement of the final prototype.

3.10.3 Barometric Sensor Revision

During firmware integration, the connected pressure sensor was updated to a BME280-compatible module. Pressure and temperature remain the measurements used by the waypoint and telemetry workflow, while the additional humidity channel is available but is not required by the current L.I.N.K. calculations. The change does not alter the decision to use a digital MEMS barometer; it only records the exact sensor library and device currently enabled in the firmware.

3.10.4 HUD Display Revision

The SSD1306 monochrome OLED originally selected for the HUD was replaced by a 1.8-inch, 128 by 160 ST7735 color TFT display. The TFT communicates over SPI instead of sharing the I²C bus with the IMU and barometer. The implemented firmware uses the TFT_eSPI library, operates the display through the ESP32-S3 HSPI interface, and maintains the former OLED code only as a disabled legacy path.

The TFT provides additional screen area, color, and layout flexibility for the completed HUD pages. The firmware currently supports six display pages and mirrors the image horizontally so that the information appears correctly after passing through the HUD optical path. This display change also required the MCU PCB to include a dedicated TFT connector and the necessary SPI, chip-select, data/command, and reset signals.

3.10.5 User Input Revision

The original rotary-encoder approach was replaced by three momentary tactile buttons. Two buttons move backward and forward through the HUD pages, while the third button performs the waypoint-ping action. The ping button must be held for approximately 1.25 seconds before the waypoint is accepted, which reduces accidental captures while the binoculars are being handled. An active buzzer provides short confirmation tones for a successful capture and a longer warning tone when a capture is rejected.

This arrangement was easier to package and test with the custom button PCBs while still providing direct control of every required HUD function. It also simplified the input firmware because page movement and waypoint confirmation can be handled as separate, clearly defined button events.

3.10.6 Power-Supply Revision

The implemented power architecture no longer uses the LTC3113 buck-boost regulator described in the original selection section. The latest prototype uses separate LM2596S switching-regulator boards for the 5 V and 3.3 V rails. Both regulators are supplied from the series-connected rechargeable 18650 battery pack. The 5 V rail supports the LiDAR and other loads that require the higher supply, while the 3.3 V rail powers the ESP32-S3, low-voltage sensors, display logic, and the remaining digital circuitry.

The LM2596S was selected for the fabricated revision because its reference circuit was easier to source, lay out, assemble, and troubleshoot within the project schedule. Using separate regulators

also prevents the higher-current 5 V load from sharing the same output stage as the lower-voltage sensor and MCU rail. The final 3.3 V and 5 V regulator PCBs use the same regulator family but have different feedback networks and output components for their required voltages. The LTC3113 discussion remains in this report to show the original design direction, while the LM2596S paragraphs document the hardware that replaced it.

3.10.7 Implemented Internal Communication Map

The final internal communication layout is a combination of I²C, UART, and SPI. The BNO055 IMU and BME280 barometer share the 400 kHz I²C bus. The NEO-M9N GNSS receiver and PTYS-12X LiDAR each use a dedicated hardware UART, which prevents the two continuous serial data streams from interfering with one another. The ST7735 TFT uses a separate SPI bus because its display updates require more bandwidth than the original I²C OLED. This implemented bus arrangement supersedes the earlier grouping that placed the OLED on the shared I²C bus.

3.10.8 Implemented Firmware Update

The current firmware is built in PlatformIO using the Arduino framework and C++ for the ESP32-S3. PlatformIO keeps the board configuration, memory settings, TFT pin definitions, and library versions in one reproducible project. The enabled drivers include the BNO055 IMU, BME280 barometer, NEO-M9N GNSS receiver, PTYS-12X LiDAR, ST7735 TFT, BLE server, button interface, buzzer, and waypoint queue. Feature flags allow individual modules to be disabled during board bring-up without removing the rest of the application.

The cooperative firmware architecture selected earlier was retained and expanded during implementation. Sensor data is serviced at approximately 20 Hz, while BLE telemetry is transmitted at approximately 5 Hz. Startup diagnostics include module self-tests and an I²C scan so that wiring or pull-up problems can be identified before normal operation. The ping workflow rejects stale or low-confidence sensor data, averages heading, pitch, range, and GNSS samples during the button-hold interval, computes the waypoint, and stores the resulting JSON payload in a local queue. Waypoints remain queued until the phone returns an acknowledgment, allowing the firmware to retry an interrupted BLE delivery without losing the captured point.

3.10.9 Implemented Web and Server Stack Update

The implemented companion application uses a React and TypeScript frontend built with Vite. It is configured as a Progressive Web Application, registers a service worker for offline application loading, and uses IndexedDB through the `idb` library to store local records and pending uploads. The phone receives telemetry and waypoint payloads through Web Bluetooth, updates the mapping interface, and retains unsent waypoint data locally until the server becomes reachable again.

The backend is a Node.js and Express application with SQLite storage through `better-sqlite3`. The SQLite database uses write-ahead logging and stores the observer position, calculated target position, range, bearing, pitch, LiDAR quality, labels, timestamps, and the original telemetry data.

The REST interface exposes the waypoint routes under `/api/pins` and a status route under `/api/health`. In production, the same Express process also serves the built React frontend, keeping the PWA and API on one host.

For deployment, the server runs on the team's Debian machine and is kept active with PM2. Tailscale Funnel provides the HTTPS endpoint needed for browser security requirements, including Web Bluetooth access from the phone. This architecture preserves the control and low recurring cost of the self-hosted approach while still allowing the application to be reached securely outside the local network.

3.10.10 Chapter 3 Revision Summary

The current L.I.N.K. prototype is therefore based on the ESP32-S3-WROOM-2, NEO-M9N GNSS receiver with an external antenna connection, BNO055 IMU, BME280 pressure sensor, PTYS-12X LiDAR, ST7735 TFT HUD, tactile-button controls, and separate LM2596S-based 5 V and 3.3 V regulator boards. The software implementation consists of cooperative PlatformIO firmware, JSON-over-BLE communication with acknowledgment and retry handling, a React and TypeScript PWA, an Express API, SQLite storage, and a self-hosted HTTPS deployment through Tailscale. Updated schematics, PCB renders, and fabricated-board photographs should use these implemented parts rather than the components shown in the original selection paragraphs.

4 Standards and Design Constraints

4.1 Standards

Engineering standards are very crucial to maintain while designing an electronic system; standards are important because they ensure universal compatibility, safety, reliability, and performance across different electronic devices. Some of the key components of L.I.N.K., including printed circuit boards (PCB), power systems, and circuit design, must all adhere to these standards. Maintaining these standards minimizes deviation from design requirements and ensures that manufacturing processes can be properly performed.

In regards to L.I.N.K., standards are applied to PCB design, lithium battery safety, software implementation, and communication protocols. These standards are important to component selection, layout, and the integration of all subsystems, which ultimately bring functionality to the final system.

4.1.1 Lithium Battery Standards

Lithium battery standards help address safety and handling requirements that are imperative for the safe and reliable use of lithium batteries in all use cases involving electronic systems. These standards are established by regulatory entities such as large industry organizations or government bodies. With regards to L.I.N.K., the project uses a single 18650 lithium-ion battery to power the entire system, which is regulated by these standards and allows users to operate the device safely.

The standout standards involving lithium batteries include: UL 1642, IEC 62133, and UN 38.3 ... standards [[67], [68], [69]]. The use of these standards ensures the selected battery meets all electrical, thermal, and mechanical safety concerns when the system is under normal and fault load conditions.

UL 1642 specifies safety requirements for individual lithium-ion cells, including protection against short circuit, overcharge, and mechanical abuse [67]. UL 1642 ensures that the 18650 lithium battery chosen to be used in L.I.N.K. is safe to use in an embedded system. IEC 62133 sets a list of guidelines that ensure safe integration of a rechargeable lithium battery into a portable electronic device such as L.I.N.K. The guidelines encompass electrical protection and consider environmental impacts [68]. L.I.N.K. will be used in a variety of different biomes and environments, therefore operation under all use cases is facilitated by these standards. UN 38.3 overviews the testing procedures that involve altitude simulations, thermal cycling, vibration, shock, and short circuit scenarios [69]. These standards are tested by the manufacturer, but the fully integrated system, L.I.N.K., will be tested against these standards.

L.I.N.K. considers safety concerns beyond just the standards that must be followed. L.I.N.K. does not include onboard charging circuitry and uses external cells that can be recharged separately; this helps reduce serious risk of fire or explosion. The use of name-brand 18650 cells ensures that L.I.N.K. adheres to manufacturer safety specifications. This minimizes the risks associated with lithium batteries and adheres to industry standards.

4.1.2 Communication Protocol Standards

The L.I.N.K. system relies on both wireless and wired communication standards to reliably transfer data between embedded peripherals, the ESP32-S3, and the phone-side companion application. Internally, several peripherals communicate with the microcontroller over a shared I²C bus, while externally the binocular device transmits telemetry and waypoint information to the phone over Bluetooth Low Energy (BLE). Because these interfaces are central to waypoint capture, HUD updates, and end-to-end data delivery, the design must follow well-defined communication standards to ensure interoperability, signal reliability, and practical integration.

4.1.2.1 Bluetooth Low Energy (BLE) Standard

Bluetooth Low Energy is the primary wireless communication method used in L.I.N.K. for transferring telemetry and waypoint payloads from the ESP32-S3 binocular unit to the phone-side companion application. The relevant governing document is the Bluetooth SIG *Bluetooth Core Specification* [4]. This specification defines the BLE physical layer, link layer, Generic Attribute Profile (GATT), advertising behavior, connection procedures, and attribute-based communication model used by BLE devices.

For L.I.N.K., the most relevant parts of the BLE standard are the advertising and connection model, GATT-based characteristic exchange, and low-power short-range operation. These features are well-suited to the project because the device transmits relatively small packets, such as telemetry snapshots and waypoint payloads, rather than large media streams. Following the BLE standard is important because the phone application must connect to the ESP32-S3 using standard BLE behavior supported by modern mobile platforms and browser-based BLE APIs.

A standards-compliant BLE design improves interoperability, reduces software integration risk, and ensures that notifications, writes, and acknowledgments are handled consistently between the firmware and the phone application.

BLE also aligns with the portable, battery-powered nature of L.I.N.K. Compared with higher-overhead wireless alternatives such as Wi-Fi, BLE is designed for low-power operation while still supporting the short-range waypoint-transfer workflow required by the system [4]. In L.I.N.K., BLE notifications are used for telemetry and waypoint transfer, while acknowledgment behavior is implemented at the application layer to prevent one-time waypoint captures from being lost during temporary communication interruptions.

4.1.2.2 I²C Bus Standard

Within the embedded hardware of L.I.N.K., the IMU, barometer, and OLED HUD display share a common wired communication bus. The relevant standard is NXP's *UM10204 I2C-bus specification and user manual* [5]. This standard defines the electrical and protocol-level behavior of the I²C bus, including its two-wire serial architecture, addressing model, acknowledgment behavior, clock synchronization, and supported operating modes.

The I²C standard is especially relevant to L.I.N.K. because multiple peripherals must coexist on the same bus without communication conflicts or signal integrity issues. The standard defines the use of the serial data line (SDA), serial clock line (SCL), pull-up resistors, and device addressing, all of which affect reliable sensor polling and display communication [5]. It also provides the basis for selecting appropriate operating speeds, such as Standard-mode or Fast-mode, depending on total bus capacitance and update-rate requirements.

Following the I²C standard helps guide practical implementation decisions in the firmware and hardware, including pull-up resistor sizing, bus frequency selection, and shared-bus stability. Because L.I.N.K. relies on repeated sensor reads and regular HUD updates, stable I²C behavior is necessary for maintaining consistent telemetry acquisition and display performance. For the current prototype, I²C is well suited because these peripherals exchange relatively low-bandwidth configuration and measurement data while benefiting from simple wiring and mature library support.

4.1.3 Software and Data Exchange Standards

In addition to hardware communication standards, the L.I.N.K. system depends on software-layer data exchange standards to move waypoint records between the phone-side PWA and the backend server. The two most relevant standards are RFC 8259, which defines the JavaScript Object Notation (JSON) data interchange format, and RFC 9110, which defines HTTP semantics [6], [7]. These standards are directly applicable because the implemented software stack uses JSON payloads transmitted over HTTP to synchronize waypoint data.

RFC 8259 defines JSON as a lightweight, text-based data interchange format built from structured objects, arrays, strings, numbers, booleans, and null values [6]. JSON is especially suitable for L.I.N.K. because it is human-readable, easy to debug, and natively compatible with the JavaScript/TypeScript stack used by the phone application and backend. The waypoint payloads

in L.I.N.K. contain structured fields such as identifiers, coordinates, range, heading, timestamps, and status values. Using JSON ensures that this information can be represented in a consistent, standards-based format across the software pipeline.

RFC 9110 defines the semantics of HTTP requests and responses, including methods such as GET and POST, response codes, and the meaning of message exchanges between clients and servers [7]. In L.I.N.K., the phone-side PWA uses HTTP to submit waypoints records to the backend server and retrieve stored data for rendering on the map. Following HTTP semantics ensures that requests and responses are handled in a predictable and interoperable way. Together, JSON and HTTP provide the standards-based software interface between the phone application and the backend, allowing locally captured waypoints to be synchronized using a widely supported and maintainable approach [6], [7].

4.1.4 Laser Classification Standard

When designing any LiDAR based system, there are several considerations regarding laser safety that must be followed. This is especially true with a ToF LiDAR module where the range at which people can be susceptible to laser is longer than the detectable range of the module. Furthermore, the range at which ToF LiDAR can detect is directly related with the output power of the laser. As such, we must understand the standard protocols for laser safety and ensure that the design of our device follows these protocols.

The standard protocol for defining the safety class is the International Electrotechnical Commission (IEC) standard IEC 60825-1 which uses a measure of the Acceptable Emission Limit (AEL), derived from the Maximum Permissible Exposure (MPE) to the human eye or skin. These limits depend on wavelength, exposure duration, beam geometry, and viewing conditions. Hence, the classification is based on hazard under foreseeable use, rather than pure output power. This standard breaks lasers down into four defined safety classes: Class 1, Class 2, Class 3, and Class 4.

4.1.4.1 Class 1 Laser

A Class 1 laser is defined as a laser system whose accessible emission does not exceed the Maximum Permissible Exposure (MPE) for any relevant wavelength and exposure duration in foreseeable operation conditions. This means that the laser is considered intrinsically safe for continuous viewing, including direct beam exposure to the eye or skin, without the need for protective measures. Importantly, Class 1 designation applies to the accessible radiation, not necessarily the internal laser power; thus, high-power lasers may still be classified as Class 1 if they are fully enclosed or engineered such that no hazardous radiation is accessible during normal use.

Similarly to normal Class 1 lasers, a subclass known as Class 1M lasers follow all of the same standard. However, the key difference for Class 1M laser's is that they are deemed hazardous when viewed through viewing optics such as magnifiers, binoculars, telescopes, and microscopes that can collect and focus additional optical power to the human eye. This often applies to lasers with large beam diameters or high beam divergence where collection optics can increase the irradiance beyond the MPE of the retina.

4.1.4.2 Class 2 Laser

A Class 2 laser is defined as a laser that emits lights in the visible wavelength range between 400-700 nanometers and does not exceed the Class 2 AEL. These lasers are considered eye safe for accidental or momentary viewing because of the human blink reflex. Typically, the assumed exposure limit is 0.25 seconds. However, deliberate and prolonged staring into the beam can exceed the retinas MPE and pose a hazard to the viewers vision. Class 2 lasers therefore rely on both emission limits and the assumption of human response for situations where the user is not wearing safety eye wear.

Similarly to the Class 1 structure, a Class 2M laser has all the same standards as Class 2, but is deemed hazardous when viewed with collecting and magnification optics. In this case, the exposure limit of 0.25 seconds no longer holds and the blink reflex may not be enough to protect the eye from damage.

4.1.4.3 Class 3 Laser

Class 3 is broken down into two subclasses, Class 3R and 3B. A Class 3R laser is defined as a laser whose acceptable emission exceeds the AEL for Class 2 or Class 1, but remains restricted to a level where the risk of injury from direct exposure is low under foreseeable conditions. This typically corresponds to an output power of 5 mW, where the beam can exceed the MPE but has an inherently lower risk than higher classifications like Class 3B and Class 4 lasers. Unlike Class 2, Class 3R lasers are not solely reliant on the blink reflex as the protective measure for the classification. In this case, stricter control measures are used as well as emphasis on avoiding direct eye exposure with the laser.

A Class 3B laser is defined as a laser that exceed the Class 3R AEL but does not exceed the Class 4 AEL, and is therefore hazardous to the eye under direct exposure. The radiation can easily exceed the MPE for the eye meaning that even brief exposure can cause injury, particular to the retina at visible and near infrared wavelengths, and to the cornea at longer infrared wavelengths. In this case, diffuse reflections are normally not hazardous, with much of the risk coming from direct or specularly reflected beams. For this reason, Class 3B laser required controlled use conditions, including restricted access to the beam, appropriate safety training, and the use of wavelength specific protective eye wear. These systems also use administrative and engineering controls to assist with maintain safe conditions during operation.

4.1.4.4 Class 4 Laser

A Class 4 laser is defined as a laser whose acceptable emission is above the Class 3B AEL, making it hazardous to both eye and skin exposure, as well from specular and diffuse reflections. At this level, emitted radiation far exceeds the MPE and therefore even indirect exposure from scattered light can be hazardous, especially at higher power densities. Class 4 lasers are also considered fire hazards, especially when in contact with combustible or absorptive materials. For this reason, Class 4 lasers require strict control measures, including engineered safeguards like beam enclosures, interlocks, safety procedures, and the mandatory use of wavelength specific protection.

4.1.5 Lens Standards

ISO 10110 is an international standard titled “Optics and optical instruments — Preparation of drawings for optical elements and systems.” The standard provides a consistent method for documenting optical components in technical drawings so that designers, manufacturers, and inspectors interpret specifications in the same way. ISO 10110 defines standardized symbols and notation used to describe key optical parameters such as surface form, curvature, material type, tolerances, and coatings. By establishing a structured system for communicating these characteristics, the standard reduces ambiguity during manufacturing and helps ensure that optical components meet their intended performance requirements [70].

In optical system design, accurate documentation is critical because even small deviations in surface shape, alignment, or material quality can significantly affect system performance. ISO 10110 drawings allow designers to clearly specify properties such as surface irregularity, centering tolerances, allowable defects, and coating requirements. These parameters are essential for controlling aberrations, minimizing reflection losses, and maintaining consistent optical transmission. Using a standardized format also simplifies communication between optical designers and fabrication facilities, allowing manufacturers to produce components that closely match the intended design specifications [70].

For the L.I.N.K. system, ISO 10110 provides a framework for documenting the lenses and optical components that form the imaging path. Parameters such as lens diameter, effective focal length, surface tolerances, and coatings can be formally defined in ISO 10110-compliant drawings. Because the binocular system must provide a clear and stable image while integrating a heads-up display, maintaining precise optical alignment and surface quality is essential. Applying ISO 10110 standards ensures that these requirements are communicated clearly to manufacturers and collaborators, supporting reliable fabrication and consistent optical performance in the final L.I.N.K. device [70].

4.2 Design Constraints

4.2.1 Power Constraints

The total system current is determined by summing the current consumption of each component on their respective voltage rails. The total current on the 3.3 V rail is given by

$$I_{3.3V} = \sum I_{3.3V, \text{components}} \quad (4)$$

The power consumption of the 3.3 V rail is then calculated using Ohm’s Law [71]:

$$P_{3.3V} = 3.3 \times I_{3.3V} \quad (5)$$

Accounting for buck-boost regulator efficiency, the total battery current drawn from the single-cell Li-ion source is expressed as [71], [30]:

$$I_{\text{bat}} = \frac{P_{3.3\text{V}}}{\eta_{3.3\text{V}} \cdot V_{\text{bat}}} \quad (6)$$

Finally, the estimated system runtime is calculated using the standard battery capacity formula [71]:

$$t_{\text{runtime}} = \frac{C_{\text{bat}}}{I_{\text{bat}}} \quad (7)$$

where I is current (A), P is power (W), η is regulator efficiency, V_{bat} is the battery terminal voltage, and C_{bat} is the battery capacity (Ah).

TABLE XXXVI: Worst-Case Current for L.I.N.K.

Component	Rail	Worst-Case Current
ESP32-WROOM-32	3.3 V	130 mA
u-blox NEO-7M GNSS	3.3 V	67 mA
Bosch BNO055 IMU	3.3 V	12.3 mA
Bosch BMP280 Barometer	3.3 V	1.12 mA
SMT-0440-T-R Buzzer	3.3 V	90 mA
SSD1306 OLED HUD Module	3.3 V	25.05 mA
PTYS-12X LiDAR Module	3.3 V	450 mA
Total 3.3 V Current	3.3 V	775.47 mA

TABLE XXXVII: Power Consumption and Runtime Estimation for L.I.N.K.

Parameter	Value
$P_{3.3\text{V}}$	2.559 W
Total Power (Ideal)	2.559 W
Battery Current (Ideal)	691 mA
Total Power (90% Efficiency)	2.843 W
Battery Current (90% Efficiency)	768 mA
Runtime (3000 mAh, Ideal)	4.34 hours
Runtime (3000 mAh, 90% Efficiency)	3.91 hours

The power constraints of L.I.N.K. are determined based on the worst-case current consumption of each component by summing all components on the regulated voltage rail [71]. In Table XXXVI, the estimated 3.3 V rail current is about 775.47 mA under the theoretical maximum load.

In Table XXXVII, the estimated battery life, assuming the efficiency of the regulators is 90% [30], can range from 3.91 to 4.34 hours depending on the capacity of the battery used [29]. Therefore, this is considered a constraint due to the limited space in L.I.N.K. and the use of

a battery as its source of power. L.I.N.K. has a minimum battery life requirement of 1 hour, meaning that under maximum load, it fulfills this requirement.

The total power consumption can be primarily attributed to the LiDAR module, which means it has the greatest impact on runtime. Power management strategies such as duty cycling can help extend battery life, along with implementing a manual on/off switch so that unnecessary power draw does not occur.

4.2.2 Revised Power Constraints for the Implemented Design

The power constraints presented above were derived during the initial design phase, when the system was specified around a single 18650 cell and a single regulated 3.3 V rail. The fabricated prototype distributes power differently: three 18650 cells are connected in series to form an 11.1 V nominal pack, and two independent LM2596S step-down regulators produce separate 3.3 V and 5 V rails. The PTYS-12X LiDAR module is supplied from the 5 V rail, while the ESP32-S3, GNSS receiver, IMU, barometer, HUD, and buzzer remain on the 3.3 V rail.

Under theoretical maximum load, the 3.3 V rail draws approximately 325.47 mA, which corresponds to 1.074 W. The 5 V rail draws 450 mA for the LiDAR module alone, which corresponds to 2.250 W. The combined worst-case load presented to the battery is therefore 3.324 W, of which the LiDAR accounts for roughly two thirds. Assuming a regulator efficiency of 90, this corresponds to a pack current of about 333 mA, compared with 299 mA under ideal conditions.

Because the three cells are connected in series rather than in parallel, the pack capacity remains that of a single cell at 2500 mAh while the terminal voltage rises to 11.1 V. The estimated runtime under continuous maximum load is therefore 8.35 hours ideally and 7.51 hours at 90% efficiency [29]. Although the cells used in the implemented design hold less charge than those assumed in the original estimate, the series pack delivers the same power at roughly one third of the current because the terminal voltage is three times higher, which more than offsets the reduced capacity. The system continues to satisfy the 1 hour minimum battery life requirement by a wide margin, and the LiDAR module remains the dominant contributor to total power consumption, so duty cycling the rangefinder and switching the device off between uses remain the most effective ways to extend operating time.

4.2.3 Laser Safety Constraints

4.2.3.1 Laser Concerns for LiDAR

With the different laser classifications defined, most pre-made LiDAR modules are designed to use Class 1 or 1M lasers. Though this means there are fewer risks involved with the operation of these LiDAR modules, it is important to always practice caution regardless of classification. This is especially true given that ToF LiDAR module used for the L.I.N.K. device uses 905 nanometer wavelength light which is not visible to the human eye. As such, the risks are greater since user can not visually determine the orientation or power of the output beam. Furthermore, the long range nature of these ToF beams means that they have small beam diameters that hold their irradiance over long distances. Even though the detectable range for the ToF module is up to 1200 meters, that does not mean that the risk for laser exposure is the same. As such, we

provide a set of countermeasures to improve the overall safety from the LiDAR system of the L.I.N.K. device.

4.2.3.2 Manual Shutter

A countermeasure we provide for the LiDAR module is a manual shutter that blocks out going light from its laser. This safety measure is standard for many high power laser systems and is a simple way to avoid accidental exposure if the user accidentally turns on the LiDAR without knowing. The minor flaw of this countermeasure is that it is entirely up to the user to use the shutter, though it acts as more of a protective measure than a true safety feature of the device.

4.2.3.3 Laser On Light

Another important safety feature of the L.I.N.K. device is the use of an external LED that tells the user and people nearby that the LiDAR module is powered on and in use. This allows others nearby to practice caution around the user using the L.I.N.K. device and to avoid direct exposure to the line of sight of the user. This is especially important in cases where the user may not realize that laser is on when the user is not looking through the L.I.N.K. device and helps avoid accidental exposure.

4.2.3.4 HUD Caution Label

Along with the external caution provided by the LED when the laser is on, an internal message displayed on the HUD is implemented to tell the user that the laser is on and to practice caution in where the user is pointing the L.I.N.K. device. This warning will also be designated in a different color from the rest of the HUD display features to ensure that the user can differentiate it from the rest of the displayed information. The combination of the HUD label and LED ensures that the user can always determine whether or not the laser is on, even if the user intends to use the range finding feature of the L.I.N.K. device or not.

4.2.3.5 Laser Classification Label Sticker

Another safety feature of the L.I.N.K. device is a laser classification sticker that is commonly seen on Class 2 and above lasers that outline the risks of operations and measure of practicing caution when the laser is on. While this typically is not required for Class 1 lasers, attaching this label sticker ensures that any user without any prior knowledge of the risks of operating the range finding feature can read and recognize the risks of turning on the laser without the need for an external manual.

4.2.4 Optics Constraints

When designing the optical stacks for L.I.N.K. there were several constraints to consider in order to create a functioning system. The system must be light weight enough to be mobile, while delivering sufficient optical power to the user. This is challenging because off the shelf optics will undoubtedly leave aberrations on the view passed to the user, that are most often corrected with more lenses. Thus we must intelligently maximize our aberration control while trying to

also minimize the weight. There must also be significant light going into the system so that a clear, high contrast view is provided. We can look to the concept of Numerical Aperture, or NA to quantify this. In general, a higher NA will yield more light in the system, but its important to note- for a fixed focal length, the easiest way to increase NA is to use larger, and thus heavier optics.

Besides weight and lens diameter size, minimizing total system focal length is crucial for a mobile device. Here, total system focal length refers to the distance between the very first lens surface, and the point where the user places their eye. A major consideration here is the telescope type in each optical barrel. Some telescope configurations are longer than others, and so deciding on a configuration that optimizes all of our paraxial needs, aberration controls, and system length is crucial.

4.2.5 Practical Constraints

4.2.5.1 Time

Time is one of the most restrictive factors shaping this project. The senior design sequence only allows two semesters to move from initial concept to a fully tested system, which forces us to be deliberate about how we allocate our time. Rather than pursuing every possible improvement, we have to concentrate on completing core functionality within the available window, even if that means leaving certain refinements unaddressed.

This constraint becomes even more significant because the second half of the project takes place during a shortened summer term. With fewer weeks available than a typical semester, the margin for troubleshooting and system polishing is reduced. To account for this, we began working on prototyping and initial design work early to secure sponsorship for most of the optical components. This proactive approach was intended to ease the burden of securing expensive materials necessary for the project and provide more time to pursue smaller refinements.

Even with that head start, the limited timeline makes iteration particularly challenging. Engineering systems rarely work perfectly on the first attempt, and resolving issues in hardware, software, or integration can take longer than expected. Each unexpected problem consumes valuable time, and with little room for delays, even minor setbacks can cascade into larger scheduling risks.

External factors introduce additional complications. Delays in receiving key components, such as sensors or custom electronics, can stall progress at critical stages. Coordination also becomes more difficult when relying on input from suppliers, advisors, and teammates. During this period, many team members are also balancing travel for career-related commitments, which further reduces opportunities for consistent collaboration.

Taken together, these constraints create a real risk of ending the project under time pressure. To avoid that outcome, we prioritize early ordering of materials, maintain structured planning, and emphasize clear communication within the team so that progress remains steady despite the challenges.

4.2.5.2 Economic

The nature of how this senior design course is structured places certain financial limitations that play a major role in how our project is executed. The project did not come with guaranteed sponsorship, meaning every expense is covered personally. Fortunately, we were able to secure sponsorship from Edmund Optics to supply most of our optical components and therefore alleviate a large portion of the cost. Even with this sponsorship, the combined cost of all the other components including the printed circuit board and the MCU along with the sensors integrated on these boards are a significant portion of our budget. This requires us to be intentional with how we spend, making sure that each purchase directly supports progress without exceeding what we can realistically afford.

Other key components, like the LiDAR module and the ESP32, also represent a significant portion of our budget. These elements are central to the system's operation, handling both the range detection and processing tasks for calculating waypoints from the data provided by the sensor suite. Because their performance directly affect the reliability of the system, we prioritized obtaining dependable units from reliable vendors rather than cutting costs in areas that could compromise results.

In addition to planned purchases, we also have to account for uncertainties. Component prices and availability can change, and shipping delays may force us to pay for faster delivery options. There is also the possibility of hardware failure during testing, which would require replacements and further strain our budget. Given these risks, careful planning is essential. By researching alternatives, avoiding unnecessary purchases, and focusing spending on the most critical elements, we aim to stay within our financial limits while still delivering a fully functional system.

4.2.5.3 Environmental

One of the most prominent design constraints for our project is the environmental constraint. Since, L.I.N.K. is designed to measure long distances using LiDAR and use a binocular system to see long range targets, its functionality is directly influenced by the environment in which the device is used. Unlike systems that rely on controlled settings, our system requires an adaptive environmental baseline in which changes in environmental conditions can impact the rangefinding accuracy and reliability. As a result, environmental conditions must be carefully considered during the design, alignment, and implementation of the LiDAR system.

The first major environmental constraint is lighting conditions. Since our system depends on an avalanche photodiode to detect light at a very specific frequency, changes in ambient light such as sunlight intensity, cloud cover, and humidity can affect the readings for the LiDAR. To account for these conditions, we specifically acquired a LIDAR module with range capabilities well beyond the targeted system specification. The extended range capability accounts for situations where ambient lighting condition may decrease the performance of the LiDAR module to below its defined specification but still well within our desired range capability.

Another environmental constraint is the optical interference, including glare, bright flashes, and reflection. Glare from the sun or other bright lights sources that reflect directly towards the LiDAR module can interfere with the module's ability to detect a reflected laser pulse and

therefore produce false distance measurements. Similarly, actions like bright camera flashes can cause momentary spikes in intensity at the LiDAR module's operating wavelength and also produce false distance measurements. This is corrected by the built in software of the LiDAR module which filtering techniques that isolate and ignore these outliers that are detected.

A third environmental constraint is the surface reflectivity of a desired target. The detectable range of a target is highly dependent on how well the target is capable of reflecting the laser light back to the LiDAR module. For instance, pointing the LiDAR module at a black surface that is highly absorptive at the laser wavelength being used is likely to produce less accurate distance measurements due to less of the light returning back to the LiDAR module. Similarly, highly diffusive surfaces decrease the reflected power and also decrease the probability of a given pulse to return to the LiDAR module. While this issues is usually accounted for in the filtering software of the module, it is important that the user is notified of this such that they can choose to ping parts of a target that avoids these surface reflectivity issues.

Lastly, the propagation environment of the laser can also introduce problems. Factors like smoke and humidity can scatter or absorb light in ways that make reading inconsistent. Such effects are generally minor but can be magnified as further targets are being measured.

Addressing these concerns is best done through choice of LiDAR module. Ensuring that the LiDAR module has built in software filtering techniques to assist in distance calculations is a must since it is able to account for most if not all of the mentioned environmental factors.

5 Application of ChatGPT and Other Similar Platforms

After the release of OpenAI's Large Language Model (LLM), ChatGPT, in November of 2022, it has evolved rapidly and has produced many different types of LLMs. For the comparison of LLMs, the main models to be compared are OpenAI's ChatGPT, Google Gemini, and Anthropic Claude, each with its own language reasoning, meaning they have different strengths and weaknesses. For ChatGPT, version 5.3 is used; for Gemini, version 3; and for Claude, Sonnet 4.6. The use of LLMs helps streamline the process of conducting research and selecting parts, instantly showing part specifications and use cases for each component, with reference to L.I.N.K.

To ensure consistency across all LLMs referenced, the same prompt was given to each, including ChatGPT, Claude, and Gemini. For each prompt, each individual conducted a case study comparing results and verifying accuracy against accredited sources. They were analyzed based on the criteria of technical accuracy, completeness, and practicality to engineering principles. This comparison serves to show the limitations of LLMs and the differences between model reasoning.

5.1 Case Study 1

Referring to Appendix A, [1a], [1b], [1c], [1d], [1e], [1f], [1g], [1h] for Case Study 1, where the prompt and outcomes are given from ChatGPT, Gemini, and Claude.

ChatGPT [1b] addressed all parts of the prompt, providing a detailed overview of several types of regulators relevant to the project. ChatGPT outlined the keywords "efficiency, cost, footprint, complexity, and suitability" from the prompt, providing succinct, easy-to-understand descriptions while maintaining technical language. Each regulator is given a brief description that further simplifies surface-level understanding. The response ends with a comparative summary table that puts all of the options side by side with all of the attributed information. At the end of the response, it gives two possible options for the specific use case of L.I.N.K., considering multiple scenarios, and does not blindly assume key information. One fault LLMs tend to have is that they automatically assume information and try to apply it to a specific use case, therefore sometimes producing inaccurate results. ChatGPT gave a very general description and did not try to suggest specific regulator selection or provide much quantitative data, but was helpful in giving a general overview.

For example, the discussion of footprint and complexity is very vague and fails to quantify these parameters in the context of PCB constraints. These include PCB area, thermal limitations, and component availability.

In terms of technical accuracy, ChatGPT remains consistent with accurate information about the electronic components mentioned. This is observed when reading the description of linear regulators and the correct mention of their low efficiency, which is caused by poor heat dissipation, thereby confirming the accuracy of the information to some degree. The response does not attempt to go beyond the surface level, which provides a strong overview but makes it difficult to apply to the overall scale of the project.

Compared to Gemini and Claude, Gemini remains extremely surface-level and addresses very few specific requirements, but it does at least provide examples of each regulator type. One thing to consider when comparing with ChatGPT is that it had already been given access to much more information about the constraints of L.I.N.K., so it may be a less fair comparison.

Regarding Claude, there is a high level of technical accuracy, providing strong technical detail like ChatGPT. There are a few major inclusions that Claude handles better, including a switched-capacitor charge pump, which reflects greater research depth. It gives a highly informative graph of the efficiency vs. load current, and an arbitrary score report of each regulator with regard to the categories mentioned in the prompt. It provides a use-case suitability matrix that is very readable and easy to interpret, along with strict design constraints that make it extremely easy to understand which regulators are most applicable to each use case.

While the information across all models is technically accurate, Claude was the most appropriate model for delivering strong technical depth from a single prompt. All responses were suitable in some regards, but cannot be used as the only source of research. They need to be compared against datasheets and supplemented with other tools, such as Webench, when selecting specific components.

5.2 Case Study 2

Referring to Appendix A, [2a], [2b], [2c], [2d], [2e], [2f], [2g] for Case Study 2, where the prompt and outcomes are given from ChatGPT, Gemini, and Copilot.

For this case study, the specific model versions used were ChatGPT 5.4 Thinking, Google Gemini base model, and Copilot default. This case study shows how each LLM handles a question regarding the different types of long ranging LiDAR given the prompt: Provide an in depth comparison between different LiDAR types for long distance ranging. Ensure that range, complexity, cost, and any other relevant factors are accounted for.

Relative to the prompt in appendix [2a], the three responses in appendices [2b], [2d], and [2f] all recognize the central long-range LiDAR trade space, but they do so with different definitions of what counts as a “type,” different levels of technical completeness, and different degrees of alignment with the requested comparison criteria of range, complexity, and cost. The ChatGPT response in [2b] interprets LiDAR types primarily by ranging principle, separating the discussion into Time-of-Flight, phase-shift/AMCW, and FMCW. The Gemini response in [2d] begins from the same principle-level distinction between ToF and FMCW, but then broadens the meaning of “type” to include implementation choices such as wavelength, detector technology, and beam steering architecture. The Copilot response in [2f] is the broadest and least sharply defined, mixing ranging principles like ToF and FMCW with use-case or platform categories such as terrestrial long-range scanners. As a result, [2b] is the most internally disciplined in taxonomy, [2d] is the most expansive in system-level context, and [2f] is the least rigorous in separating physical ranging methods from deployment formats.

In terms of scope and completeness, appendix [2b] is the strongest direct answer to the prompt because it covers three distinct ranging families and evaluates each one against the requested criteria in a consistent way. Its treatment of ToF is practical and balanced, its discussion of AMCW explains why ambiguity limits reduce its suitability for kilometer-scale ranging, and its treatment of FMCW captures both the performance upside and the engineering penalty. That inclusion of AMCW in [2b] is especially important because it makes the comparison feel genuinely comparative rather than simply binary. By contrast, appendix [2d] largely narrows the core comparison to ToF versus FMCW and omits AMCW entirely, so although it is richer in practical detail, it is less complete as a survey of LiDAR ranging modes. Appendix [2f] is the most limited in scope because it effectively provides a high-level summary rather than an in-depth comparison; it mentions the major tradeoffs correctly, but it does not unpack them deeply enough to satisfy the “in depth” part of [2a] as well as [2b] or [2d] do.

With respect to treatment of range, appendix [2b] does a good job of linking range to first-order physical constraints. It explains that ToF can reach hundreds of meters to beyond a kilometer with adequate aperture and pulse energy, while also noting eye-safety and attenuation limits. It also clearly explains why AMCW is generally constrained to short-to-medium distances unless extra disambiguation schemes are used, and why FMCW can offer strong long-range sensitivity due to coherent detection. Appendix [2d] is arguably even more concrete on range-limiting mechanisms because it introduces the 905 nm versus 1550 nm wavelength tradeoff, tying long-range performance to eye safety and detector technology in a very practical way. It also connects beam steering choices to long-range performance, noting how flash architectures spread energy

over the full scene and therefore struggle at long distance, while mechanical scanners preserve beam quality better. Appendix [2f] addresses range more generically, saying that high-energy pulses can reach hundreds of meters and that terrestrial scanners can exceed a kilometer, but it leaves the underlying reasons less developed. So on pure explanatory richness around range, [2d] is strongest, [2b] is close behind and more methodically organized, and [2f] is correct but thin.

On system complexity, appendix [2b] again stands out for clarity. It explains ToF as comparatively simple because it relies on mature pulsed sources, fast photodetectors, and timing electronics, then contrasts that with FMCW's need for narrow-linewidth lasers, coherent mixing, chirp linearity, and heavy signal processing. It also makes a nuanced point that AMCW removes some timing burden but reintroduces complexity through multi-frequency disambiguation and phase-noise sensitivity. Appendix [2d] goes further into implementation complexity by describing coherent detection, FFT-heavy processing, and ultra-stable lasers for FMCW, while also adding a second axis of complexity through scanning mechanisms such as mechanical assemblies, MEMS mirrors, flash illumination, and optical phased arrays. That makes [2d] feel more like a system engineering assessment than a pure ranging-principle comparison. Appendix [2f] captures the broad idea that FMCW is more complex than ToF and that terrestrial scanners often require mechanical scanning and calibration, but it does not explain those complexities with the same specificity. In other words, [2b] is the clearest comparative explanation, [2d] is the richest systems discussion, and [2f] is the most compressed.

Cost is another category where the responses diverge significantly. Appendix [2b] discusses cost in relative terms, presenting ToF as the most cost-effective mature option and FMCW as historically expensive due to complexity, while describing AMCW as moderate in cost but limited in long-range usefulness. This is adequate and logically consistent, but still somewhat generalized. Appendix [2d] offers the strongest cost analysis because it identifies concrete cost drivers: 905 nm ToF benefits from cheaper silicon detectors, while 1550 nm designs require more expensive InGaAs sensors; mechanical units remain costly because of moving parts and packaging; FMCW remains expensive because of laser stability and processing requirements. This makes [2d] the most grounded response in terms of why the cost differences exist rather than merely stating that they do. Appendix [2f] mentions that cost scales with power, detector sensitivity, optics, and calibration, which is directionally correct, but it lacks the granularity seen in [2d]. Therefore, if the evaluation criterion is how well the response explains cost formation rather than merely mentioning cost, [2d] is strongest, followed by [2b], then [2f].

A major distinction between appendices [2b] and [2d] is that [2d] expands beyond ranging physics into the architecture of full LiDAR systems. Its discussion of mechanical scanning, MEMS, flash, and OPA gives a more realistic picture of how long-range LiDAR products are actually built and why certain solutions dominate in automotive or mapping markets. That is a real strength, because a practical engineer often cares not only about ToF versus FMCW but also about whether the beam is mechanically scanned, whether the design is solid-state, and what packaging penalties result. However, this broader scope also causes [2d] to partially drift from the prompt in [2a], because the prompt asks for a comparison between LiDAR "types" in the context of long-distance ranging, and [2d] treats "type" as both sensing principle and steering mechanism. Appendix [2b] stays more tightly locked to the prompt by keeping the comparison anchored around measurement method. Appendix [2f] similarly broadens the definition of type,

but unlike [2d], it does so without enough depth to justify the broadened frame, so the result feels less analytical and more descriptive.

In terms of technical balance and objectivity, appendix [2b] is the most even-handed. It does not overstate the superiority of FMCW, instead presenting it as theoretically powerful but still burdened by practical manufacturing and cost barriers. It also does not dismiss ToF as old or inferior; instead, it frames ToF as the present-day practical winner because of maturity, cost, and adequate range. Appendix [2d] is also strong, but its language is more rhetorical and market-oriented. Phrases like “rising star,” “gold standard,” and “truth and economy” make the analysis more vivid, but also slightly less neutral and slightly more vendor-market flavored than [2b]. Appendix [2f] is neutral in tone, but because it is shorter and less detailed, it sometimes reads like a synthesis paragraph from a general guide rather than a deep comparative analysis. Thus, [2b] is the most academically balanced, [2d] is the most industry-inflected, and [2f] is the most generic.

Another important point is how each response handles the “other relevant factors” requested in [2a]. Appendix [2b] includes ambient light susceptibility, multipath reflections, lack of direct velocity measurement in ToF, and direct radial velocity sensing in FMCW. Appendix [2d] expands this category further by adding wavelength-dependent eye safety, detector material constraints, interference immunity, beam quality across field of view, vibration sensitivity in MEMS, and sidelobes in OPA. This makes [2d] the richest response in peripheral engineering factors, especially those that matter in real deployments. Appendix [2f] mentions atmospheric conditions, beam divergence, scanning geometry, field of view, and acquisition speed, which are relevant, but it does not elaborate much on them. So for secondary but important design variables, [2d] is the most comprehensive, [2b] is selective but strong, and [2f] is again more of a concise overview.

As writing, the three responses also differ in structure and usability. Appendix [2b] is organized in a clear sequence: establish the major principles, evaluate each one, then synthesize the overall trade space in a concluding paragraph. That makes it easy to follow and especially useful for a reader who wants a principled comparison. Appendix [2d] is more layered and reads almost like a mini technical article. It starts with ToF versus FMCW, then folds in wavelength choices, then beam steering methods, then future market direction. This makes it highly informative, but slightly less focused and somewhat more sprawling. Appendix [2f] is the most compact and perhaps the easiest to skim, but that brevity comes at the expense of comparative depth. For a reader seeking a quick summary, [2f] is serviceable. For a reader seeking a strong engineering comparison, [2b] and [2d] are clearly better.

5.3 Case Study 3

Referring to Appendix A, [3a], [3b], [3c], [3d], [3e], [3f], [3g] for Case Study 3, where the prompt and outcomes are given from ChatGPT, Gemini, and Copilot.

For this case study, the specific model versions used were ChatGPT 5.4 Thinking, Google Gemini base model, and Copilot default. This case study shows how each LLM handles a question

regarding how to specifically choose and consider a prism for binocular use: Give me a list and in depth comparison of different prism configuration for optical binoculars. Take into account cost, weight, and any optical parameters you think are necessary when picking a prism type.

ChatGPT 3[b] provides the most technically complete and engineering-oriented treatment of binocular prism configurations. It frames the discussion in terms of system-level optical design, explicitly connecting prism choice to parameters such as transmission efficiency, phase error, alignment tolerance, and optical throughput. It also expands beyond the standard Porro vs. roof dichotomy by including Reverse Porro, Schmidt–Pechan, Abbe–Koenig, and less common systems like Amici and Uppendahl prisms. The inclusion of an engineering comparison table and explicit design recommendations reflects a perspective aligned with optical system design workflows rather than end-user selection. In contrast, 3[d] and 3[f] are narrower in scope and omit several of these less common configurations, indicating a more constrained treatment of the design space.

The Gemini response in 3[d] shifts the framing significantly toward usability and purchasing decisions. While it still introduces key prism types (Porro, Schmidt–Pechan, and Abbe–Koenig), the explanations are simplified and structured around intuitive concepts such as brightness, compactness, and “3D depth.” It also introduces the Perger-Porro hybrid, which is not covered in 3[b] or 3[f], but does so in a descriptive rather than analytical manner. Compared to 3[b], it lacks discussion of deeper optical phenomena such as phase errors, coating dependencies, and alignment tolerances, and instead emphasizes practical trade-offs like portability and cost for consumers. This makes 3[d] more accessible but less rigorous from an optical engineering standpoint.

The Copilot response in 3[f] occupies a middle ground between the other two. It focuses on the three dominant prism architectures—Porro, Schmidt–Pechan, and Abbe–Koenig—and provides a structured breakdown of optical characteristics, mechanical traits, and cost. Unlike 3[d], it incorporates more explicit technical reasoning, including references to total internal reflection, phase correction, and transmission efficiency. However, compared to 3[b], it does not explore secondary or legacy prism configurations and maintains a tighter focus on commonly used designs. Its comparison table is more quantitative in presentation than 3[d], but less expansive in scope than the engineering table in 3[b].

Overall, the three responses differ primarily in depth, scope, and intended audience. The response in 3[b] is the most comprehensive and aligned with optical engineering analysis, suitable for someone designing or modeling binocular systems. The response in 3[d] is the most user-oriented, prioritizing clarity and purchasing guidance over technical completeness. The response in 3[f] balances these approaches by focusing on the most relevant prism types while still maintaining a structured and moderately technical comparison. Together, they illustrate how different AI systems prioritize either completeness, accessibility, or conciseness when addressing the same technical prompt.

5.4 Case Study 4

Referring to Appendix A, [4a], [4b], [4c], [4d], [4e], [4f], [4g] for Case Study 4, where the prompt and outcomes are given from ChatGPT, Gemini, and Claude.

For this case study, the specific model versions used were ChatGPT 5.4 Thinking, Google Gemini 3 Pro, and Anthropic Claude Opus 4.6 High. This case study shows how each LLM handles a firmware architecture question directly relevant to L.I.N.K.: the choice between a cooperative, timer-based main loop and a FreeRTOS multi-task architecture for polling four I²C sensors at 20 Hz on the ESP32-S3. The L.I.N.K. firmware uses the cooperative approach, so this prompt tests whether each model can accurately articulate the trade-offs behind that decision and provide a practical recommendation.

ChatGPT [4b] opened with the most important observation: at 20 Hz, the ESP32-S3 is nowhere near its timer limits, so the decision is “less about raw timing capability and more about isolation, sleep behavior, and how you want failures to be contained.” This demonstrates a strong engineering intuition. ChatGPT correctly identified that I²C is serial and half-duplex, meaning FreeRTOS tasks do not create real parallel sensor I/O on a shared bus; they mostly add scheduler jitter and locking overhead. It noted that the ESP32-S3 has two I²C controllers, so tasks only buy real bus-level parallelism if devices are actually split across both controllers. This is a critical insight that directly validates L.I.N.K.’s design, in which all four sensors and the OLED display share a single I²C bus over GPIO 8 and 9.

On power consumption, ChatGPT made an accurate point: “the power loss comes from busy polling, not from the label ‘main loop’ versus ‘RTOS.’” It explained that both approaches can be efficient if they block properly between samples, and that ESP-IDF’s tickless idle works with either architecture. This is more precise than a blanket statement that cooperative loops waste power, which is a common misconception. ChatGPT also mentioned that ESP-IDF power management adds up to 40 μ s of interrupt latency, which it correctly noted is negligible relative to a 50 ms sample period.

For watchdog management, ChatGPT explained that a cooperative poller has straightforward WDT handling each cycle, either completes, times out, or triggers a recovery path in one place, whereas a multi-task design requires every polling task to block or yield correctly, and every path that can stall must be bounded. Its practical recommendation was to use a single periodic sensor-poll task with cooperative scheduling, triggered by `vTaskDelayUntil()` or an `esp_timer` callback, and to add per-sensor tasks only when there is a real need for fault isolation or parallelism across a second I²C controller. This is a measured, well-justified recommendation that aligns with L.I.N.K.’s current architecture.

Gemini [4d] took a different, more opinionated stance, declaring FreeRTOS tasks “the objectively better choice” for the ESP32-S3. It argued that writing a cooperative loop on the ESP32 is “essentially fighting the framework and leaving the dual-core and power-saving capabilities on the table.” While Gemini is correct that the ESP32-S3 natively runs on FreeRTOS via ESP-IDF, calling the cooperative approach objectively wrong is an oversimplification. The Arduino framework’s `loop()` function already runs inside a FreeRTOS task, so a cooperative poller is not competing with the RTOS it runs within. This is an important distinction that Gemini failed

to make.

Gemini’s assessment of power consumption was the least accurate of the three models. It claimed that cooperative loops are “generally worse for power” due to busy-waiting, using `while(millis() - lastTime < 50)` as an example. While this is true of naïve implementations, the characterization does not hold for a well-structured cooperative loop that yields to the idle task. ChatGPT and Claude both identified this nuance more clearly. On a positive note, Gemini did correctly describe the I²C bus contention problem and suggested using `xSemaphoreCreateMutex()` to lock the bus during transactions. It also provided a useful summary comparison table that organized the trade-offs across jitter, bus contention, power efficiency, multi-core usage, and WDT risk, which makes the information easy to digest at a glance.

Gemini’s strongest contribution was its discussion of the Task Watchdog Timer in the context of I²C bus hangs: it explained that if a sensor pulls the SDA line low and freezes a blocking call, the cooperative loop gets stuck and the TWDT is not fed, potentially causing a system reset. It correctly noted that RTOS-aware I²C drivers can set explicit hardware timeouts and return `ESP_ERR_TIMEOUT`, allowing graceful bus recovery. This is a valid point, though it applies primarily to the ESP-IDF I²C driver L.I.N.K. uses Arduino-framework Wire library calls, which have their own timeout behavior. The broader claim that FreeRTOS is “objectively better” was not well supported, given the specific constraints of the L.I.N.K. hardware (a single shared bus, ample timing margin, and a battery-powered prototype where simplicity accelerates development).

Claude [4f] delivered the most balanced and structured response of the three models. It organized its analysis across four clearly labeled axes: real-time constraints, power consumption, code complexity, and watchdog timer management, and provided a verdict for each axis rather than deferring to a single blanket recommendation.

On real-time constraints, Claude correctly identified that cooperative loops give fully predictable execution order and minimal jitter as long as each iteration completes within the 50 ms budget, but noted the critical weakness: “if *any* sensor read blocks (e.g., clock stretching on I2C), **all** other sensors are delayed.” For FreeRTOS, it acknowledged better isolation between sensor tasks but correctly pointed out that I²C bus contention still matters because the bus is shared. This is the same serialization insight that ChatGPT made, and it directly applies to L.I.N.K.’s hardware configuration.

On power consumption, Claude was the only model to argue that the cooperative loop is *more* power-efficient by default, not less. It explained that a cooperative loop can explicitly enter light sleep until the next 50 ms window, incurring no context-switching overhead and no scheduler ticks consuming power, whereas FreeRTOS requires enabling `configUSE_TICKLESS_IDLE` to prevent the 1 kHz tick interrupt from blocking deep sleep states. This is a more accurate characterization than Gemini’s claim that cooperative loops are “generally worse for power,” and it aligns with ChatGPT’s observation that power loss comes from busy polling rather than the scheduling model itself.

Claude’s code complexity analysis was the most practically useful of the three. It provided

concrete ESP-IDF code examples for both approaches: a cooperative loop using `esp_timer_get_time()` with sequential sensor reads, and a FreeRTOS design using `xSemaphoreCreateMutex()` for I²C bus arbitration with `xTaskCreatePinnedToCore()` for per-sensor tasks. The code examples make it easy to see the difference in complexity: the cooperative version is roughly 10 lines of straightforward sequential logic, while the FreeRTOS version requires mutex creation, per-task stack allocation (approximately 4 KB each), semaphore take/give pairs, and pinned core assignments. Claude explicitly noted that undersized stacks cause silent corruption, a debugging hazard that is especially painful on embedded targets without JTAG.

For watchdog management, Claude identified a practical nuance that the other models missed: in the default ESP-IDF configuration, the TWDT monitors the idle task rather than application tasks. This means that in a FreeRTOS design, a hung sensor task may go unnoticed because the idle task continues to run and feed the watchdog. Explicitly adding each sensor task to the TWDT via `esp_task_wdt_add(NULL)` introduces per-task boilerplate that the cooperative approach avoids entirely.

Claude's final recommendation was the most nuanced of the three: start with a cooperative loop for sensor-polling-focused, power-sensitive systems, move to FreeRTOS tasks when concurrent networking or variable sensor latencies demand it, and consider a hybrid approach a single FreeRTOS task that owns all I²C polling cooperatively (sequential reads, no mutex needed) with separate tasks for networking and processing. This hybrid suggestion is directly applicable to L.I.N.K., where the main loop already runs cooperatively within a FreeRTOS task while the NimBLE stack operates in its own task. Claude also provided a summary comparison table covering real-time performance, power, complexity, watchdog management, I²C bus sharing, failure isolation, and RAM usage, making the trade-offs easy to reference at a glance.

Across all three models, the technical information about FreeRTOS primitives, I²C bus behavior, and ESP32-S3 hardware capabilities was largely accurate. The key differentiator was in engineering judgment and how each model framed its recommendation. ChatGPT and Claude both arrived at the cooperative approach as the correct default for this specific hardware configuration a single shared I²C bus with ample timing margin while providing accurate reasoning about bus serialization and power management. Gemini reached the opposite conclusion, declaring FreeRTOS "objectively better," which is defensible as a general ESP32-S3 guideline but does not account for the specific constraints of L.I.N.K. Gemini's power consumption analysis contained the most significant inaccuracy: it characterized all cooperative loops as busy-waiting, a claim both ChatGPT and Claude correctly refuted.

Claude was the most useful model for this particular question because it combined accurate technical depth with practical code examples, a balanced recommendation that acknowledged when each approach is appropriate, and a hybrid suggestion that maps directly onto L.I.N.K.'s existing architecture. ChatGPT was a close second, providing a nuanced analysis that correctly framed the decision in terms of isolation and failure containment rather than raw performance. Gemini, while technically competent in its discussion of RTOS primitives and watchdog behavior, lost credibility by presenting a one-sided verdict that did not account for the specific hardware constraints described in the prompt. All specific API references (`vTaskDelayUntil()`, `esp_task_wdt_add()`, `xSemaphoreCreateMutex()`, `ESP_ERR_TIMEOUT`,

configUSE_TICKLESS_IDLE) and hardware claims were verified against the ESP-IDF v5.x documentation and the ESP32-S3 Technical Reference Manual to confirm accuracy before inclusion in this report.

6 System Optical Design

6.1 Paraxial Design

When designing any optical system, starting with a first pass paraxial design is important to get key characteristics established. A system can not be optimized unless a baseline is known.

6.1.1 Desired Optical Characteristics

We have decided to design a system to have greater than 5x magnification, 4.5° true field of view, and 22.5° apparent field of view.

6.1.2 Zemax Design, and Baseline Aberrations

We have designed a paraxial version of our system, with an objective lens of 40 mm diameter by +160 mm focal length. The eye piece lens is 25 mm diameter by +40 mm focal length.

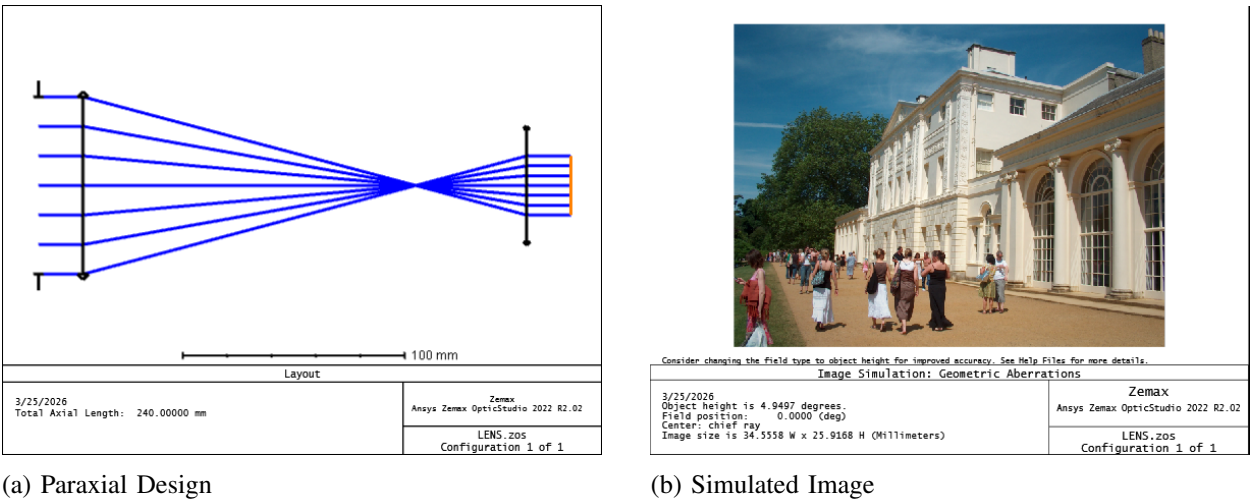


Fig. 7: Paraxial Design and Simulated Image.

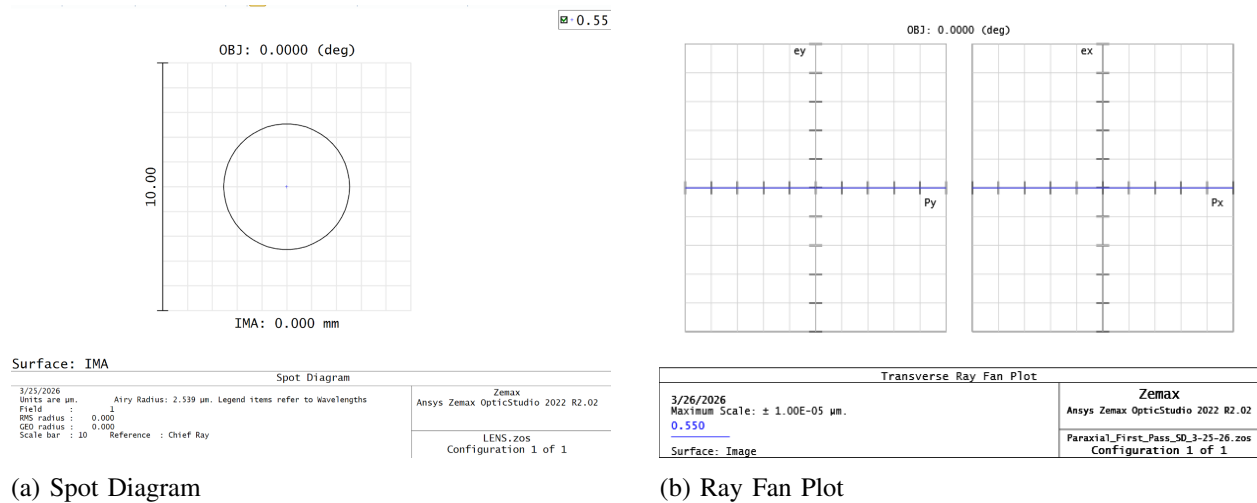


Fig. 8: Paraxial Spot Diagram and Ray Fan Plot.

6.2 Implementation of Off the Shelf Components

6.2.1 Full Component List

First, let's discuss which elements have been picked for the real, non-paraxial system, along with their justifications, and specifics.

TABLE XXXVIII: Optical Component Summary for L.I.N.K. Optical Stack

Optical Element	Focal Length	Diameter	Reason for Inclusion
Achromatic Objective Lens	+160 mm	40 mm	Collects incoming scene light and forms the initial real image used for magnification. Provides the aperture needed for image brightness.
Beam Splitter (70:30)	N/A	20 mm	Combines the OLED display projection with the external scene by transmitting visible light while reflecting the HUD image into the optical path.
Focus Extender	-100 mm	20 mm	Pushes intermediate focus outside of Porro prism so that a field lens may be used.
Porro Prism Pair	N/A	25 mm	Erects the inverted image and folds the optical path to maintain a compact binocular form factor while improving depth perception.
Field Lens	+60 mm	25 mm	Maintains field uniformity and directs rays toward the eyepiece pupil, improving illumination and field stability.
Eyepiece Lens	+50 mm	25 mm	Magnifies the intermediate image to produce the final apparent field of view seen by the user.

6.2.2 Optical Barrel #1: No Beam Splitter

Since the HUD system only exists in one of the Optics Barrels, one will not have the beam splitter. The first optics barrel is only the above white boxes, while the second optics barrel includes the above shaded blue box.

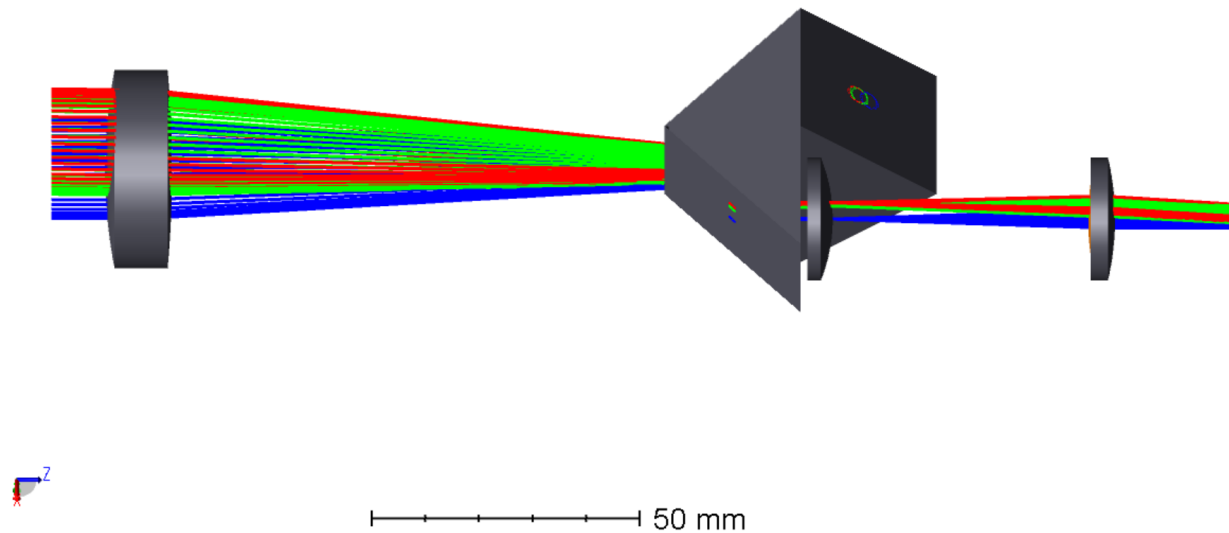


Fig. 9: Zemax Layout

6.2.3 Optical Stack #1: Optimization Results

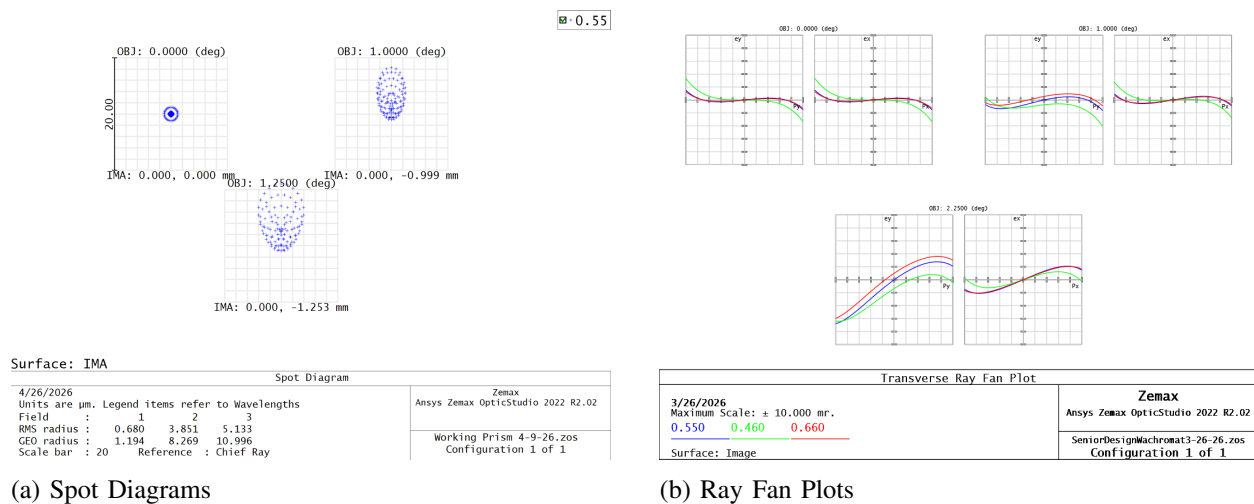


Fig. 10: No Beam Splitter: Spot Diagrams and Ray Fan Plots at Different Field Angles.



Consider changing the field type to object height for improved accuracy. See Help Files for more details.

Image Simulation: Geometric Aberrations	
3/26/2026 Object height is 2.2500 degrees. Field position: 0.0000 (deg) Center: chief ray - relative illumination cannot be computed Image size is 0.2989 W x 0.2242 H (direction cosines)	Zemax Ansys Zemax OpticStudio 2022 R2.02 SeniorDesignWachromat3-26-26.zos Configuration 1 of 1

Fig. 11: Simulated Image

Now, with the added elements, we have access to higher field angles, as well as access to better aberration control. See Zemax lens view, and analysis outputs. After designing and reviewing the necessary elements we have arrived at a solution which meets our design goals, and is optimized for aberration reduction.

6.2.4 Optical Barrel #2: Beam Splitter

For Optical Barrel #2, the beam splitter and curved mirror are necessary to introduce the micro-OLED display. This system comprises the Integrated HUD system. The Zemax design excluding the mirror is below:

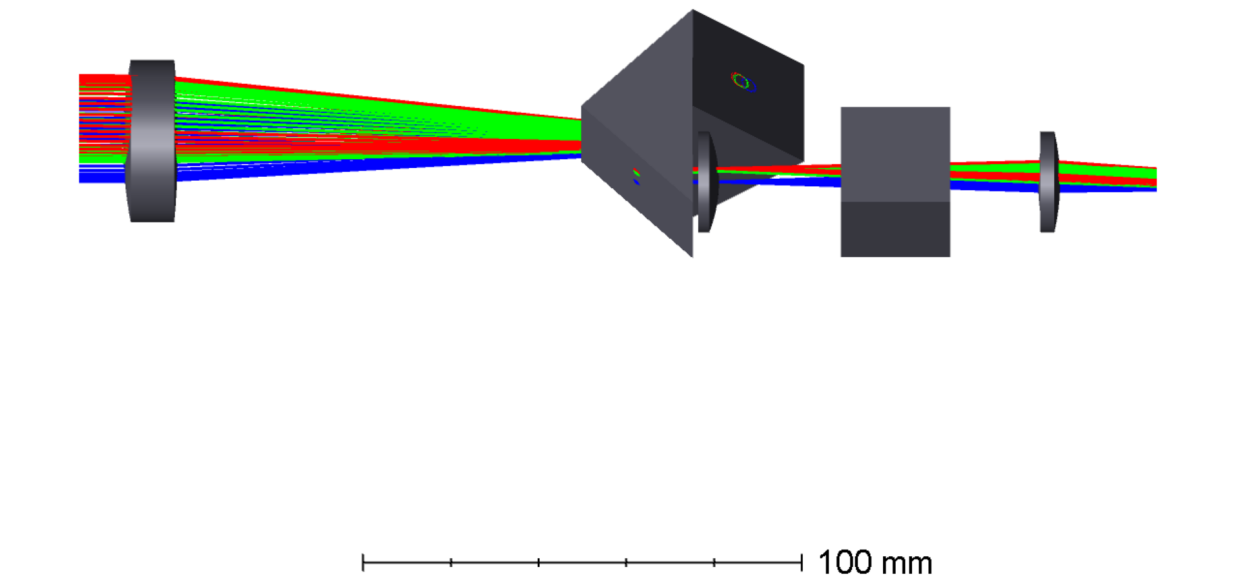
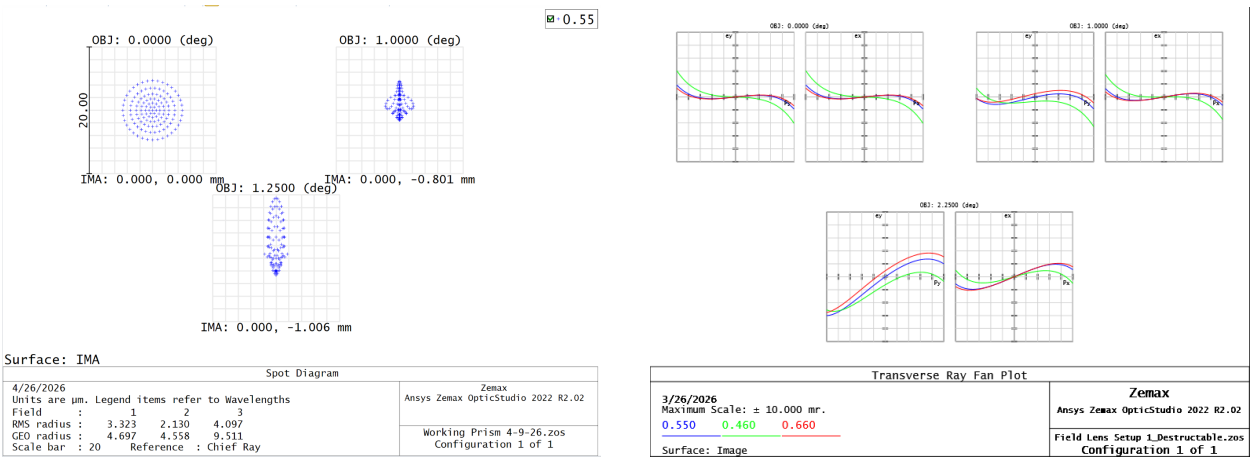


Fig. 12: Zemax Layout

6.2.5 Optical Barrel #2: Optimization Results

Optical Barrels #1 and #2 have near identical aberration, power and F.O.V. profiles. Seen below are the aberrations for the Optical Barrel #2: With Beam Splitter.



(a) Spot Diagrams

(b) Ray Fan Plots

Fig. 13: Spot Diagrams and Ray Fan Plots at Different Field Angles.

The culmination of the aberrations and distortions have been used to simulate an image of the final system below:

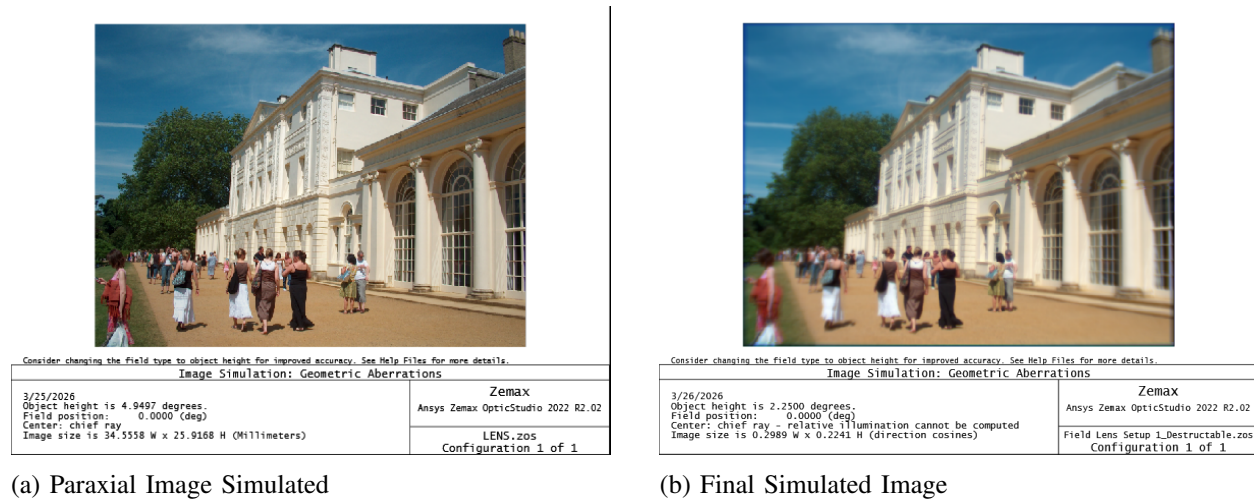


Fig. 14: Paraxial vs Final Simulated Image.

6.3 Integrated H.U.D. System

The design of the Integrated H.U.D. system was fairly simple, such that a single 1 F lens system is how we obtain magnification of the OLED display. The image from the micro-OLED display is first displayed onto the beam splitter which then combines the image into the optical path of the Optical Stack. Now the user is able to see the HUD information without the need to adjust their eyes focus relative to the target. Below is the current Zemax design with annotations:

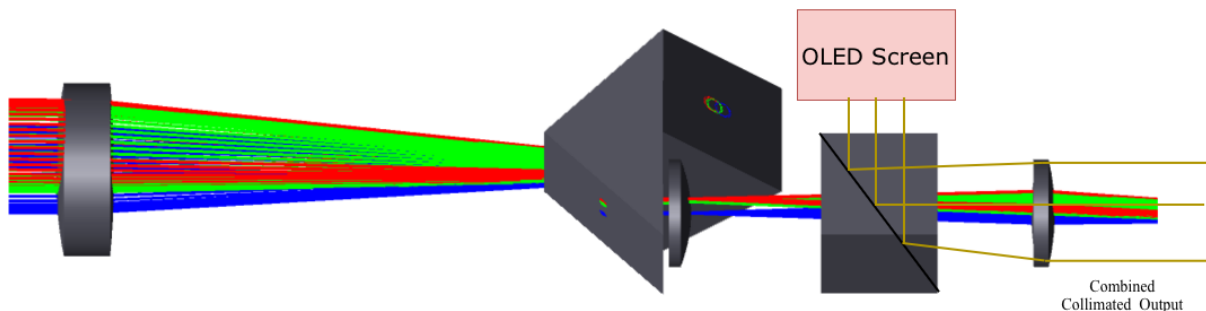


Fig. 15: H.U.D. Zemax Layout

6.4 Conclusions

We have designed a two barreled binocular system which can perform the necessary optical specifications. One barrel allows for the integration of the head-up display, while maintaining the same aberration, power, and F.O.V. profiles. The

7 Hardware Design

7.1 Hardware General Architecture

In Fig. 3, the diagram illustrates the sub-level architecture of the electronic components and how they interact with one another. The PSU, consisting of two regulators, is labeled as PCB #1, where a regulated 3.3 V rail supplies the sensor suite, ESP32, and HUD, and a regulated 5 V rail supplies the LiDAR. The communication protocols between the sensor and peripheral suite are labeled, including I²C, UART, and PWM.

7.2 Electrical Power System

7.2.1 Voltage Regulator

Fig. 16 shows the schematic for the 3.3 V rail, which is implemented with an LM2596S-3.3 step-down regulator. The 5 V rail, shown in Fig. 17, uses the same topology with an LM2596S-5.0 in place of the 3.3 V part. Because both are fixed-output versions of the device, the output voltage is set by an internal divider referenced to the 1.23 V band-gap reference, and no external feedback network is required. As seen in the schematic, the FEEDBACK pin is instead tied directly to the output node at the load side of the inductor, which is the connection specified by the datasheet for the fixed-output parts. The general relationship for the adjustable version of the device is

$$V_{\text{out}} = V_{\text{ref}} \left(1 + \frac{R_2}{R_1} \right), \quad V_{\text{ref}} = 1.23 \text{ V} \quad (8)$$

which is realized internally at 3.3 V and 5 V in the parts selected here. Unlike a buck-boost converter, the LM2596 is a step-down topology only, so the input rail must remain above the regulated output by at least the dropout margin of approximately 1.5 V across the full operating range. The ON_OFF_N pin is an active-low enable and is tied to ground in this design so that the regulator is enabled whenever input power is present. No external timing components are needed, as the switching frequency is fixed internally at 150 kHz.

Another design choice involved selecting the passive components surrounding each regulator. The datasheet sizes the inductor using the volt-microsecond constant

$$E \cdot T = (V_{\text{in}} - V_{\text{out}}) \frac{V_{\text{out}}}{V_{\text{in}}} \cdot \frac{1000}{f_{\text{sw}} (\text{kHz})} \quad \text{V} \cdot \mu\text{s} \quad (9)$$

which, evaluated at the nominal input voltage and combined with the maximum load current, identifies the required inductance from the datasheet nomograph. This procedure yields a 33 μH inductor for the L.I.N.K. operating conditions, implemented with a Bourns SRR1260-330M. This part is a shielded power inductor with low DCR, which minimizes electromagnetic interference

and conduction loss, and its saturation current rating exceeds the peak switch current of the converter.

The remaining passives follow the datasheet reference design. A $68\ \mu\text{F}$ low-ESR electrolytic capacitor is placed at VIN to supply the pulsed input current drawn by the internal switch and to limit input ripple, and a $220\ \mu\text{F}$ low-ESR capacitor at the output filters the inductor ripple current and sets the converter's transient response. Diode CR1 serves as the catch diode, providing a current path for the inductor during the switch off-time; a Schottky type is used to minimize forward-voltage loss, and it is rated above both the peak switch current and the maximum input voltage.

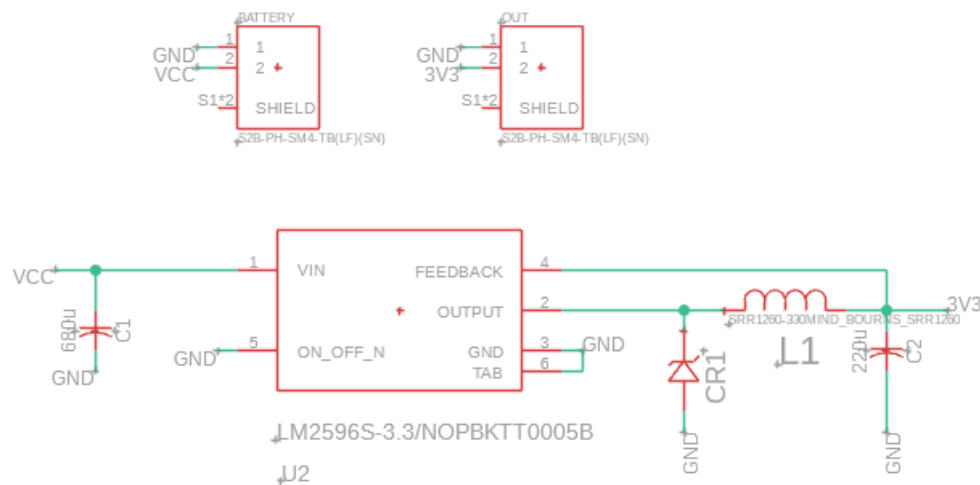


Fig. 16: LM2596S 3V3 Buck Converter Schematic

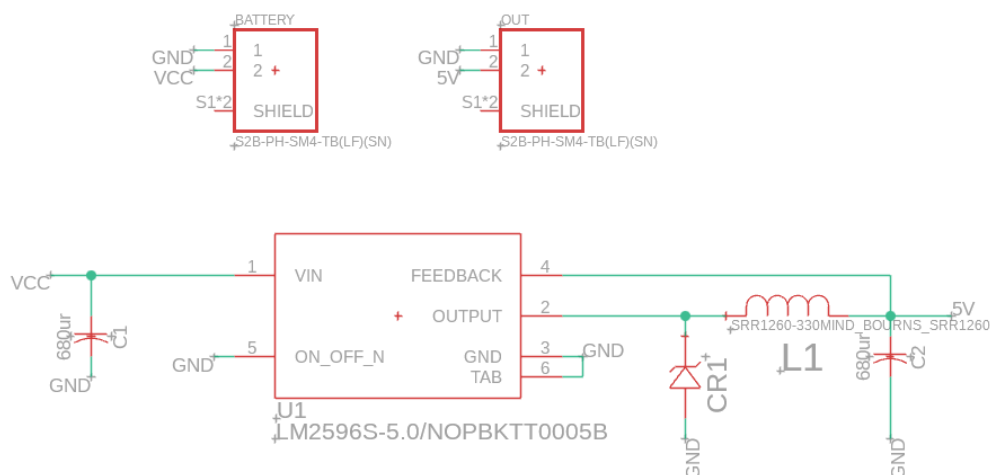


Fig. 17: LM2596S 5V Buck Converter Schematic

7.2.2 Power Distribution to Regulators

In Fig. 18, this is the general power configuration that supplies the regulators. The nominal 11.1 V from the 3 batteries in series is fed into a switch that then flows into a female header. This header connects to the regulators via a male header, where the voltage is regulated to produce the 3.3 V and 5 V rails. The regulated voltages are then distributed to the subsystems of L.I.N.K.

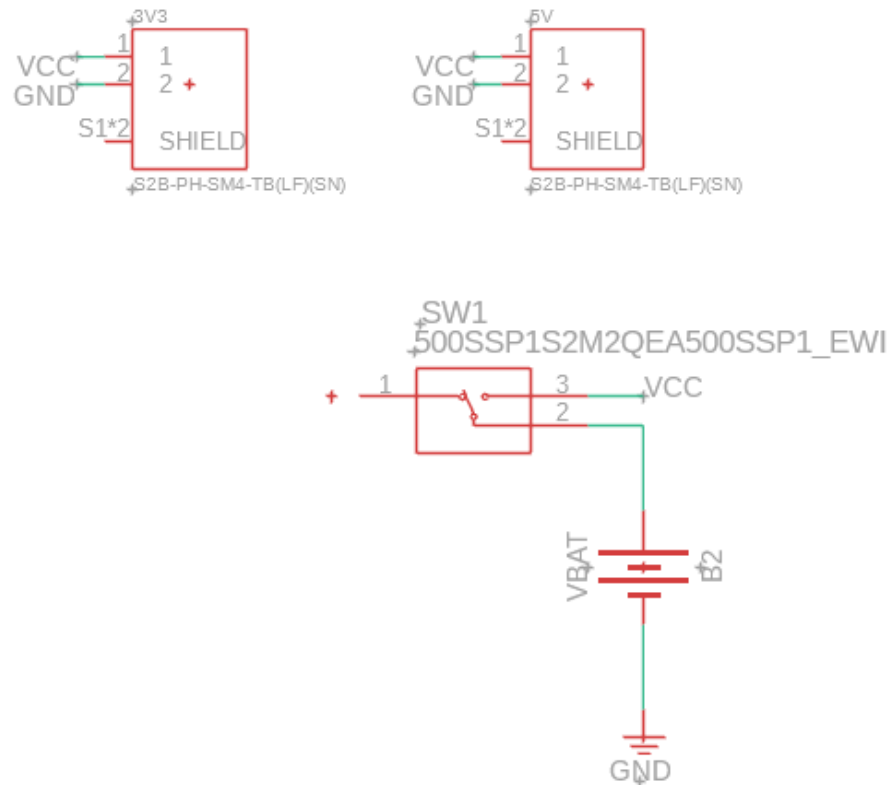


Fig. 18: Power Architecture Schematic Showing Battery Connection

7.3 Sensors and Peripherals

7.3.1 USB to Serial Converter

In Fig. 19, a USB-A to serial converter is shown. USB-A allows for fast and efficient transfer of data, that can be used to program the ESP32, it connects via a JST to the MCU schematic.

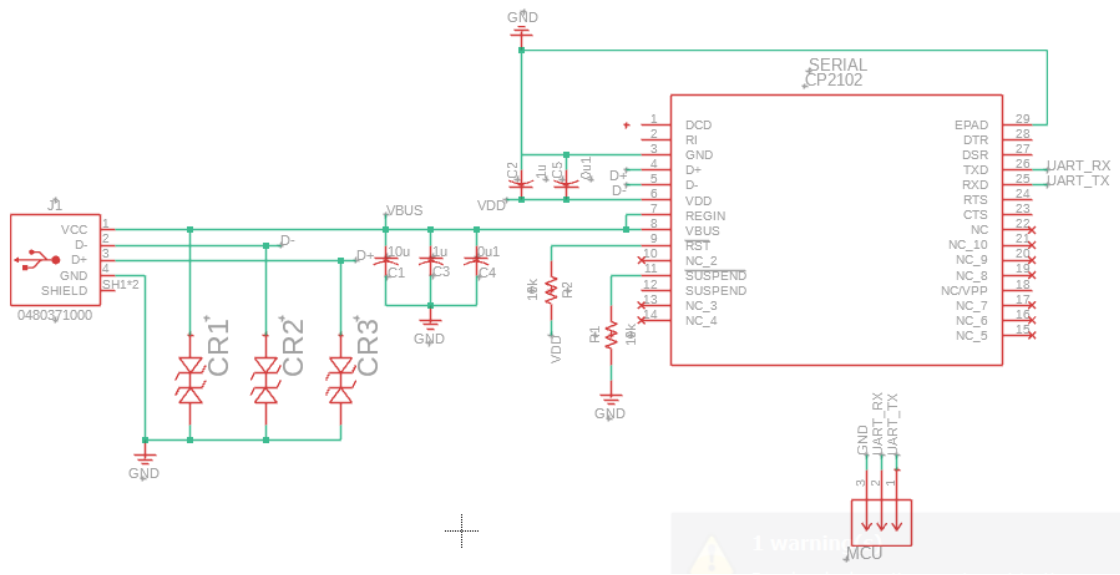


Fig. 19: USB-A to Serial Converter Schematic

7.3.2 Digital Pressure Sensor

In Fig. 20, the BMP280 digital pressure sensor is configured to operate in I²C mode using the SDA/SDA and SCK/SCL lines. The interface mode of the BMP280 is selected by the state of the CS pin, so R1 pulls CS to the 3.3 V rail through a 10 kΩ resistor to hold it high and place the device in I²C mode. Resistors R3 and R4, also 10 kΩ, serve as the bus pull-ups on SDA and SCL respectively, ensuring proper logic levels on the open-drain lines.

The BMP280 is powered through the 3.3 V rail, with both VDD and VDDIO tied to the same supply so that the core and interface voltage domains match the logic level of the host controller. A 100 nF decoupling capacitor (C1) is placed near the supply pins to reduce high-frequency noise and stabilize the supply during conversion cycles. Both ground pins are tied to the common system ground.

The sensor is broken out to headers J1, JP1, and JP2, which carry the 3.3 V, GND, SDA, and SCL nets. These provide access to the I²C bus for debugging and allow the sensor to be mounted remotely from the main board if required by the enclosure layout. This configuration minimizes wiring complexity and supports the low power consumption that is critical for integration into the L.I.N.K. system.

7.3.4 GNSS Receiver Module

In Fig. 22, the GNSS receiver module, based on the u-blox NEO-M9N, is supplied power through the 3.3 V rail with local decoupling to ensure stable operation during acquisition and tracking. The RF signal is received through the antenna input, allowing the module to acquire satellite signals. Supporting passive components are included to ensure proper operation of the GNSS IC. The module communicates position data to the ESP32 over a UART interface, enabling reliable location reporting within the L.I.N.K. system.

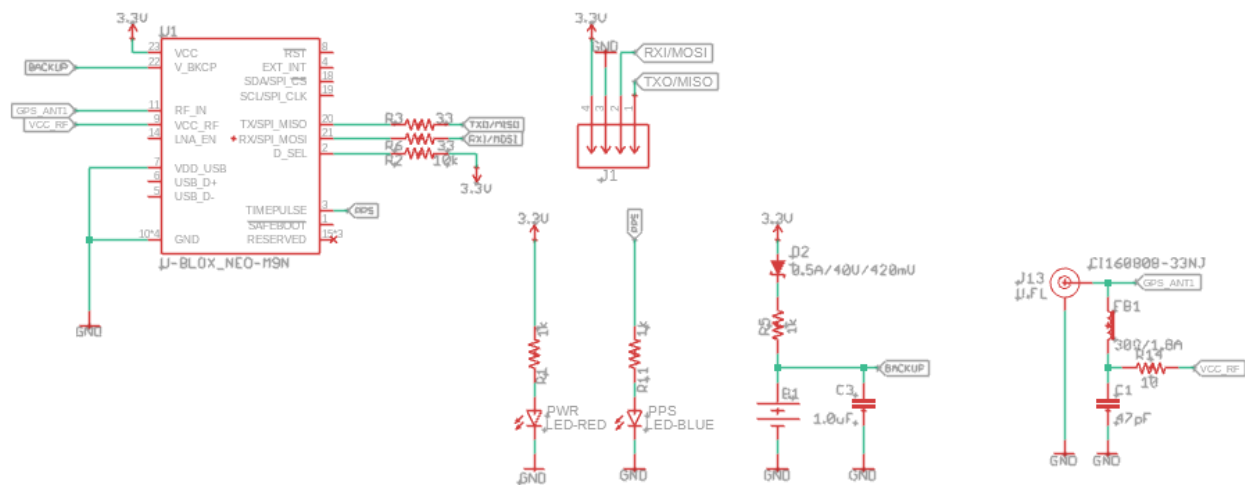


Fig. 22: GNSS Module Schematic for u-blox NEO-M9N Receiver

7.3.5 OLED Display (HUD)

In Fig. 23, the connector board for the OLED heads-up display is shown. The display module is a commercial off-the-shelf unit, so the board serves as a breakout that routes power and communication between the module and the main PCB rather than implementing the display driver itself. The module is supplied from the 3.3 V rail and communicates with the ESP32 over the shared I²C bus using the SDA and SCL lines. Grounds are tied to the common system ground. Routing the display through a dedicated connector board allows the HUD to be mounted separately from the main PCB as required by the enclosure layout, without adding wiring complexity to the L.I.N.K. system.

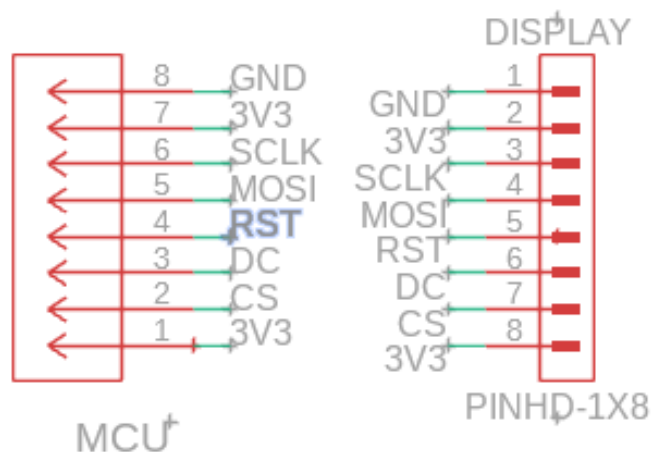


Fig. 23: OLED Display Connector Schematic

7.3.6 Button Interface

In Fig. 24, the button connector board is shown, which provides the user input interface for L.I.N.K. Each button is wired to a general purpose input pin on the ESP32, allowing the microcontroller to detect a press and trigger the corresponding function in firmware. The board carries the 3.3 V and ground nets alongside the signal lines so that the buttons can be mounted on the exterior of the enclosure while the logic remains on the main PCB. This configuration keeps the user controls accessible without routing the full peripheral bus off-board.

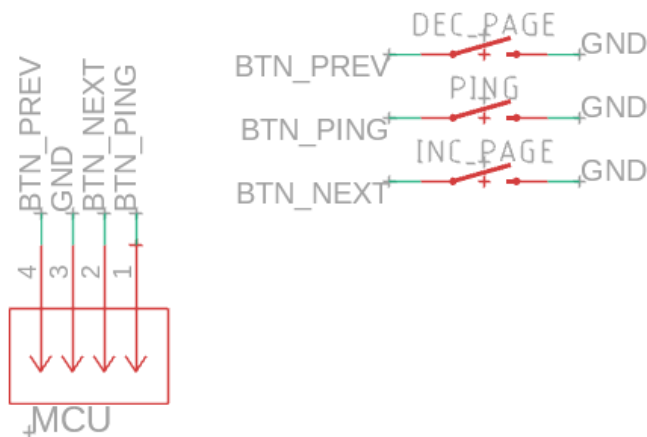


Fig. 24: Button Interface Connector Schematic

7.3.7 LiDAR Module

In Fig. 25, the connector board for the LiDAR module is shown. The LiDAR is a commercial off-the-shelf unit, so the board provides the power and communication interface between the module and the main PCB. Unlike the remaining peripherals, the LiDAR is supplied from the

5 V rail produced by the LM2596S-5.0 regulator, as the module requires a higher supply voltage than the 3.3 V logic domain. The module communicates ranging data to the ESP32 over UART, and its ground is tied to the common system ground.

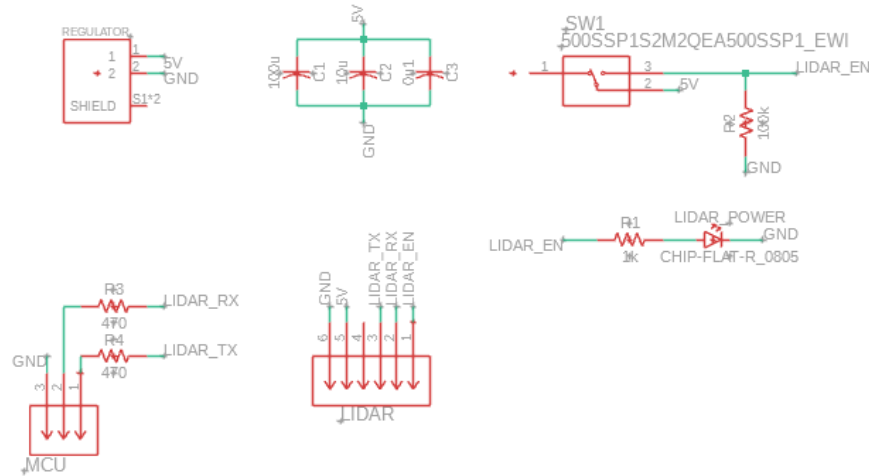


Fig. 25: LiDAR Module Connector Schematic

7.4 Lower Level Subsystem

7.4.1 MCU

In Fig. 26, the ESP32-WROOM-32 microcontroller controls communication with all peripherals and sensors in L.I.N.K. and is powered via the 3.3 V rail, utilizing decoupling capacitors to stabilize noise. The microcontroller interfaces with all peripherals and sensors via multiple communication protocols, including I²C, UART, and PWM. Other pins are also labeled for external peripherals such as the OLED HUD, IMU, and LiDAR. The I²C lines utilize pull-up resistors to ensure proper logic is maintained. This configuration ensures proper communication between all peripherals and overall functioning of L.I.N.K.

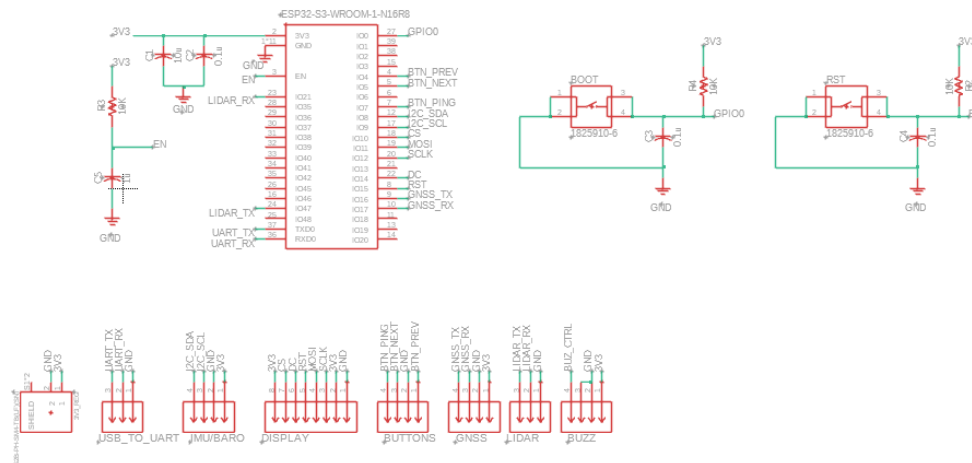


Fig. 26: ESP32 Microcontroller Schematic

8 Software Design

Two pieces of software keep L.I.N.K. running, and neither one is useful without the other. On the binocular side, an ESP32 firmware image handles sensor fusion, HUD rendering, and BLE communication, and it is also where each waypoint is born the instant the ping button is pressed. On the operator's phone, a Progressive Web Application picks up that BLE traffic, drops markers on a live map, and tucks the same waypoints away in local storage so that they can later be pushed to the cloud when connectivity allows. Nothing else passes between the two halves of the system: every byte travels over the BLE link as JSON, wrapped in a small acknowledgment protocol that exists for one reason, which is to make sure no waypoint is ever lost between the moment it is calculated and the moment it is stored. The remainder of this chapter walks through both applications, the data structures they share, the interfaces that join them, and the user-facing screens an operator will actually see in the field.

8.1 System Software Architecture

Architecturally the system is a three-tier stack. At the bottom sits the ESP32 firmware, which gathers sensor data, computes waypoints, and pushes JSON over BLE. Above that runs the phone-side PWA, which keeps the Bluetooth session alive, paints telemetry and pins on a map, caches recent activity locally, and manages an upload queue for whenever the network is available. The top tier is a small self-hosted Node.js/Express service backed by SQLite; its job is to validate incoming waypoint payloads, persist them, and reply with the ack the phone needs in order to close the loop back to the binoculars.

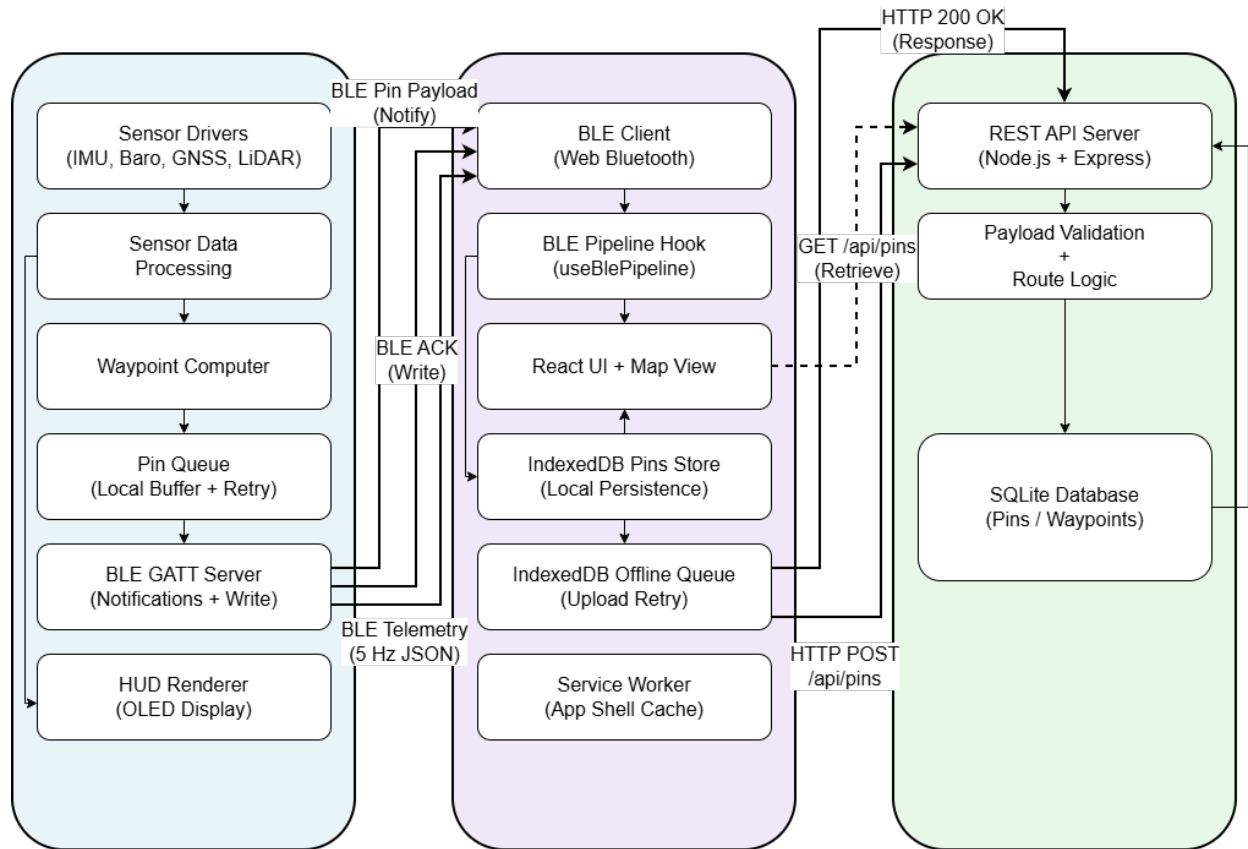


Fig. 27: Software Architecture Block Diagram

Each tier owns a clearly bounded slice of work. The embedded tier handles sensors, waypoint math, the on-device queue, and the HUD overlay. The client tier owns the BLE session, the offline-first cache of recent pins and map state, and the conversation with the server. The server tier handles validation, SQLite persistence, and the read endpoint that lets the phone repopulate the map after a fresh install. Crucially, none of the tiers depends on the others being available in real time. Drop the BLE link and the firmware keeps collecting telemetry and parking waypoints in its local buffer until the phone shows up again. Yank the phone off Wi-Fi or cellular and the PWA still drives the map, queues fresh pins to IndexedDB, and quietly drains the queue the moment a connection returns.

8.2 Firmware Design

From the moment power is applied, the firmware on the ESP32 is the only thing keeping the binoculars alive. It is written in C++ against the Arduino framework [37] and built with PlatformIO [42]. Rather than reach for an RTOS, we structured the program as one cooperative main loop in which every subsystem is scheduled by comparing the current millisecond clock against a per-task “last ran at” marker. The result is a non-preemptive design that sidesteps the usual concurrency headaches while still giving us comfortably better than millisecond timing

resolution for everything the loop has to touch. Annotated source listings of the firmware modules referenced in this section are reproduced in Appendix C.

8.2.1 Firmware Boot and Initialization Sequence

Before the main loop ever runs, the firmware works through a fixed startup script. Each step has to succeed (or fail in a known way) before the next one fires, which is what makes the boot phase deterministic. The script is summarized below.

- 1) Serial debug output is initialized at 115200 baud for development logging.
- 2) I2C on GPIO 8 (SDA) & GPIO 9 (SCL) at 400 KHz.
- 3) Sensor driver init order The order of initialization is IMU (BNO055 [38]), barometer (BMP280 [39]), GNSS (GT-U7 [72]), LiDAR (ToF rangefinder). All the driver initialization functions set up the device and check that communication is possible.
- 4) All sensor drivers include a self-test to verify the device's operational status.
- 5) Bring up the OLED screen and show a boot screen indicating the status of each module as Pass or Fail.
- 6) Our custom BLE GATT server [11] is instantiated with our custom L.I.N.K. service along with the characteristics: Telemetry, Pin Payload, and ACK. Advertising has begun immediately.
- 7) The main loop begins.

A failed self-test does not stop the boot. The offending module is simply tagged with a `FAIL` health state and the rest of the system carries on. That choice matters in the field: lose the barometer mid-mission and the device can still capture waypoints using GNSS, IMU, and LiDAR alone, which is far more useful than a brick. Every telemetry frame carries the current health flags forward to the phone, so the user always sees an accurate picture of which sensors are pulling their weight.

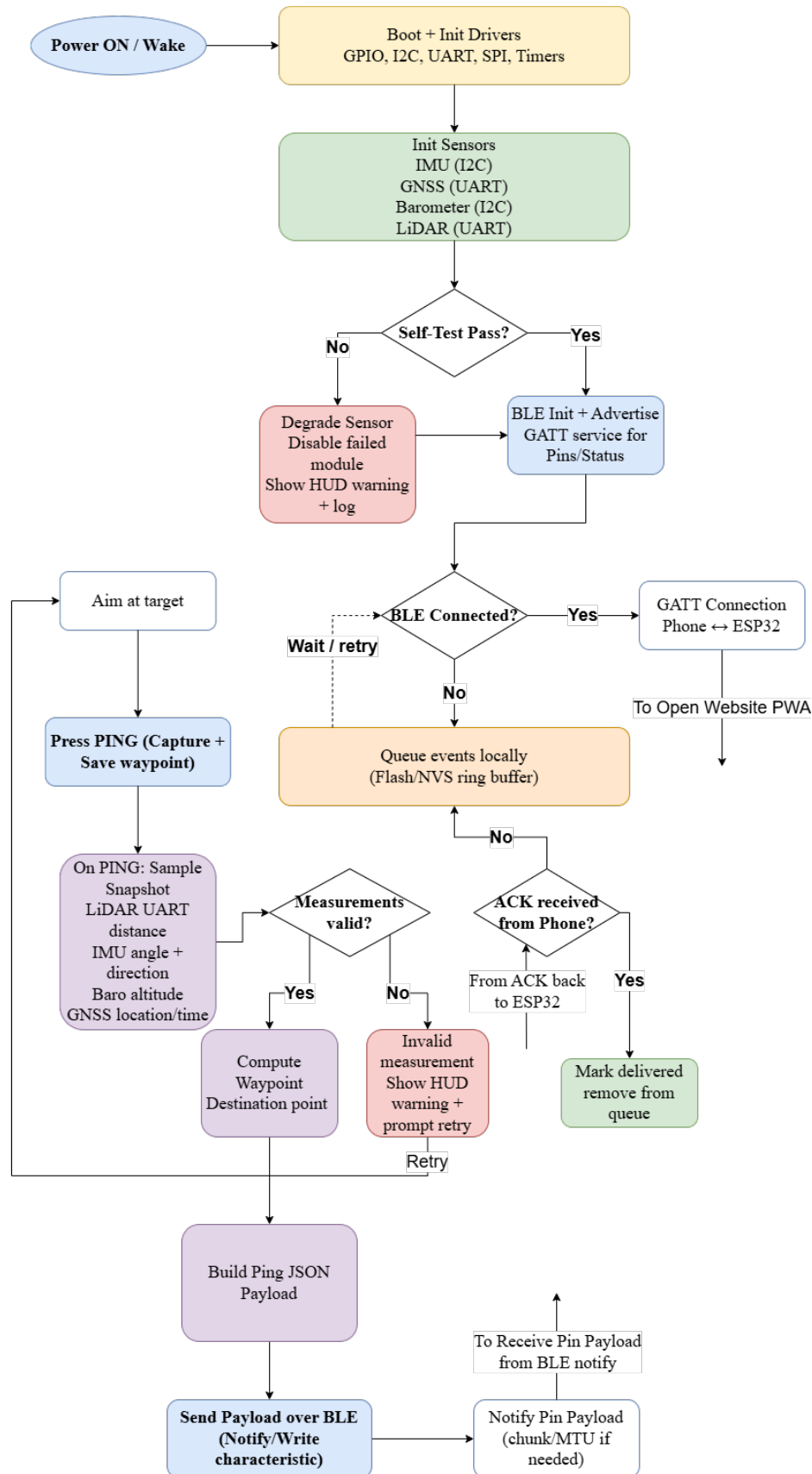


Fig. 28: Software Flowchart (Part 1): On-Device Waypoint Capture and BLE Transmit.

The flowchart traces the same boot sequence end-to-end: it opens at Power ON/Wake, threads through driver initialization and self-test, branches on the Pass/Fail outcome of each module, and finally drops into the BLE advertising state where the firmware sits waiting for a phone.

8.2.2 Main Loop State Machine

Once initialization is finished, control passes to a main loop that the firmware never leaves until power is removed. The loop is deliberately starved of any `delay()` or other blocking call; instead, each subsystem owns a “last ran at” timestamp and only executes when enough wall-clock time has passed. Five scheduling periods govern the day-to-day life of the device:

- Sensor polling: 50 ms (20 Hz) Read all sensor drivers and store the measurements in a shared TelemetrySnapshot.
- BLE telemetry streaming: every 200 ms (5 Hz). We are currently serializing the TelemetrySnapshot to JSON [12] and sending this as a BLE notification on the Telemetry characteristic.
- Ping button detection: On continuously with 250 ms debounce. When the ping button is detected, the waypoint capture sequence will begin.
- Queue retry: every 10 seconds. All unacknowledged waypoints in the local queue are retransmitted to the phone.
- OLED display refresh: every 500 ms (2 Hz), and the HUD overlay is re-rendered with the last sensor reading.

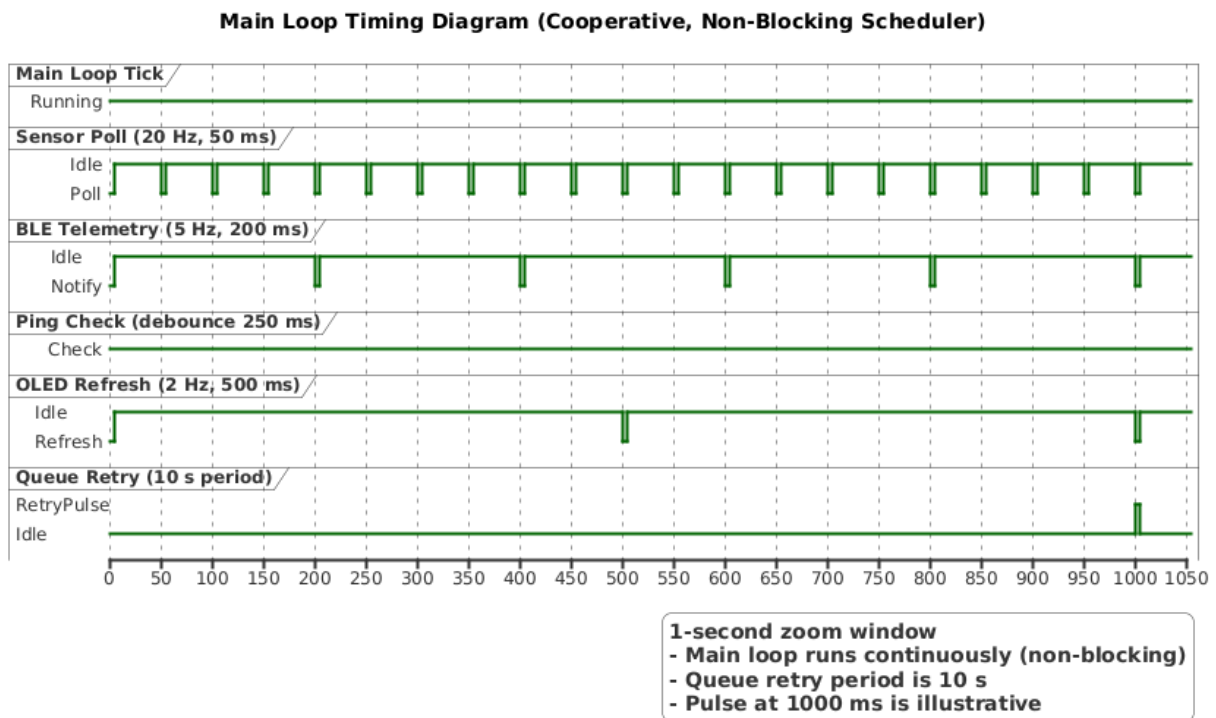


Fig. 29: Main Loop Timing Diagram

Nothing in the loop blocks, and the worst offender, an I²C read from the BNO055, finishes in under 3 ms. A typical pass through the entire loop wraps in well under 10 ms, which leaves more than 40 ms of slack inside every 50 ms sensor-polling window. In practice, no subsystem ever has to wait on another to finish.

8.2.3 Sensor Driver Architecture

Every sensor lives behind its own driver module, paired as a header and an implementation file (for example `imu_driver.h` and `imu_driver.cpp`). All four drivers expose the same three-call interface, which keeps the call sites in `main.cpp` uniform regardless of which silicon is on the other end:

- `init()` brings the hardware up and verifies the bus link, returning `true` only when the chip answers.
- `selfTest()` pulls a sample reading and sanity-checks it against a plausible range; a `true` return means the data path is alive end-to-end.
- `read()` returns a typed struct holding the latest measurement and the module's current health flag.

The shared driver header pattern and the typed sensor structures are listed in Appendix C. Those typed return structures all live in one shared header, `sensor_types.h`:

TABLE XXXIX: Sensor Data Structures returned by driver `read()` functions

Structure	Fields	Description
<code>ImuData</code>	<i>Position:</i> heading, pitch, roll <i>Calibration:</i> <code>calibSys</code> , <code>calibGyro</code> , <code>calibAccel</code> , <code>calibMag</code> <i>Status:</i> <code>state</code>	Euler orientation from BNO055 fusion engine. Calibration values 0–3.
<code>GnssData</code>	<i>Position:</i> lat, lon, alt, accuracy <i>Signal:</i> <code>sats</code> <i>Quality:</i> <code>fix</code> <i>Status:</i> <code>state</code>	Position from GT-U7 GPS. Fix indicates quality (none/2d/3d/dgps).
<code>BaroData</code>	<i>Readings:</i> <code>pressureHpa</code> , <code>tempC</code> , <code>altM</code> <i>Status:</i> <code>state</code>	Atmospheric readings from BMP280. Altitude estimated from pressure.
<code>LidarData</code>	<i>Range:</i> <code>rangeM</code> <i>Signal:</i> <code>quality</code> , <code>valid</code> <i>Status:</i> <code>state</code>	Distance from ToF rangefinder. Quality indicates signal strength.

A small `ModuleState` enum carries the per-sensor health flag through the system. It admits three values: `OK` for a sensor that is talking and producing trustworthy numbers, `DEGRADED` when the chip is responding but the data quality has slipped, and `FAIL` when the module has gone silent or is returning nonsense. We re-evaluate this flag on every read cycle and forward it untouched into the next telemetry frame.

Each driver also carries a compile-time switch (`HAS_IMU`, `HAS_BARO`, `HAS_GNSS`, `HAS_LIDAR`). Setting the switch to zero drops the driver out of the build entirely; the call sites then resolve to a stub that returns defaults and reports failure. That escape hatch was invaluable during hardware bring-up because we could deliberately quiet a misbehaving sensor without touching its callers, and it doubles as a clean degradation path if a board ships with a missing component.

8.2.4 Waypoint Capture Sequence

Of every operation the firmware performs, the waypoint capture is the one that has to be right. It takes the snapshot of raw sensor data that exists at the instant the user presses the ping button and turns it into a fully formed, geolocated pin record. The chain of events looks like this:

- 1) Button press is detected and debounced (250 ms minimum interval between pings).
- 2) The current `TelemetrySnapshot` is captured, freezing the sensor state at the moment of the ping.
- 3) Measurement validity is checked: the GNSS must have a fix, and the LiDAR must have a valid range return. If either condition fails, the HUD displays a warning message and the capture is aborted.
- 4) The target waypoint coordinates are computed using the Haversine forward projection [73]. The observer's GNSS position is projected forward along the bearing reported by the IMU using the LiDAR range, with optional elevation correction using the IMU pitch angle and barometric altitude.
- 5) A unique identifier is generated for the pin record.
- 6) The pin payload is assembled as a JSON object containing the observer position, target position, aiming data (bearing, pitch, range, signal quality), metadata (timestamp, label), and the full telemetry snapshot.
- 7) The pin payload is enqueued in the local circular buffer (capacity: 32 entries).
- 8) If a BLE connection is active, the payload is immediately transmitted as a notification on the Pin Payload characteristic.
- 9) The HUD displays a confirmation indicator.

Figure 9 shows this same path graphically: starting from the “Press PING” node, the diagram threads through measurement validation, the haversine target calculation, JSON assembly, and the BLE notify that hands the pin to the phone. The corresponding firmware code that performs the haversine projection in step 4 is reproduced in Appendix C.

8.2.5 Local Queue and Retry Logic

Waypoints captured while the BLE link is down still have to survive, and that is the entire purpose of the local pin queue. We implement it as a 32-slot RAM-resident circular buffer, with each slot holding a serialized JSON payload alongside a single ack flag.

A new pin always lands in the next free slot. If every slot is already occupied by an unacknowledged payload, the oldest one is overwritten. Yes, that is technically data loss, but in practice it requires a user to capture 32 separate waypoints without the phone ever connecting and ack-ing a single one, which is a corner of the failure space we were comfortable accepting.

Every ten seconds a retry timer fires. On each pass the firmware walks the queue and re-notifies every entry that still has not been acknowledged. When the phone eventually finishes processing a pin and confirms it has been stored on the server, it writes back an ACK message containing that pin's ID through the ACK characteristic. The firmware's BLE write callback matches the ID against the queue, flips the corresponding entry's ack flag, and the next retry pass quietly compacts those acknowledged entries out of the buffer. The full enqueue, ACK, and prune logic is listed in Appendix C.

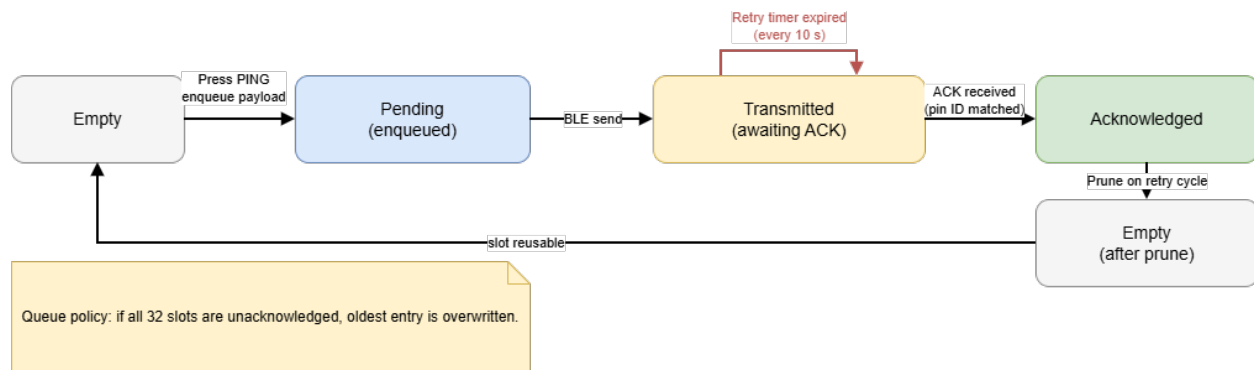


Fig. 30: Queue State Diagram

8.2.6 HUD Display Design

The HUD itself is a small SSD1306 [40] OLED, 128×64 pixels, glued in front of the binocular eyepiece and driven over I²C [17]. The firmware repaints it twice a second using the Adafruit SSD1306 and GFX libraries [41], [74]. We laid the screen out as a single dashboard so that everything an operator needs while still looking through the optics is visible without scrolling:

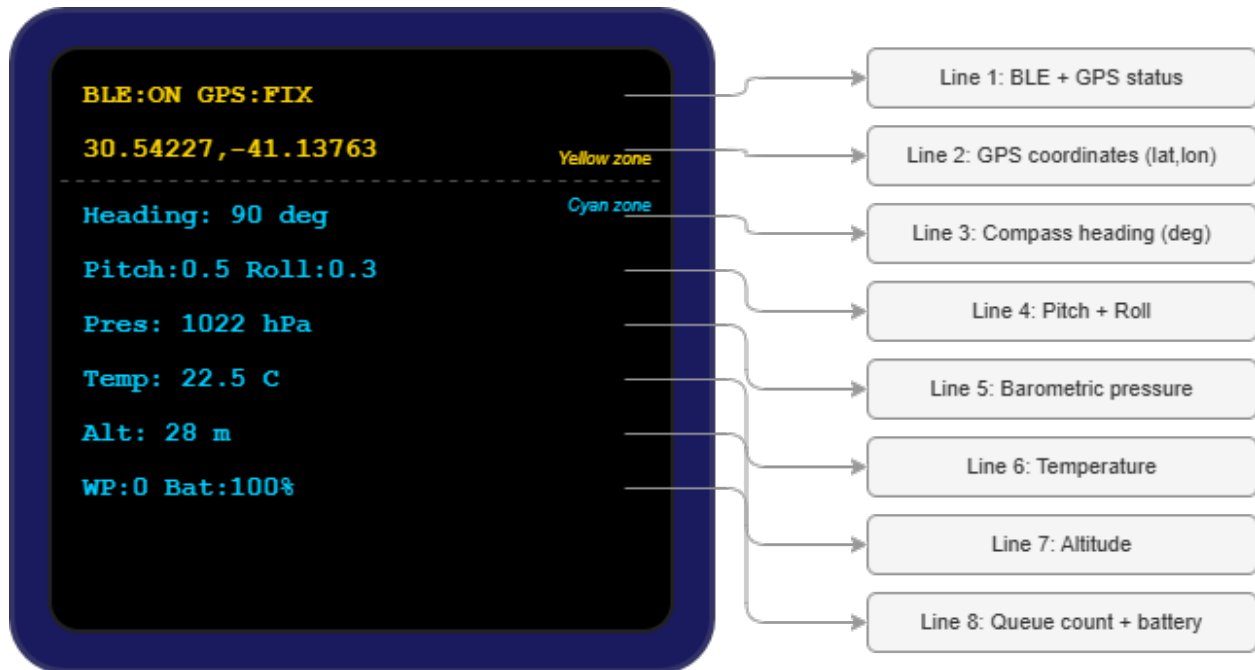


Fig. 31: HUD Layout Diagram



Fig. 32: Real HUD Layout

- Line 1: BLE connection status (icon or text) and GPS fix status
- Line 2: Latitude, longitude
- Line 3: Compass heading in degrees

- Line 4: Pitch and roll angles
- Line 5: Barometric pressure in hPa
- Line 6: Temperature in degrees Celsius
- Line 7: Barometric altitude in meters
- Line 8: Queued waypoint count and battery percentage

The whole layout is rendered in the smallest monospace font we could lay hands on, a 6×8 pixel face, simply because that is what makes the most information fit inside 128×64 . Even short status fragments such as “BLE” or “GPS” get abbreviations on their lines so the rest of the row stays usable. When a sensor enters `FAIL`, its line collapses to “---” rather than risk showing a misleading number.

The HUD also doubles as live capture feedback. After a successful waypoint capture the entire screen flashes “PIN SAVED” for a second before falling back to the dashboard. If the capture is rejected because the GNSS lacks a fix or the LiDAR has no return, the flash reads “NO FIX” or “NO RANGE” instead, and the operator simply tries again.

8.3 Phone Application Design (PWA)

On the phone side, the user-facing software is a Progressive Web Application built in React [54] and TypeScript [51] and bundled by Vite [57]. We chose a map-centric layout because that is what an operator actually wants in front of them. Surrounding the map are the live binocular telemetry, a list of recent pings, the current BLE connection state, and the user’s own GPS position. Because the whole thing ships as a PWA, it can also be installed to the home screen, hold its own when the network drops via a service worker [14], and persist waypoint data through IndexedDB [50].

8.3.1 Application Architecture

Internally the PWA splits cleanly into four layers: a thin UI component layer, a single state-management hook, a small service layer that owns external I/O, and a persistence layer on top of IndexedDB. The layers talk through React state and callbacks, and a UI component never reaches directly into a service or the database.

The **UI component layer** is essentially two React components doing most of the visible work. `MapView` renders a full-screen MapLibre GL JS [52] canvas and lays the user’s location, the binocular’s location, and every captured waypoint marker on top of it. `BleStatus` carries the BLE connect button, the live telemetry badges (battery, per-module health), the pending-queue counter, and any error toasts that need surfacing. The parent `App` component is the glue: it owns the shared state, threads the recent-pings list down to the children, and exposes the center and follow controls.

The **hook layer** is a single custom hook, `useBlePipeline`, and it is where the entire data pipeline really lives. The hook subscribes to BLE events, normalizes incoming pin payloads into the internal pin shape, writes new pins to IndexedDB, fires off the local notification, and kicks

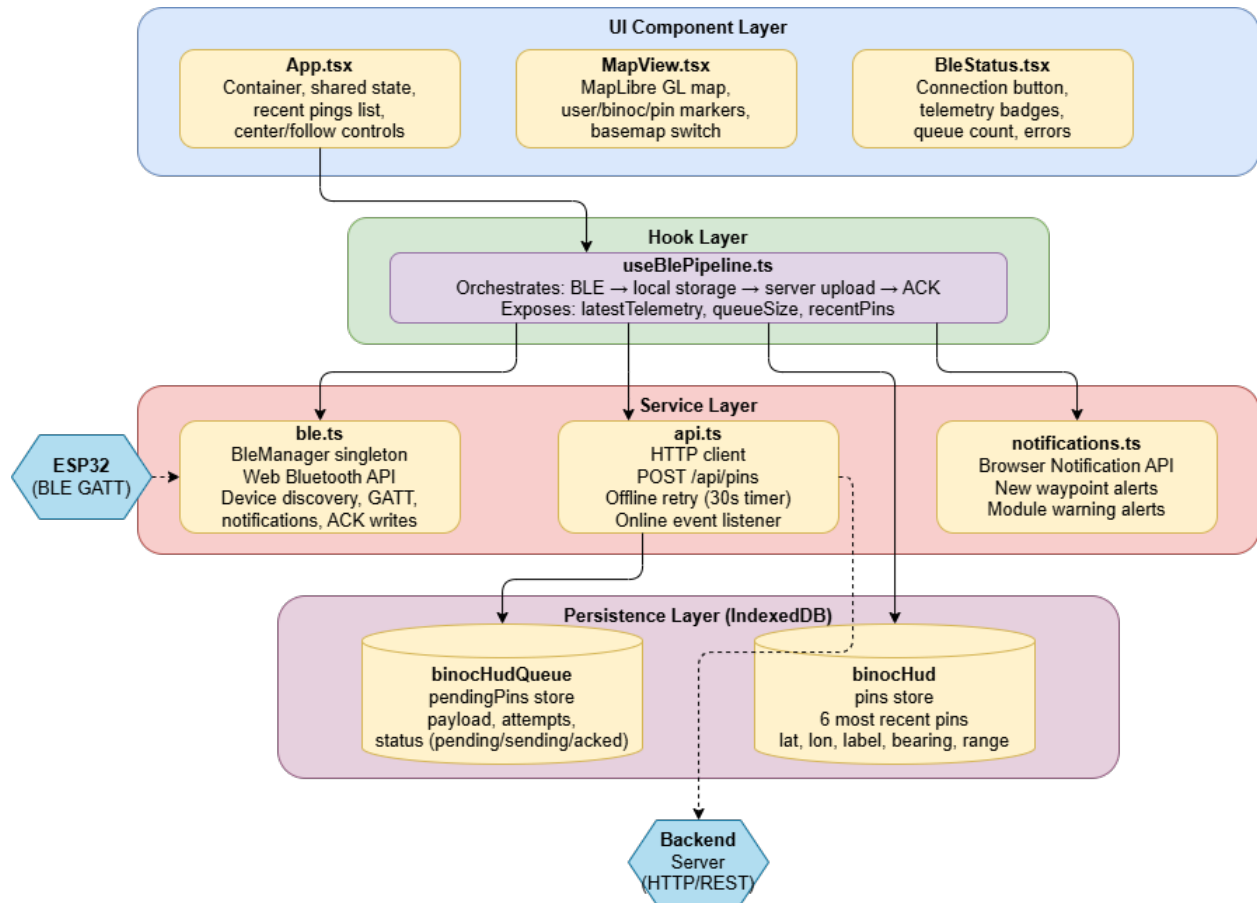


Fig. 33: PWA Component Architecture Diagram

off the server upload. Whatever the rest of the React tree needs to render — the latest telemetry snapshot, the live queue size, the recent pins — comes back out through the hook's exposed state.

The **service layer** is split into three modules:

- **ble.ts**: A singleton `BleManager` class that wraps the Web Bluetooth API [48]. It handles device discovery, GATT connection management, characteristic subscription and notification handling, and acknowledgment writes. The manager emits typed events (`connectionChange`, `telemetry`, `pinPayload`, `error`, `moduleWarning`) that the hook layer consumes.
- **api.ts**: The HTTP client responsible for posting pin payloads to the backend server and managing the offline retry queue. It validates incoming payloads, enqueues them to IndexedDB, attempts immediate upload, and retries pending entries every 30 seconds or on browser online events.
- **notifications.ts**: A thin wrapper around the browser Notification API [75] that requests permission during first use and dispatches local push notifications for new waypoints and module health warnings.

The **persistence layer** sits on top of two separate IndexedDB databases, both accessed through the `idb` library [76]:

- `binocHudQueue`: Stores the offline upload queue with fields for payload, attempt count, last attempt timestamp, and delivery status (`pending`, `sending`, or `acked`).
- `binocHud`: Stores the six most recent pins displayed on the map, including target coordinates, observer coordinates, bearing, range, and LiDAR quality metadata.

Source excerpts for the PWA service modules and the haversine forward projection `geo.ts` are listed in Appendix C.

8.3.2 BLE Communication Flow

The PWA never talks to the ESP32 except through GATT, and the conversation is partitioned across the three characteristics that the L.I.N.K. service exposes:

- 1) **Telemetry channel (Notify, 5 Hz)**: The phone subscribes to the Telemetry characteristic upon connection. Every 200 ms, the ESP32 sends a JSON notification containing the complete sensor snapshot. The `BleManager` decodes the `DataView` buffer using `TextDecoder`, parses the JSON, and emits a `telemetry` event. The hook layer stores this in React state, triggering UI re-renders. On each telemetry frame, the manager also inspects the `modules` object; if any module reports a status other than `"ok"`, a `moduleWarning` event is emitted to trigger a push notification.
- 2) **Pin Payload channel (Notify, on demand)**: When the ESP32 captures a waypoint, it sends the full pin payload as a notification on the Pin Payload characteristic. The `BleManager` parses this payload and emits a `pinPayload` event. The `useBlePipeline` hook receives the event, converts the payload to the internal pin shape, persists it to IndexedDB, triggers a “New Waypoint Received” notification, and initiates the server upload pipeline.
- 3) **ACK channel (Write)**: After the phone successfully uploads a pin payload to the backend server and receives a `200 OK` response with `{ok: true}`, it writes an acknowledgment message to the ACK characteristic. This message is a JSON object containing `{ack: true, pinId: "UUID"}`. The ESP32’s BLE write callback matches the `pinId` to its local queue and marks the entry as acknowledged.

The connection itself is run as a small four-state machine: `disconnected`, `connecting`, `connected`, and `error`. The user kicks things off by tapping the “Connect ESP32” button, which fires `navigator.bluetooth.requestDevice()` with the L.I.N.K. service UUID as the filter. Once the GATT server is reached, the manager subscribes to notifications on both Telemetry and Pin Payload and grabs a writable handle to the ACK characteristic. If the server later disappears — for example because the binoculars walked out of range — the manager drops back to `disconnected` and emits a `connectionChange` event so the UI can prompt the user to retry.

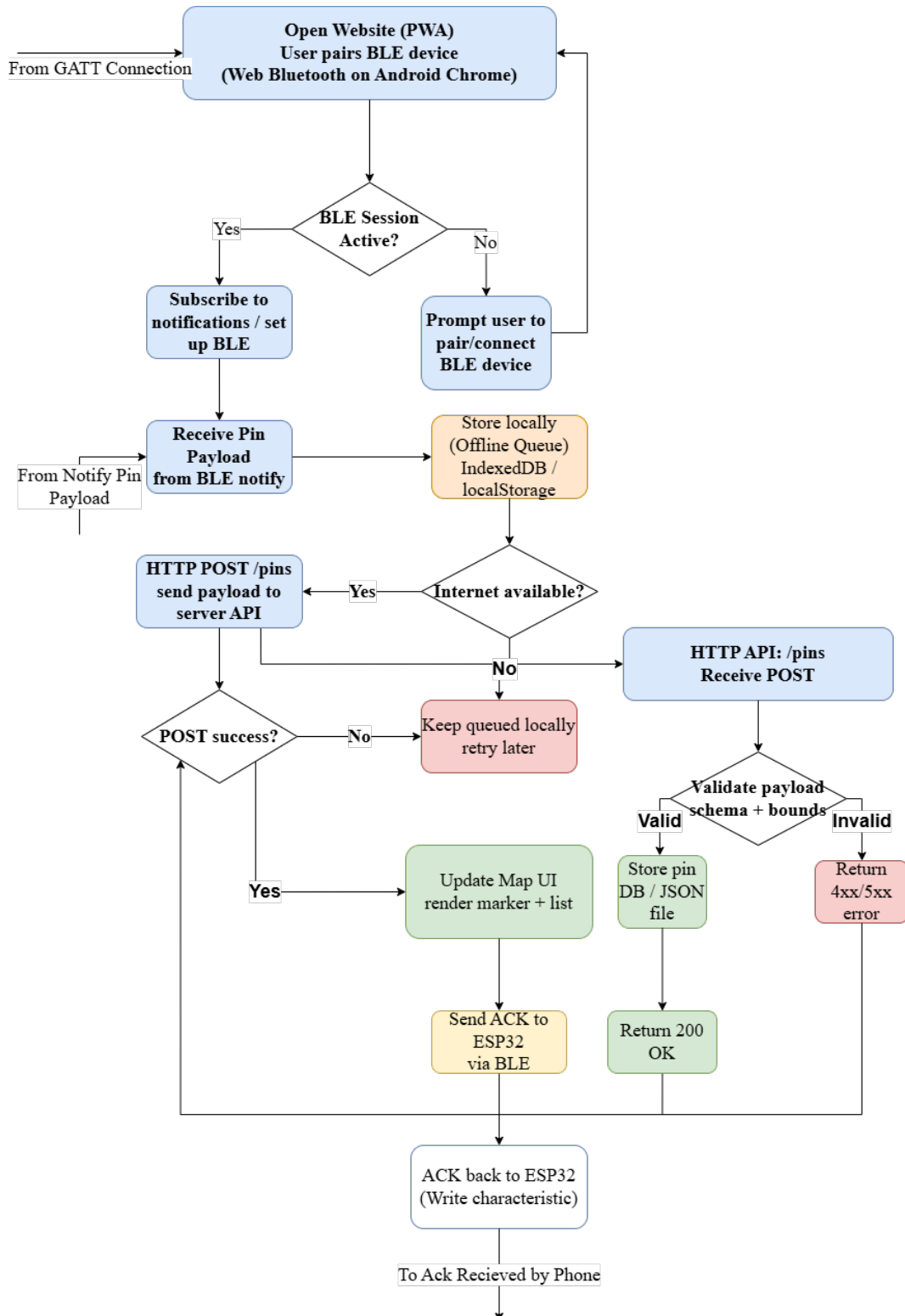


Fig. 34: Software Flowchart (Part 2): Phone PWA Processing, Offline Queueing, Server Upload, and ACK Return.

The figure walks through that same end-to-end flow on the phone side. It starts the moment the operator opens the PWA and pairs the binoculars over Web Bluetooth in Chrome on Android. If a BLE session is already up, the app subscribes to the GATT notifications immediately; otherwise it nudges the user to connect first. As soon as a pin payload arrives over notify, it is dropped into IndexedDB so that the offline queue is the source of truth even before the network is consulted. From there the app checks for connectivity: when online it issues an HTTP POST to the server API, the server validates the schema and bounds, persists the row, and the map UI gains a new marker while the phone fires its ACK back over the BLE write characteristic. When the network is missing or the POST fails, the payload simply stays parked in the local queue for the retry loop to revisit, which is exactly what the next subsection describes.

8.3.3 Offline-First Data Pipeline

The PWA was built offline-first from day one because the alternative — losing waypoints because someone wandered out of LTE coverage — was simply not acceptable. The pipeline that protects against that scenario decouples receiving a pin from delivering it to the server, with explicit state at every hop.

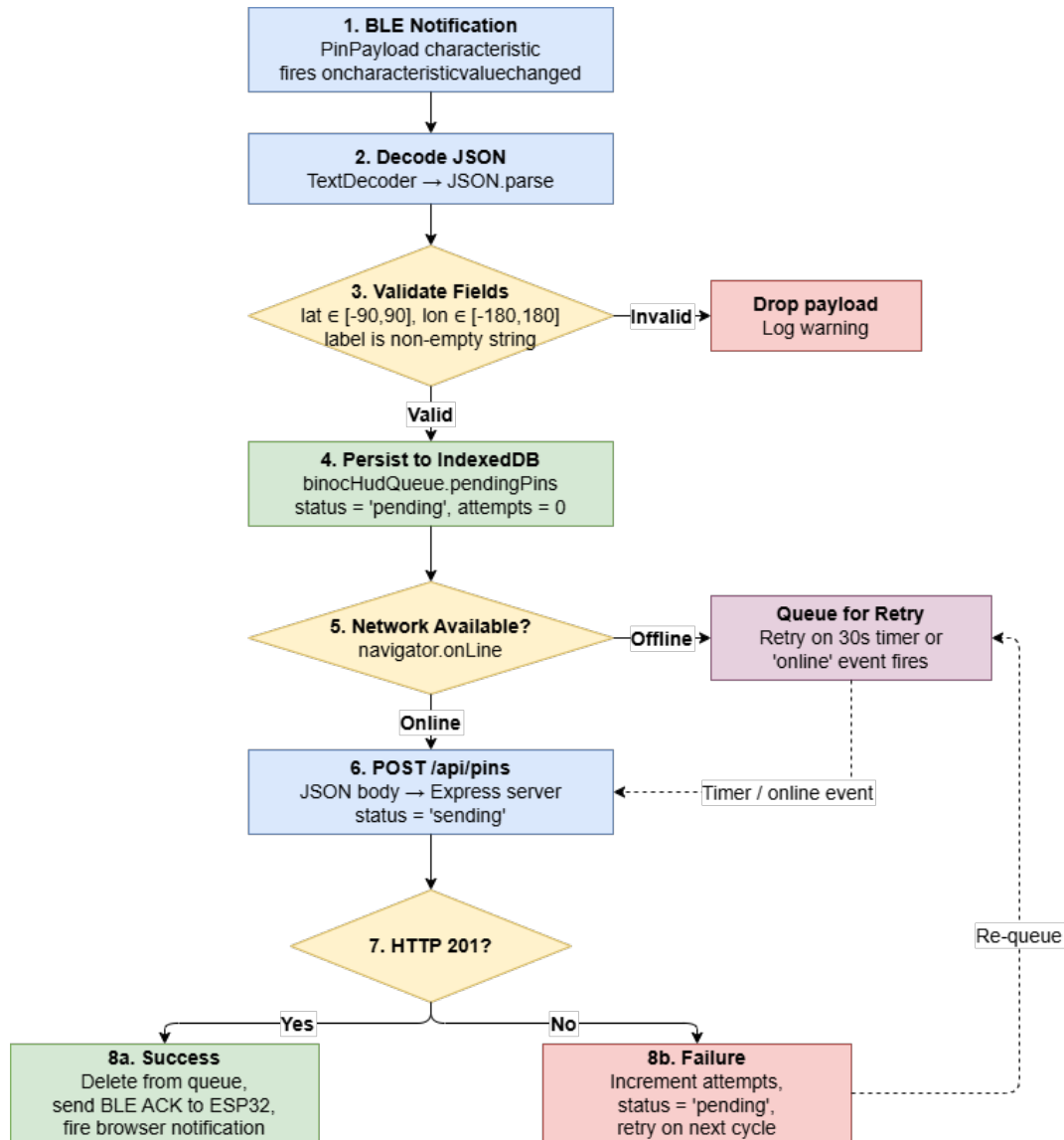


Fig. 35: Offline-First Data Pipeline Flowchart

When a payload arrives over BLE, the pipeline runs through these stages:

- 1) **Validation:** The payload is checked for required fields (`id`, `observer.lat`, `observer.lon`, `target.lat`, `target.lon`). Malformed payloads are logged and discarded.
- 2) **Local persistence:** The validated payload is written to the `pendingPins` object store in IndexedDB with an initial status of "pending" and an attempt counter of zero.
- 3) **Immediate upload attempt:** The pipeline immediately attempts to POST the payload to the server at `/api/pins`. The entry status is updated to "sending" and the attempt counter is incremented.
- 4) **Success path:** If the server responds with 200 OK and `{ok: true}`, the entry status is updated to "acked". The phone then sends a BLE ACK back to the ESP32, completing the end-to-end delivery. Acked entries older than 24 hours are pruned from IndexedDB.

automatically.

- 5) **Failure path:** If the POST fails due to network error, server error, or timeout (10-second threshold), the entry status is reverted to "pending". The entry remains in the queue for retry.

Two triggers drive the retry behind that pipeline. A background timer fires every 30 seconds and walks every "pending" entry, attempting an upload on each. In parallel, the browser's online event flushes the queue immediately whenever connectivity comes back. The two triggers together keep the gap between "the network returned" and "the server has the pin" as short as possible.

For visibility, the `BleStatus` component polls the queue length every five seconds and renders the current count in the status row, so the operator can always see at a glance how many waypoints are still waiting to ship.

8.3.4 Map View and User Interface

The map dominates the phone-side UI, and it is rendered with MapLibre GL JS [52], an open-source WebGL mapping engine. The interface itself went through two distinct shapes before reaching its current form. The first build was a desktop client used as a development sandbox: a wide-aspect MapLibre canvas down the middle, a left-hand "LINK MAP" panel for BLE connection and basemap controls, a right-hand "Recent Pings" panel, and a simulated-ping button that let us exercise the full server pipeline before the BLE link was even ready. That desktop layout became the reference point for every UI primitive we eventually carried forward — the marker styling, the control panel, the recent-pings list, the basemap toggle, the GPS accuracy halo — and is shown in Figure 36.



Fig. 36: Desktop Web Client (Early Development Build) Showing the Original LINK MAP Layout, Recent Pings Panel, and Simulated-Ping Control

With the desktop client validating the data flow end-to-end, the same component tree was simply

re-laid out for portrait phone screens. The wide two-column desktop layout collapsed downward: the BLE status bar moved to the top edge, the map took over the upper two-thirds of the screen, and the Recent Pings panel slid in beneath it. Marker semantics, basemap toggling, and the BLE control affordances all carried over verbatim from the desktop build.

The map fills the screen and supports two basemap sources the user can swap between:

- **OpenStreetMap:** Raster tiles [77] at 256 pixels per tile, supporting zoom levels 0 through 19. This provides a standard street and terrain view suitable for general navigation.
- **Esri World Imagery:** Satellite imagery tiles for terrain recognition and landmark identification, which is particularly useful when correlating visual observations through the binoculars with the digital map.

Three families of markers ride on top:

- **User position (blue dot):** Derived from the browser's `navigator.geolocation.watchPosition()` API [49]. An accuracy circle with a blue fill and outline is drawn around the marker to indicate GPS uncertainty.
- **Binocular position (green dot):** Derived from the `gnss.lat` and `gnss.lon` fields in the latest telemetry snapshot. This marker is only displayed when the ESP32 reports a valid GPS fix.
- **Waypoint pins (red markers):** Up to six recent waypoint markers are displayed, each positioned at the computed target coordinates. Tapping a pin opens a detail modal showing the observer coordinates, target coordinates, bearing, range, LiDAR quality, and timestamp.

A “Follow Binoculars” toggle changes how the map behaves between updates. With it on, the camera re-centers on the binocular’s GPS position every time fresh telemetry arrives. With it off, the map sits wherever the user last dragged it. Independent of either, a “Center Me” button snaps the map back to the phone’s own GPS at any time.

Pinned to the bottom of the screen is a teleprompter-style list of the six most recent waypoints, each annotated with its label, coordinates, and haversine distance from the operator’s current position. We format distances in meters below 1000 m and switch to kilometers with two decimals beyond that, so the list reads cleanly whether the target is across a parking lot or across a campus.

Figures 37 through 40 capture the deployed PWA running on an Android phone. Figure 37 is the landing screen right after install, with the “Connect ESP32” button in its idle state and no live telemetry yet. Figure 38 shows the same app once a BLE session is up and running, with the user-position dot, the binocular-position dot, and a handful of captured waypoint markers all sharing an OpenStreetMap basemap. Figure 39 is the waypoint detail modal that surfaces when the operator taps a pin, exposing observer coordinates, target coordinates, bearing, range, LiDAR quality, and the capture timestamp side by side. Figure 40 catches the `BleStatus` badge row mid-outage: with the internet deliberately disabled, the pending-queue counter is visible and the captured waypoints remain on screen for the operator while they wait their turn to ship.

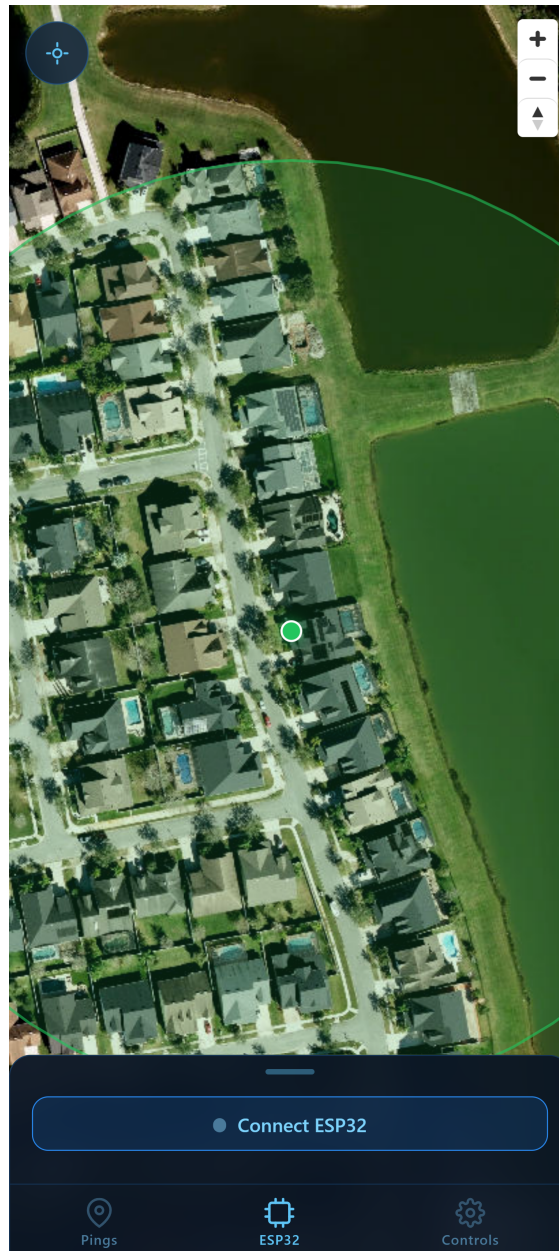


Fig. 37: PWA Landing View with BLE Connect Control

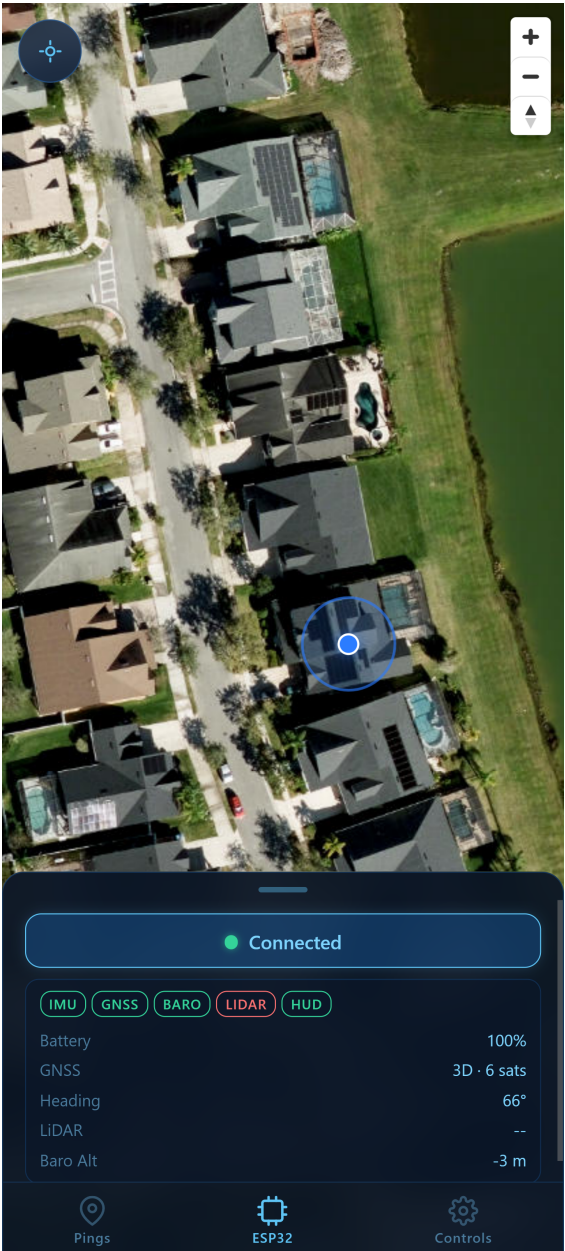


Fig. 38: PWA Map View with User, Binocular, and Waypoint Markers

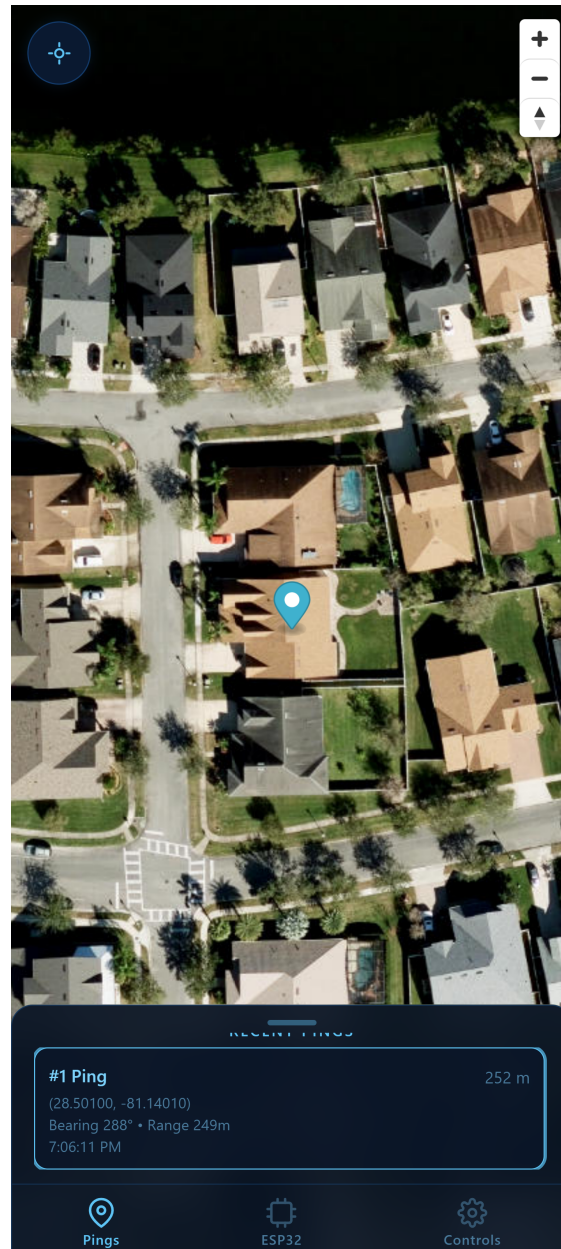


Fig. 39: Waypoint Detail Modal Showing Captured Sensor Context

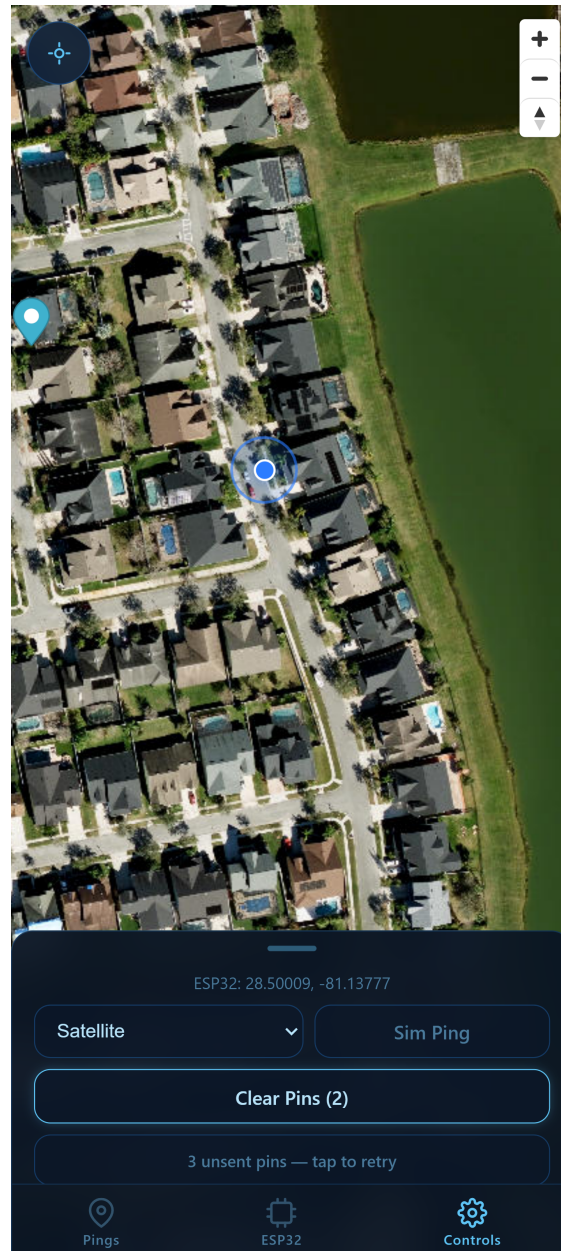


Fig. 40: BLE Status Panel Showing Pending-Queue Count During Simulated Offline Operation

8.3.5 PWA Installation and Service Worker

Shipping as a PWA gives us two practical wins: the app installs to the home screen like any native one, and it survives the loss of the network. Behind that experience are just two pieces of plumbing — the web app manifest and the service worker.

The **web app manifest** (`manifest.json`) is what tells the browser the app exists in the first place. It declares the name (“L.I.N.K. HUD”), the display mode (`standalone`, so the app runs without browser chrome), the theme and background colors, the start URL, and the icon set.

With those fields in place, the browser will offer the user an “Add to Home Screen” prompt, which is the moment the PWA stops feeling like a webpage.

The **service worker** (`sw.js`) is registered the first time the app loads, and it earns its keep in two ways:

- **App shell caching:** The service worker caches the HTML, CSS, JavaScript, and static assets required to render the application. This allows the PWA to load and display its interface even when the device has no internet connectivity.
- **Notification support:** The service worker enables push notification delivery even when the application tab is in the background. This is used to alert the user when a new waypoint is received or when a sensor module enters a degraded or failed state.

One important caveat: the service worker does *not* cache map tiles or any other dynamically fetched content. Whether tiles are available offline depends entirely on network state and whatever the browser has already cached on its own. The offline-first data pipeline described in Section 8.3.3 is a separate mechanism that works through IndexedDB and does not depend on the service worker at all.

8.4 Server Design

The backend gives waypoints a permanent home and acts as the synchronization point that the phone talks to. We kept it deliberately small: a Node.js [55] process running Express.js [56] for HTTP routing [15] on top of better-sqlite3 [78], [58] for storage. In production it lives on a dedicated Debian host under PM2, which handles automatic restarts and bringing the service up after a reboot. The same deployment also serves the static build of the PWA, so a single host carries both the API and the client.

8.4.1 API Endpoint Design

The HTTP surface is intentionally tiny — three endpoints in total:

TABLE XL: Server API Endpoints

Method	Path	Auth	Description
POST	/api/pins	None	Receives a pin payload from the phone. Validates required fields, inserts into the database, and returns <code>{ok: true, pinId}</code> .
GET	/api/pins	None	Returns all stored pins ordered by creation time (newest first), limited to 100 records. Response format: <code>{ok: true, pins: [...]}</code> .
GET	/api/health	None	Liveness probe that returns <code>{status: "ok"}</code> . Used during deployment to verify the server is running.

Before any row is written, the POST handler runs the payload through a short list of checks:

- The `id` field must be present and non-empty.

- The `observer.lat` and `observer.lon` fields must be present.
- The `target.lat` must be within the range $[-90, 90]$ and `target.lon` within $[-180, 180]$.
- If any validation check fails, the server responds with HTTP 400 and an error description.

The actual insert uses `INSERT OR IGNORE` keyed on the pin's `id` column, which is what makes the endpoint idempotent. If the phone retries a pin that has already landed (because, say, the ack was lost on the way back), the duplicate quietly disappears instead of producing a constraint violation. That property is not just convenient; it is what allows the offline-first retry mechanism in Section 8.3.3 to be aggressive without ever having to worry about poisoning the database. The complete validation logic and prepared-statement insert are reproduced in Appendix C.

Every database operation goes through a prepared statement that is compiled once at server startup and executed with bound parameters thereafter. The prepared-statement path is faster than re-parsing SQL on each request, but the larger reason we use it is that user-supplied values never get spliced into a query string, which closes the door on SQL injection completely.

8.4.2 Database Schema

A single SQLite file, `linkhud.db`, holds the entire dataset. We open it in Write-Ahead Logging (WAL) mode so that reads don't block while a write is in flight. The schema itself is one table:

TABLE XLI: Database Schema: `pins` Table

Column	Type	Description
<code>id</code>	TEXT PRIMARY KEY	UUID generated by the ESP32 at capture time.
<code>received_at</code>	INTEGER NOT NULL	Epoch-millisecond timestamp recorded by the phone at the moment the pin was received over BLE.
<code>created_at</code>	TEXT	Server-side insertion timestamp (SQLite <code>datetime('now')</code>).
<code>observer_lat</code>	REAL	Observer latitude from GNSS.
<code>observer_lon</code>	REAL	Observer longitude from GNSS.
<code>observer_alt</code>	REAL	Observer altitude in meters.
<code>observer_acc</code>	REAL	Horizontal accuracy estimate in meters ($\text{HDOP} \times 5$).
<code>target_lat</code>	REAL	Computed target latitude.
<code>target_lon</code>	REAL	Computed target longitude.
<code>target_alt</code>	REAL	Estimated target altitude.
<code>bearing_deg</code>	REAL	IMU heading at time of capture (degrees).
<code>pitch_deg</code>	REAL	IMU pitch at time of capture (degrees).
<code>range_m</code>	REAL	LiDAR range to target (meters).
<code>lidar_quality</code>	INTEGER	LiDAR signal strength (0–255).
<code>label</code>	TEXT	User-assigned or default waypoint label.
<code>telemetry</code>	TEXT	Full telemetry JSON snapshot at capture time.

The `telemetry` column is a single TEXT blob that holds the entire telemetry snapshot serialized as JSON. That is, on its face, a denormalization choice, but it is a deliberate one: it guarantees that every contextual detail about the system at the instant of capture rides alongside the waypoint record forever. For post-mission analysis, sensor calibration sanity checks, or simply trying to reconstruct what the operator was looking at when they pressed the button, having that snapshot already there is enormously more useful than chasing it across additional tables.

8.4.3 Authentication and Security

The current deployment does not implement application-level user authentication, and that is a deliberate choice rather than an oversight. The server runs on a private network reachable only through a Tailscale VPN mesh, and the phone client connects to it across that encrypted tunnel. In a deployment shaped like this one, the network boundary is the primary access control.

Even without login screens, several layers of defense are still in play:

- **Input validation:** All POST payloads are validated for required fields and range constraints before database insertion. Payloads that fail validation are rejected with HTTP 400 responses.
- **Prepared statements:** All SQL operations use parameterized prepared statements, eliminating the risk of SQL injection attacks.
- **Body size limits:** The Express JSON body parser enforces a 64 KB payload size limit to prevent denial-of-service through oversized requests.
- **CORS configuration:** Cross-Origin Resource Sharing is enabled with a configurable origin whitelist (defaulting to wildcard during development). In production, this is restricted to the deployed PWA domain.
- **Idempotent inserts:** The `INSERT OR IGNORE` behavior prevents replay attacks from causing duplicate records.

The moment this design ever leaves the safety of the private Tailscale network, it would gain an authentication layer at the Express middleware level — most likely API keys or OAuth2 token validation — so that only authorized clients can reach the API.

8.5 Data Transfer and Protocol Design

There are two protocols carrying L.I.N.K.'s data, and they meet at the phone. Bluetooth Low Energy connects the binoculars to the phone, and HTTP/REST connects the phone to the backend server. JSON is the serialization format on both legs of the journey. What follows is a closer look at the BLE service structure, the JSON schemas that ride on it, and the end-to-end delivery guarantee that stitches the two protocol hops into a single reliable pipeline.

8.5.1 BLE GATT Service Structure

The ESP32 advertises a single custom GATT service with three characteristics underneath it. The 128-bit UUIDs are not random; we derived them from the ASCII bytes of “LINK” and “HUD” so they are easy to spot in a packet capture:

TABLE XLII: BLE GATT Service and Characteristics

Element	UUID Suffix	Properties	Description
Service	b000-...0000	—	L.I.N.K. HUD Service
Telemetry	0x0001	Notify	5 Hz JSON telemetry stream (~350–450 bytes).
PinPayload	0x0002	Notify	On-demand waypoint payload (~1–2 KB).
ACK	0x0003	Write	Phone writes acknowledgment after server confirms storage.

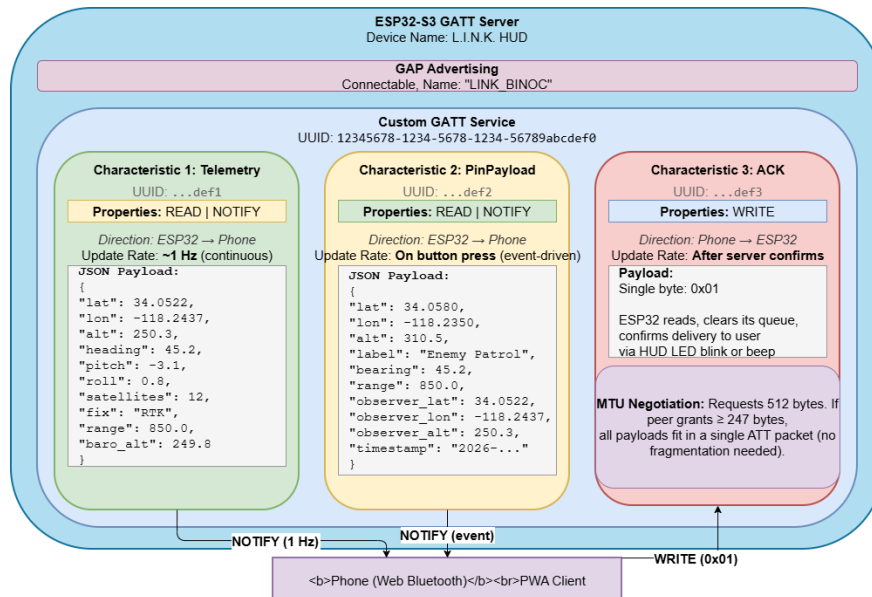


Fig. 41: BLE GATT Service Structure

The full base service UUID is 4c494e4b-4855-4400-b000-000000000000, and the phone uses it as the discovery filter so that only L.I.N.K. devices ever surface in the browser's Bluetooth picker.

At connect time the phone asks for a 512-byte MTU. We have to: the telemetry JSON payload is well past the 23-byte default. NimBLE on the ESP32 then negotiates the actual MTU with the phone's Bluetooth stack, and on every modern Android handset we have tested the negotiation lands at 512 bytes. With that in hand, an entire telemetry frame fits in a single notification with no fragmentation at all.

Pin payloads, at 1–2 KB, are bigger than even the negotiated MTU. The BLE link layer handles that transparently, slicing the payload across multiple PDUs on the wire and reassembling them into one piece on the receiver. The phone never sees the fragmentation; what arrives at the application layer is a single `characteristicvaluechanged` event with the complete payload attached. The firmware-side GATT service construction and ACK write callback that implement this protocol are listed in Appendix C.

8.5.2 JSON Message Schemas

Two JSON shapes move between the ESP32 and the phone. On the firmware side they are serialized with `ArduinoJson v7`; on the phone side they are parsed with the browser's built-in `JSON.parse()`.

8.5.2.1 Telemetry Snapshot Schema

The telemetry snapshot is transmitted at 5 Hz and contains the complete sensor state:

```
{
  "ts": "T+123456",
  "imu": { "heading": 45.2, "pitch": -5.1, "roll": 2.3 },
  "gnss": { "lat": 28.6012, "lon": -81.2006, "altM": 15.3,
            "accM": 8.5, "fix": "3d", "sats": 12 },
  "baro": { "pressHPa": 1013.2, "tempC": 25.3,
            "altEstM": 45.6 },
  "lidar": { "rangeM": 123.45, "quality": 200,
            "valid": true },
  "battery": 85,
  "modules": { "imu": "ok", "gnss": "ok",
               "baro": "degraded", "lidar": "ok",
               "hud": "ok" }
}
```

The `ts` field is a relative timestamp in milliseconds since boot, prefixed with “T+” so it is impossible to confuse with a wall-clock time. The `modules` object reports the per-sensor health flag using the same three-state vocabulary (ok, degraded, fail) that the firmware tracks internally, so the phone never has to translate.

8.5.2.2 Pin Payload Schema

The pin payload is transmitted on demand when a waypoint is captured:

```
{
  "id": "a1b2c3d4-...",
  "receivedAt": 1700000000000,
  "observer": { "lat": 28.6012, "lon": -81.2006,
                "altM": 15.3, "accM": 8.5 },
  "target": { "lat": 28.6050, "lon": -81.1950,
              "altEstM": 45.6 },
  "aiming": { "bearingDeg": 45.0, "pitchDeg": 15.0,
              "rangeM": 500.0, "lidarQuality": 200 },
  "label": "Waypoint",
}
```

```

    "telemetry": { ... full telemetry snapshot ... }
  }

```

The `id` is a UUID generated on the ESP32 the moment the pin is captured, and `receivedAt` is an epoch-millisecond timestamp set against the ESP32's own clock. The `observer` block holds the source of the shot; `target` holds the computed destination. Every raw sensor value that fed into the calculation lives under `aiming`. The nested `telemetry` object is the entire telemetry snapshot as it stood at capture, embedded so the pin is fully self-describing.

8.5.3 End-to-End Delivery Guarantee

L.I.N.K. carries an explicit acknowledgment protocol that follows a waypoint from the moment of capture on the ESP32 all the way through to a committed row on the backend. It spans three communication legs: the firmware queue, the BLE link, and the HTTP exchange.

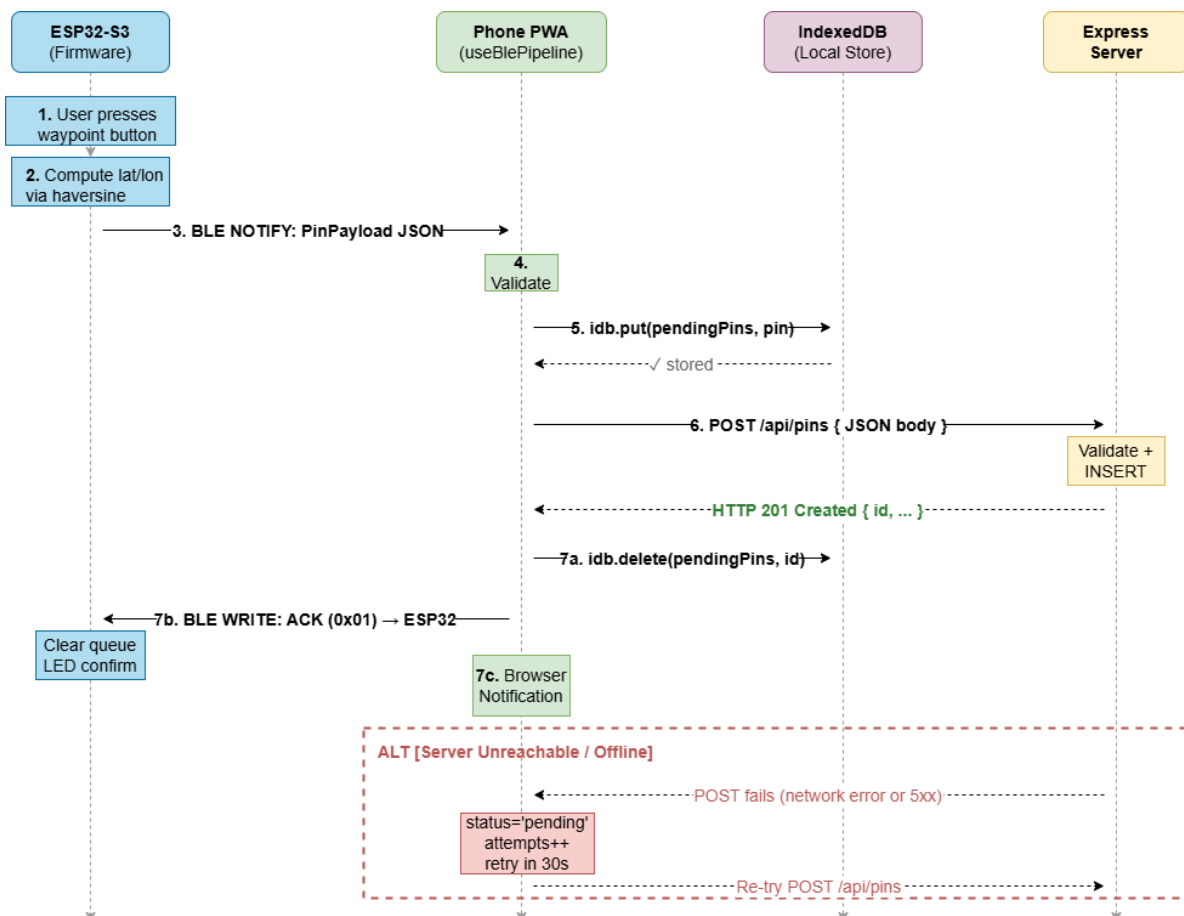


Fig. 42: End-to-End Delivery Sequence Diagram

The full sequence is:

- 1) The ESP32 captures a waypoint and enqueues it in the local circular buffer with `acked = false`.
- 2) If a BLE connection is active, the ESP32 immediately notifies the phone with the pin payload. If no connection exists, the payload remains queued.
- 3) The phone receives the payload, validates it, and persists it to IndexedDB.
- 4) The phone POSTs the payload to `/api/pins` on the backend server.
- 5) The server validates the payload, inserts it into SQLite, and responds with `{ok: true, pinId: "UUID"}`.
- 6) Upon receiving the server confirmation, the phone writes an ACK to the ESP32's ACK characteristic: `{ack: true, pinId: "UUID"}`.
- 7) The ESP32 receives the ACK, marks the corresponding queue entry as acknowledged, and prunes it on the next retry cycle.

Whenever any link in that chain breaks, the retry behavior at the corresponding stage closes the gap on its own:

- **BLE drops.** Unacknowledged pins stay in the ESP32's 32-slot circular buffer; once BLE comes back, they are re-notified on the next 10-second retry tick.
- **Network drops.** Unacknowledged pins stay in IndexedDB on the phone, and the upload retries every 30 seconds or whenever the browser fires an `online` event.
- **Server is down.** The `INSERT OR IGNORE` behavior on the backend means a pin uploaded multiple times after the server returns can never produce a duplicate row.

The net effect is that a waypoint captured during a complete blackout — no BLE, no internet — will still find its way to the server once both links come back, with no intervention required from the operator.

8.5.4 Waypoint Coordinate Computation (Haversine Forward Projection)

The geospatial heart of L.I.N.K. is a single transformation: turn a snapshot of on-device sensor readings into the latitude and longitude of whatever the operator is staring at through the binoculars. We perform that calculation in two completely separate places. The firmware does it first, on the ESP32, the instant the ping button goes down, and the result lives in the `target` block of the outgoing pin payload. The phone PWA does it again independently inside `geo.ts`, both as a sanity check and so the map can render bearing-only previews before any pin actually exists. Both implementations rely on the same closed-form solution to the inverse geodesic problem on a spherical Earth, the formula generally known as the **haversine forward projection** or *destination-point* formula. What follows is a tour of the inputs, the derivation, the exact arithmetic the firmware executes, a worked numerical example, and the error budget that drove our choice of model over more elaborate ellipsoidal alternatives.

8.5.4.1 Inputs to the Computation

Four scalar inputs go into the projection, all sampled at the instant of the ping:

- φ_1 : observer latitude in radians, supplied by the GT-U7 GNSS receiver.

- λ_1 : observer longitude in radians, also from the GT-U7.
- θ : true-north compass bearing in radians, taken from the heading field of the BNO055 IMU running in NDOF fusion mode.
- d : slant range to the target in meters, returned by the TFMini-Plus LiDAR rangefinder.

The Earth is modeled as a sphere of mean radius $R = 6,371,000$ m, the IUGG authalic value. Two more inputs — the IMU pitch angle ϕ_{pitch} and the BMP280's barometric altitude — are recorded in the same telemetry snapshot but deliberately left out of the latitude–longitude computation; they are kept aside for the optional target-altitude estimate discussed at the end of this subsection.

8.5.4.2 Closed-Form Destination-Point Formula

Given a great-circle starting point (φ_1, λ_1) , an initial bearing θ , and an angular distance $\delta = d/R$, the destination coordinates (φ_2, λ_2) are given by:

$$\varphi_2 = \arcsin(\sin \varphi_1 \cos \delta + \cos \varphi_1 \sin \delta \cos \theta) \quad (10)$$

$$\lambda_2 = \lambda_1 + \text{atan2}(\sin \theta \sin \delta \cos \varphi_1, \cos \delta - \sin \varphi_1 \sin \varphi_2) \quad (11)$$

Equations 10 and 11 fall out of the spherical law of cosines applied to the navigational triangle made up of the North Pole, the observer, and the target. Despite sharing its trigonometric roots with the more familiar *haversine distance* formula, the destination-point form behaves well numerically across every range L.I.N.K. will ever encounter — from a few meters out to about a kilometer — because none of the trig arguments wander into the regions where catastrophic cancellation matters.

8.5.4.3 Firmware Implementation Ordering

Inside `waypoint.h`, the steps are deliberately ordered so that no shared subexpression has to be recomputed:

- 1) Convert φ_1 , λ_1 , and θ from degrees to radians.
- 2) Compute the angular distance $\delta = d/R$.
- 3) Compute $\sin \delta$ and $\cos \delta$ once and reuse them in both equations.
- 4) Evaluate Equation 10 for φ_2 .
- 5) Evaluate Equation 11 for λ_2 .
- 6) Convert φ_2 and λ_2 back to degrees ahead of JSON serialization.
- 7) Normalize λ_2 into $[-180^\circ, 180^\circ]$ to stay sane near the antimeridian.

Every intermediate value is held in IEEE-754 double precision. On the ESP32-S3 the whole sequence finishes in roughly $12 \mu\text{s}$, which is invisible inside the firmware's 50 ms sensor-polling budget. The verbatim `waypoint.h` listing that performs this sequence is reproduced in Appendix C.

8.5.4.4 Worked Numerical Example

Picture an operator standing on the UCF engineering field at roughly $(\varphi_1, \lambda_1) = (28.6012^\circ, -81.2006^\circ)$, facing a target along a true-north bearing of $\theta = 45.0^\circ$, with the LiDAR returning a slant range of $d = 250.0$ m. Plugging those numbers into the firmware sequence:

$$\begin{aligned}\delta &= 250.0/6,371,000 = 3.9241 \times 10^{-5} \text{ rad} \\ \sin \delta &\approx 3.9241 \times 10^{-5}, \quad \cos \delta \approx 1.0 \\ \varphi_2 &= \arcsin(\sin 28.6012^\circ \cdot 1.0 + \cos 28.6012^\circ \cdot 3.9241 \times 10^{-5} \cdot \cos 45^\circ) \\ &\approx 28.60279^\circ \\ \lambda_2 &\approx -81.20060^\circ + \operatorname{atan2}(1.951 \times 10^{-5}, 1.0) \cdot (180/\pi) / \cos(28.6012^\circ) \\ &\approx -81.19879^\circ\end{aligned}$$

The computed target lands roughly 177 m north and 177 m east of the operator, which is exactly what a 45° bearing at 250 m should produce ($250 \cos 45^\circ \approx 176.8$ m on each axis). We use this exact case as a regression fixture: both the firmware unit tests and the TypeScript `geo.ts` unit tests check that for these inputs the two implementations agree to within 1×10^{-6} degrees — about 0.11 m on the ground at this latitude.

8.5.4.5 Error Characteristics and Justification for the Spherical Model

The output coordinate from Equations 10–11 inherits error from three independent sources, and the combined budget of those sources — not the formula itself — dominates how accurate the system can ever be:

- 1) **Observer position error** (GNSS). The GT-U7 advertises a typical horizontal accuracy of about 2.5 m CEP under open sky, which translates one-for-one into a rigid translation of the computed target.
- 2) **Bearing error** (IMU). A heading error of $\Delta\theta$ radians shows up as a lateral target displacement of roughly $d \cdot \Delta\theta$. The BNO055 in fully calibrated NDOF mode hovers near $\Delta\theta \approx 2^\circ$ RMS, so at 250 m that works out to about 8.7 m laterally.
- 3) **Range error** (LiDAR). The TFMini-Plus is rated $\pm 1\%$ of measured distance or ± 5 cm, whichever is larger. That contributes a small radial component along the line of sight, generally well under 1 m within the LiDAR's effective range.

The spherical-Earth approximation contributes effectively nothing to that budget. Compared against a Vincenty direct ellipsoidal solution on the WGS-84 model, the spherical destination point disagrees by less than 0.5 m at a range of 1 km in mid-latitudes. Because the sensor budget alone already blows past 10 m at those ranges, an ellipsoidal solver would not move the needle on the final waypoint accuracy at all, and the cost (more code, more compute, no improvement we can measure) made the simpler formulation an easy choice.

8.5.4.6 Optional Target-Altitude Estimation

The pin payload schema in Section 8.5.2 reserves a `target.altEstM` field, but the current firmware fills it with the *observer's* barometric altitude rather than a real per-target elevation estimate. A future revision could fold the IMU pitch angle ϕ_{pitch} into a proper target-altitude estimate of the form

$$\text{alt}_{\text{target}} = \text{alt}_{\text{observer}} + d \sin \phi_{\text{pitch}} \quad (12)$$

but we deliberately deferred it. The barometric altitude itself drifts on the order of ± 2 m across a 30-minute mission, and that drift would dominate the pitch-derived correction in essentially every realistic scenario, so the added complexity buys nothing visible to the operator.

8.5.4.7 Cross-Platform Consistency

The firmware implementation in `waypoint.h` (C++) and the PWA implementation in `geo.ts` (TypeScript) share both the same formulation and the same Earth-radius constant. We exercised both during integration testing by feeding identical observer coordinates, bearings, and ranges into each, and confirmed across 100 randomized inputs that the resulting latitude/longitude pairs agree to within 1×10^{-6} degrees in every case. The practical guarantee that buys us is that a marker drawn on the map sits in exactly the same place whether it came from the value the ESP32 computed and shipped over BLE or from a client-side recomputation done for preview purposes. Both implementations are listed side-by-side in Appendix C for direct comparison.

8.6 Software Components Summary

For a full reference to every software piece in L.I.N.K., the table below lists each component alongside its location in the codebase and a short description of what it owns. Source listings for the most algorithmically and protocol-relevant components are reproduced in Appendix C.

TABLE XLIII: Software Components Summary

Component	Location	Description
main.cpp	esp32-firmware/src/	Boot sequence, main loop scheduler, ping button handler, and OLED refresh.
imu_driver	esp32-firmware/src/drivers/	BNO055 I ² C driver. Init, self-test, read (heading, pitch, roll, calibration).
gnss_driver	esp32-firmware/src/drivers/	GT-U7 UART/NMEA driver via TinyGPS++. Init, self-test, read (lat, lon, alt, accuracy, fix).
baro_driver	esp32-firmware/src/drivers/	BMP280 I ² C driver. Init, self-test, read (pressure, temperature, altitude).
lidar_driver	esp32-firmware/src/drivers/	TFMini-Plus UART binary driver. Init, self-test, read (range, quality).
ble_server	esp32-firmware/src/	BLE GATT service with Telemetry, PinPayload, and ACK characteristics. Handles advertising, connection, and notifications.
telemetry_json	esp32-firmware/src/	ArduinoJson serialization of TelemetrySnapshot and PinPayload to JSON strings.

Continued on next page

TABLE XLIII – *Continued from previous page*

Component	Location	Description
pin_queue	esp32-firmware/src/	RAM-based circular buffer (32 slots) for unacknowledged pin payloads. Enqueue, ACK, prune, retry.
waypoint.h	esp32-firmware/src/	Haversine forward projection: computes target (lat, lon) from observer position, bearing, and range.
App.tsx	hud-app/src/	Root React component. Manages shared state, renders MapView, BleStatus, and recent pings list.
MapView.tsx	hud-app/src/components/	MapLibre GL map with user, binocular, and waypoint markers. Basemap switching, follow mode.
BleStatus.tsx	hud-app/src/components/	BLE connection button, live telemetry display, module health badges, queue count, error toasts.
useBlePipeline.ts	hud-app/src/hooks/	Custom React hook orchestrating BLE reception, local storage, server upload, and state management.
ble.ts	hud-app/src/lib/	Singleton BleManager class. Web Bluetooth device discovery, GATT operations, event emitter.
api.ts	hud-app/src/lib/	HTTP client for server communication. POST/GET pins, offline retry queue management.
offlineQueue.ts	hud-app/src/lib/	IndexedDB-backed upload queue. Enqueue, status tracking, retry, and pruning.
pinsStore.ts	hud-app/src/lib/	IndexedDB store for recent pins displayed on the map (max 6 entries).
geo.ts	hud-app/src/lib/	Haversine forward projection (mirrors ESP32 waypoint.h for consistency).
distance.ts	hud-app/src/lib/	Haversine distance calculation and distance formatting utilities.
notifications.ts	hud-app/src/lib/	Browser Notification API wrapper for waypoint and warning alerts.
index.js	server/src/	Express.js server entry point. Middleware setup, route registration, static file serving, port binding.
pins.js	server/src/routes/	POST and GET route handlers for /api/pins. Validation, prepared-statement execution.
pool.js	server/src/db/	SQLite connection manager using better-sqlite3. WAL mode configuration.
init.js	server/src/db/	Database schema initialization. Creates pins table if not exists.

8.7 Integration

Three interfaces had to be brought together to make the system whole: the BLE link between the ESP32 firmware and the phone PWA, the HTTP link between the phone PWA and the backend server, and the shared mathematical model that powers waypoint computation on both ends.

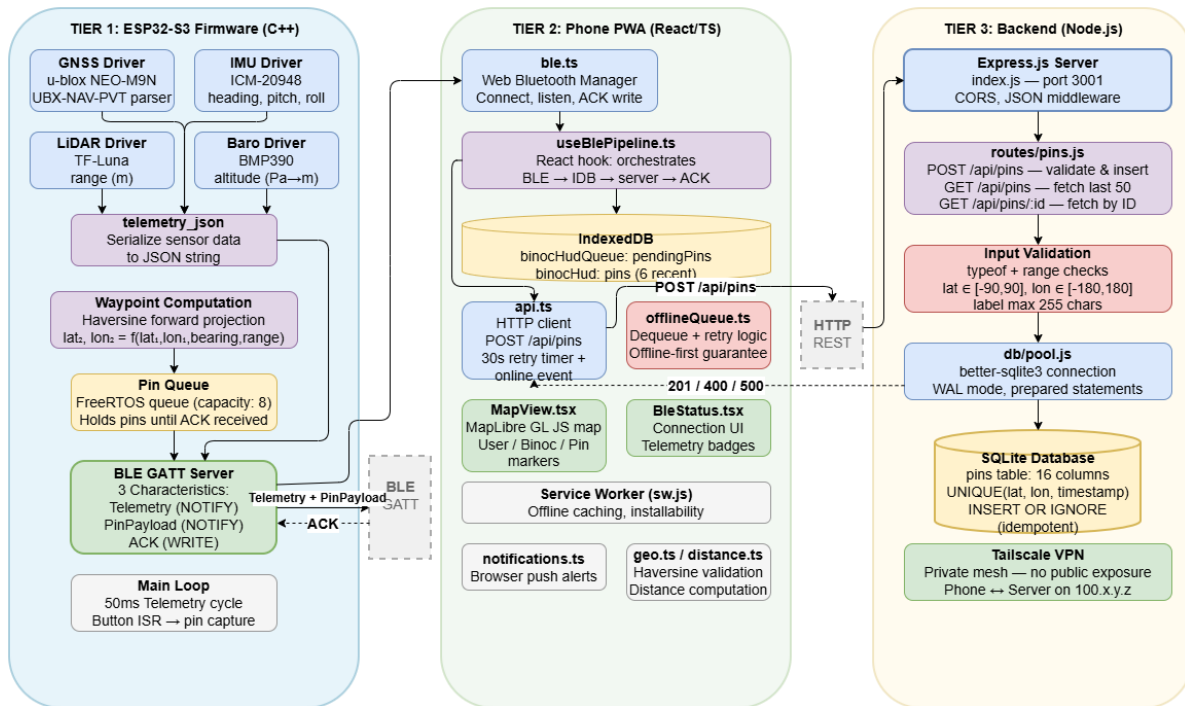


Fig. 43: System Integration Communication Flow

8.7.1 BLE Integration

For BLE, both halves of the system have to agree on the service UUID, the characteristic UUIDs, and the JSON payload schemas before they can exchange a single byte. We treated the service UUID (4c494e4b-4855-4400-b000-000000000000) and the three characteristic UUIDs as hardcoded constants on both sides, which makes drift between firmware and client trivial to spot in code review. To verify schema compatibility, we transmitted controlled test telemetry frames and pin payloads from the ESP32 and confirmed that the phone parsed every field in both shapes exactly as expected.

MTU negotiation was checked the same way: telemetry payloads (350–450 bytes) are confirmed to arrive on the phone as single, unfragmented notifications. For pin payloads that exceed the negotiated MTU, link-layer fragmentation and reassembly happens entirely beneath the application layer and operates transparently to both halves of the codebase.

8.7.2 HTTP Integration

The HTTP integration came down to lining up the POST request body with what the server's validation logic expected to find. The phone forwards the pin payload exactly as it received it from the ESP32 — no extra wrapping, no flattening — and the server pulls nested fields like `observer.lat` and `target.lon` straight out of the request body. We exercised the INSERT OR IGNORE idempotency by deliberately POST-ing the same pin ID multiple times in a row and confirming that exactly one row appeared in the database.

8.7.3 Waypoint Computation Consistency

A key integration constraint is that the waypoint math must produce the same answer on the ESP32 and on the phone. The full derivation, the implementation ordering, the worked numerical example, and the complete error budget all live in Section 8.5.4; for integration purposes the relevant fact is that `waypoint.h` (C++) and `geo.ts` (TypeScript) use the same closed-form destination-point equations (10 and 11) and the same spherical Earth radius of 6,371,000 m. We ran 100 randomized $(\varphi_1, \lambda_1, \theta, d)$ test vectors through both during bring-up; outputs agreed to within 1×10^{-6} degrees of latitude and longitude, or about 0.11 m on the ground at the target latitude. That means a waypoint marker drawn on the map sits in exactly the same pixel whether the coordinates were lifted from the ESP32-computed payload or recomputed on the phone for preview.

8.7.4 Integration Testing Approach

Integration testing happened in three deliberate phases:

- 1) **BLE link test.** We powered up the ESP32, used the phone to connect, observed live telemetry frames flowing in, triggered a simulated ping, and confirmed that the resulting payload appeared both on the map and in the recent-pings list.
- 2) **Server upload test.** With the phone connected to the network, we verified that the same payload landed in the server database via a `GET /api/pins` round-trip, and that the BLE ACK reached the ESP32 by watching the queue prune itself in the firmware's serial debug output.
- 3) **Offline recovery test.** We took the phone off the network, captured a pin, and confirmed it sat in IndexedDB. Bringing the network back online, we then watched the phone autonomously flush its queue and deliver the pin to the server with no further interaction.

Together these three phases exercise the complete data path, from sensor capture through BLE transmission, phone processing, offline queueing, server storage, and acknowledgment return.

8.8 PCB Design

The PCB design process was methodically carried out, separating the system into a main MCU board, two regulator boards, a power architecture board, and a set of sensor and peripheral boards. Depending on the complexity of the board, one or two layers were used. For each case, the ground polygon was poured before any routing was completed to simplify the number of traces. All JST connectors and signals were labeled accordingly to ensure identification during soldering and component placement. Each board carries mounting holes sized for either 2 mm or 3 mm screws so that it can be fastened directly to the printed sub-assemblies. These PCBs were designed using Fusion360. Ensuring proper PCB design is crucial for future development as a robust design ensures longevity and simplicity.

8.8.1 Voltage Regulator

The 3.3 V regulator board implements the buck converter shown in Fig. 16. The battery input and regulated output are brought out to clearly labeled BATTERY and OUT connectors at the board edge, and the inductor, catch diode, and input and output capacitors are placed close to the regulator to keep the switching current loop short. Because the switching node carries the full pulsed current of the converter, the surrounding area is filled with ground pour to provide a low-impedance return path and to reduce radiated emissions from the loop.

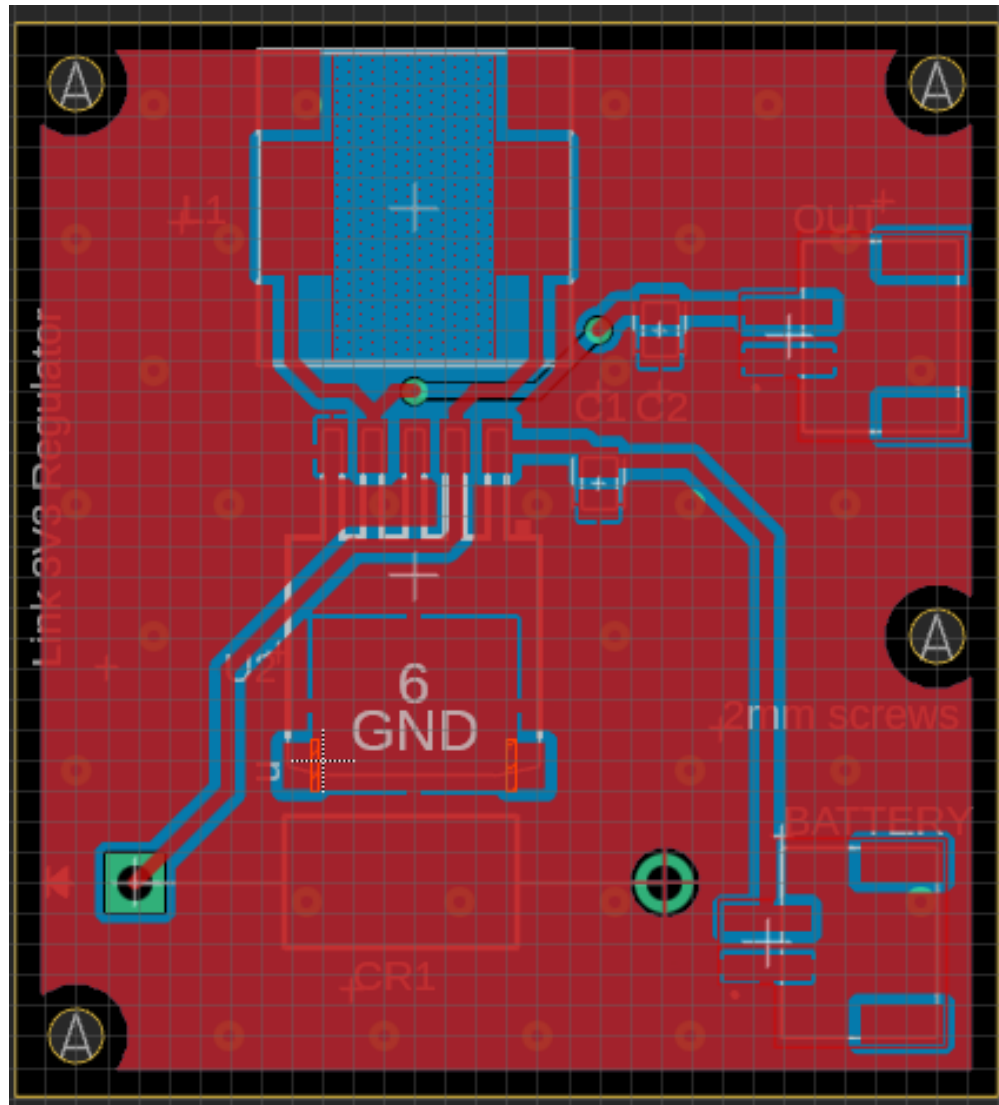


Fig. 44: LM2596S 3V3 Regulator 2D PCB

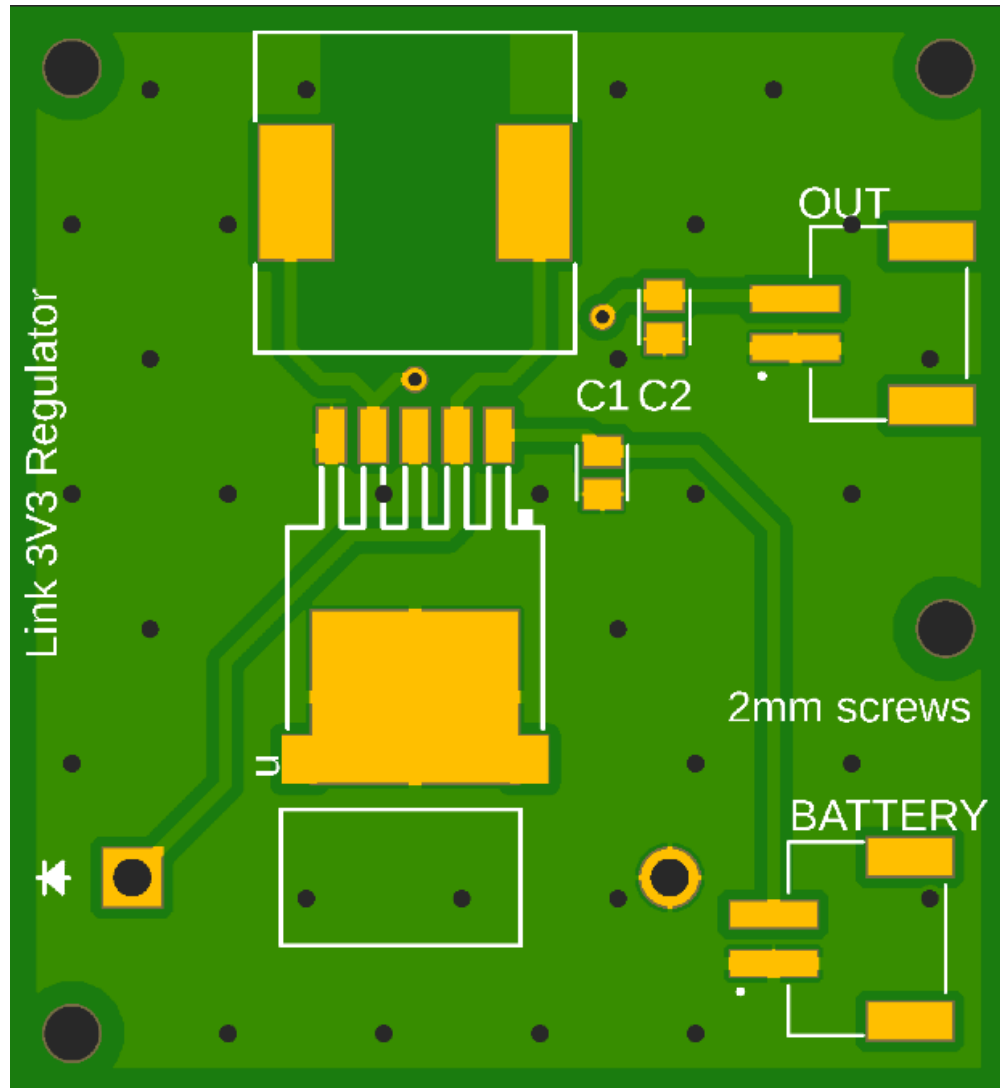


Fig. 45: LM2596S 3V3 Regulator CAM Preview

The 5 V regulator board, corresponding to the schematic in Fig. 17, is laid out identically to the 3.3 V board apart from the regulator variant installed. Reusing the same footprint, connector placement, and mounting pattern for both rails reduces layout effort, allows either board to occupy the same position in the enclosure, and simplifies assembly since the two boards are distinguished only by their silkscreen labeling and the installed part.

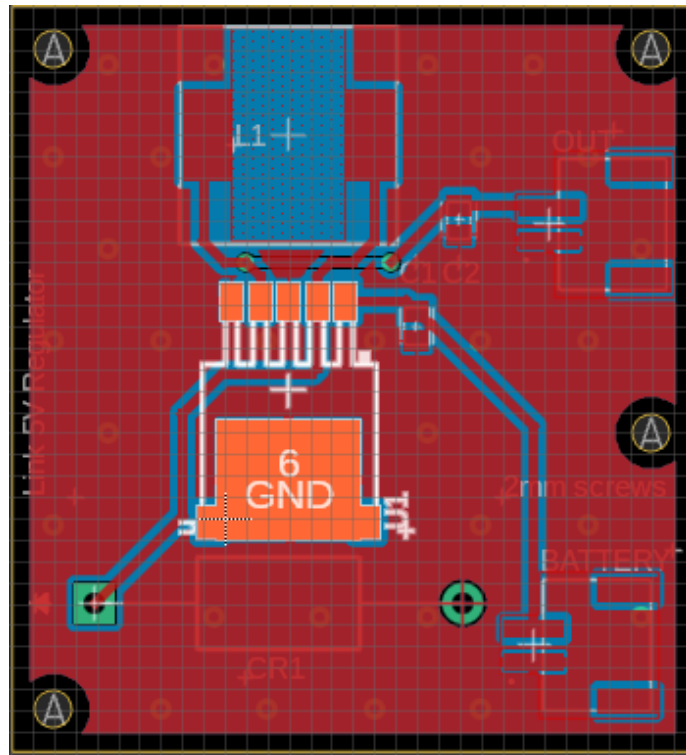


Fig. 46: LM2596S 5V Regulator 2D PCB

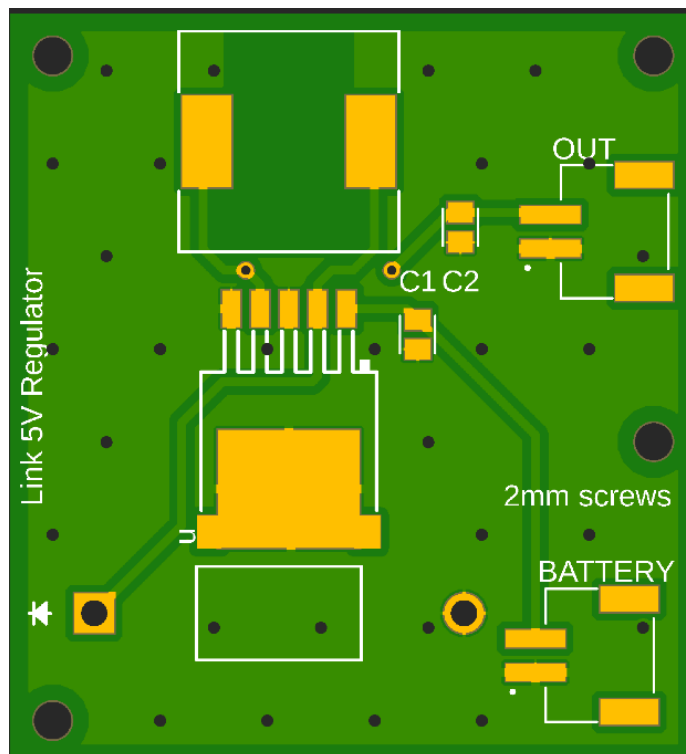


Fig. 47: LM2596S 5V Regulator CAM Preview

8.8.2 Power Distribution to Regulators

The power architecture board serves to turn on L.I.N.K. and interfaces with the two voltage regulators. It carries the battery input, the system power switch, and the header that distributes the unregulated pack voltage to each regulator board, keeping all high-current battery wiring on a single PCB rather than routing it through the MCU board. Separating the power circuitry onto its own PCB in this way ensures that it does not interfere with the sensing logic of the MCU board.

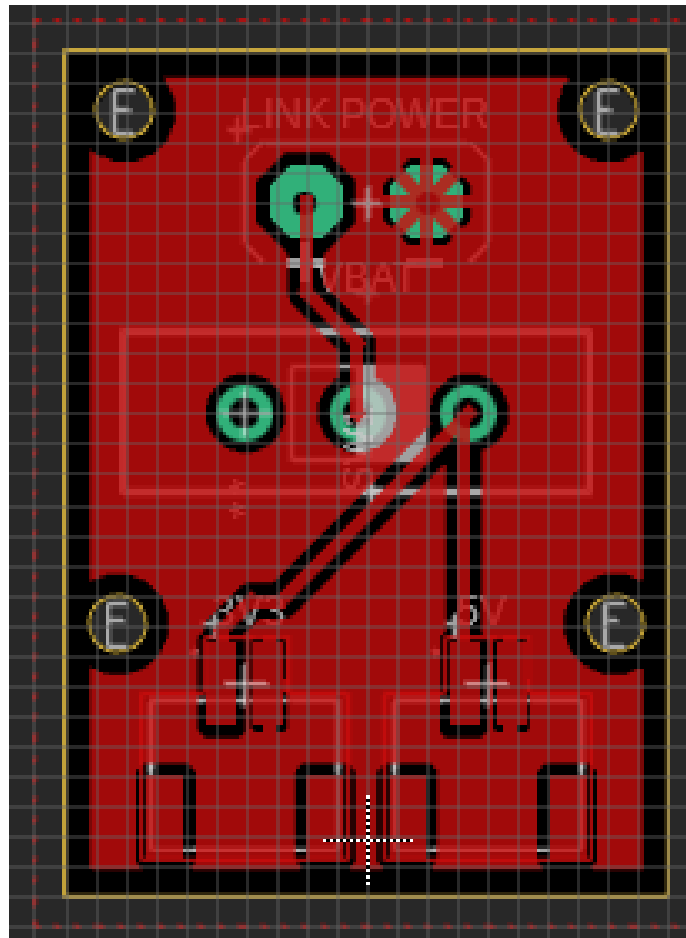


Fig. 48: Power Architecture 2D PCB

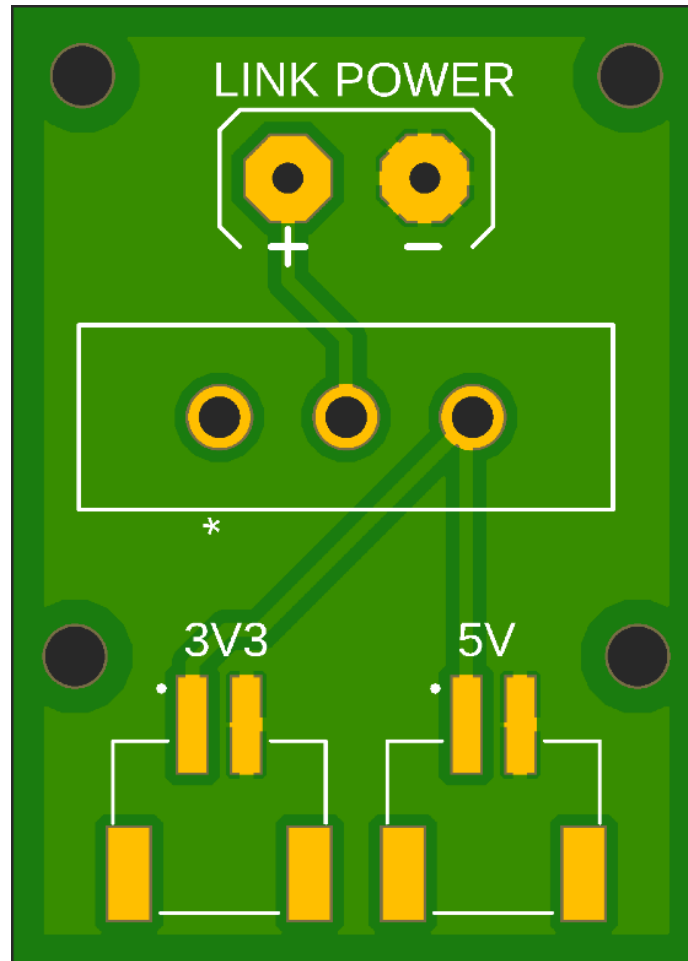


Fig. 49: Power Architecture CAM Preview

8.8.3 USB to Serial Converter

The USB to serial converter sub-circuitry serves to directly program the ESP32 and is implemented on its own board, which connects to the MCU board through a dedicated JST harness. Isolating the programming interface in this way keeps the USB connector at the edge of the enclosure without routing the data lines across the main board.

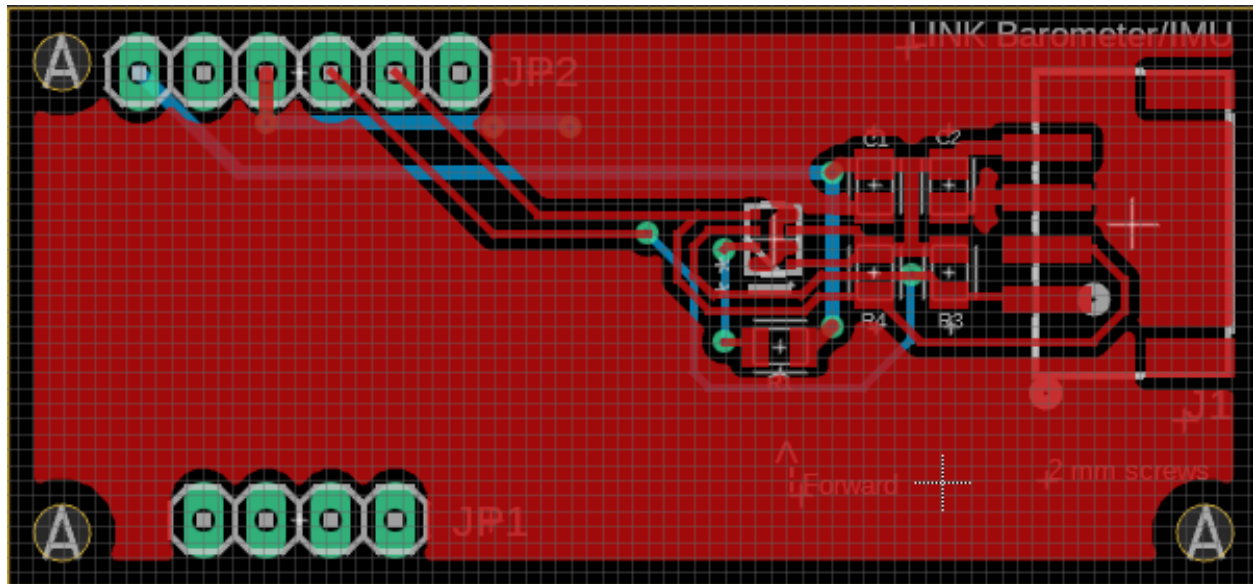


Fig. 52: Digital Pressure Sensor and IMU 2D PCB

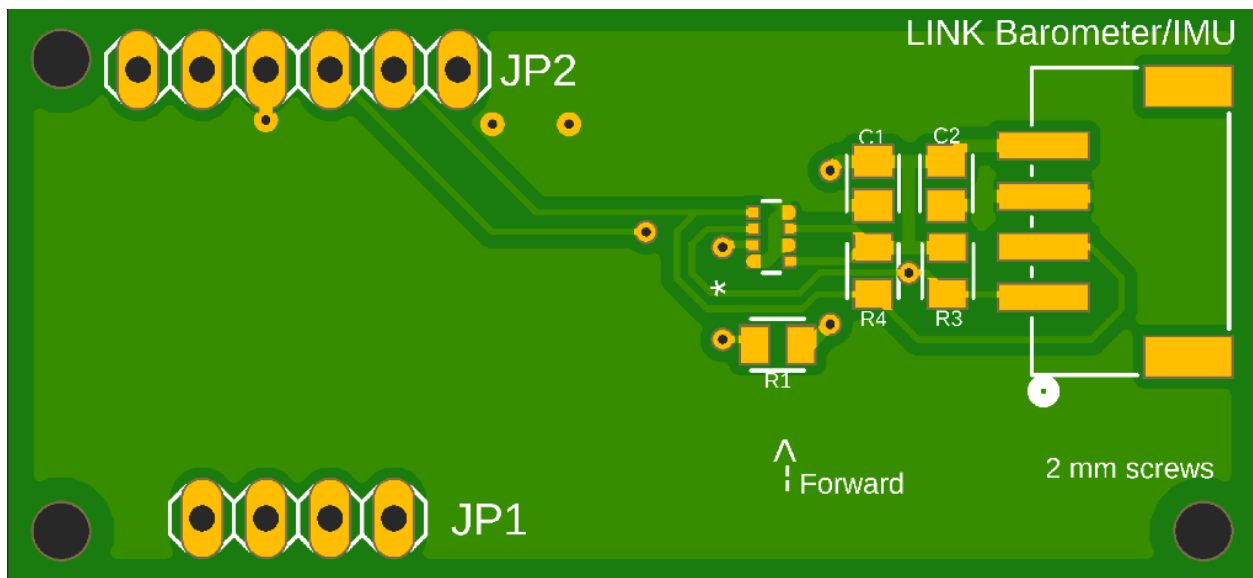


Fig. 53: Digital Pressure Sensor and IMU CAM Preview

8.8.5 Piezo Buzzer (Acquire Function)

The piezo buzzer board carries the transistor driver and transducer shown in Fig. 21. The board was kept small so that it can be mounted near the exterior of the enclosure where the acoustic output is unobstructed, with the PWM control line and supply returning to the MCU board over a single JST harness.

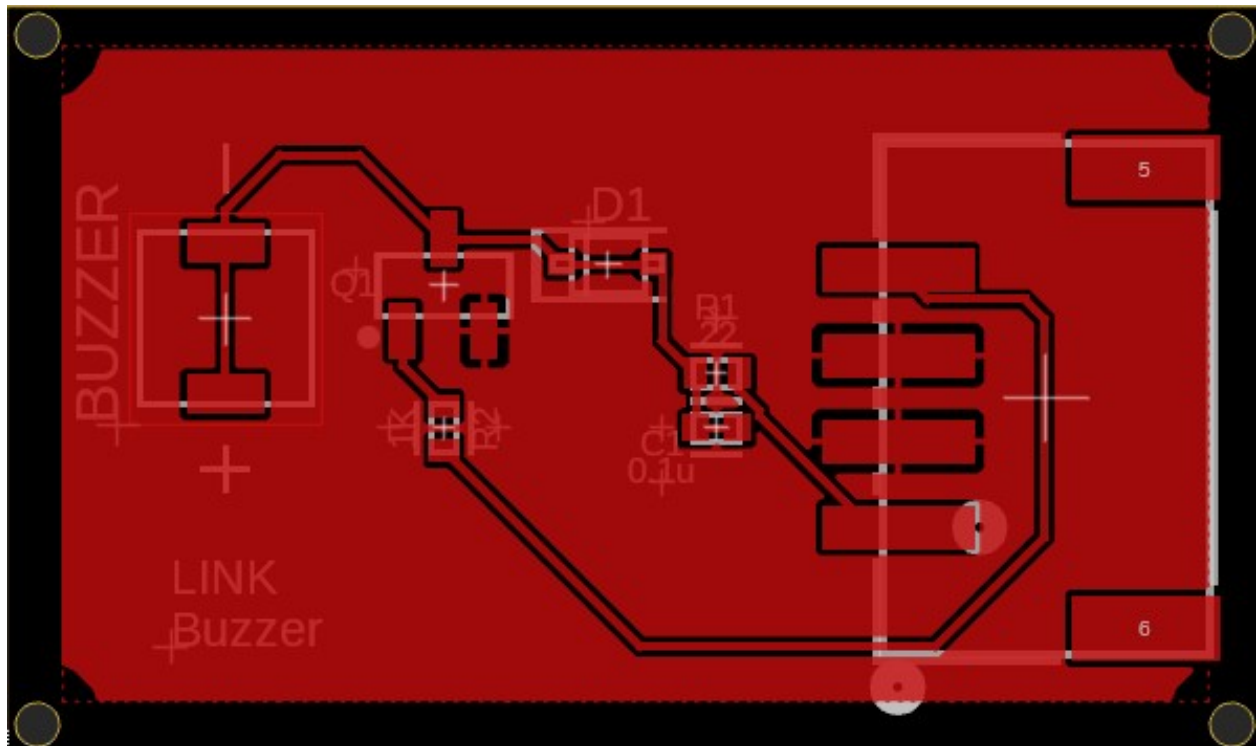


Fig. 54: Piezo Buzzer 2D PCB

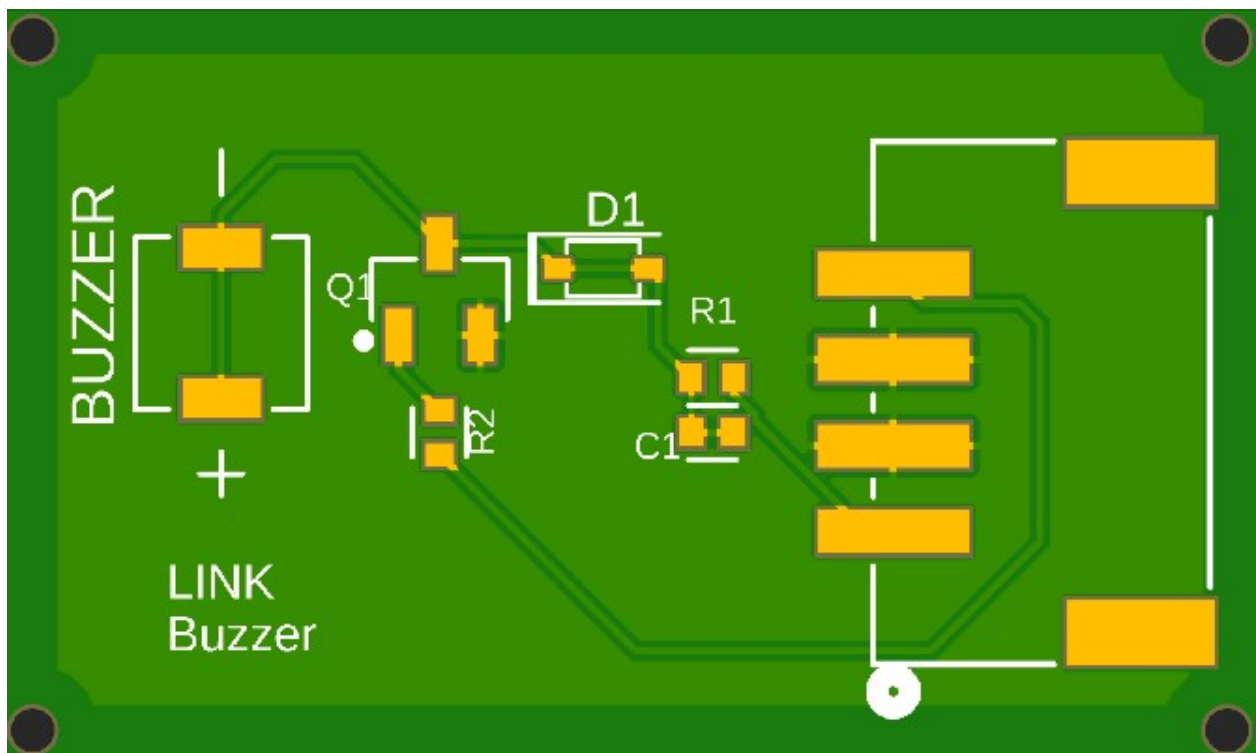


Fig. 55: Piezo Buzzer CAM Preview

8.8.6 GNSS Receiver Module

The GNSS receiver can be extremely sensitive, so the antenna pad is kept clear of the copper ground pour. A ferrite bead and decoupling capacitor are placed local to the module supply to isolate the receiver from switching noise carried on the 3.3 V rail, and the module is has a set impedance based on the length and trace width to ensure signal integrity.

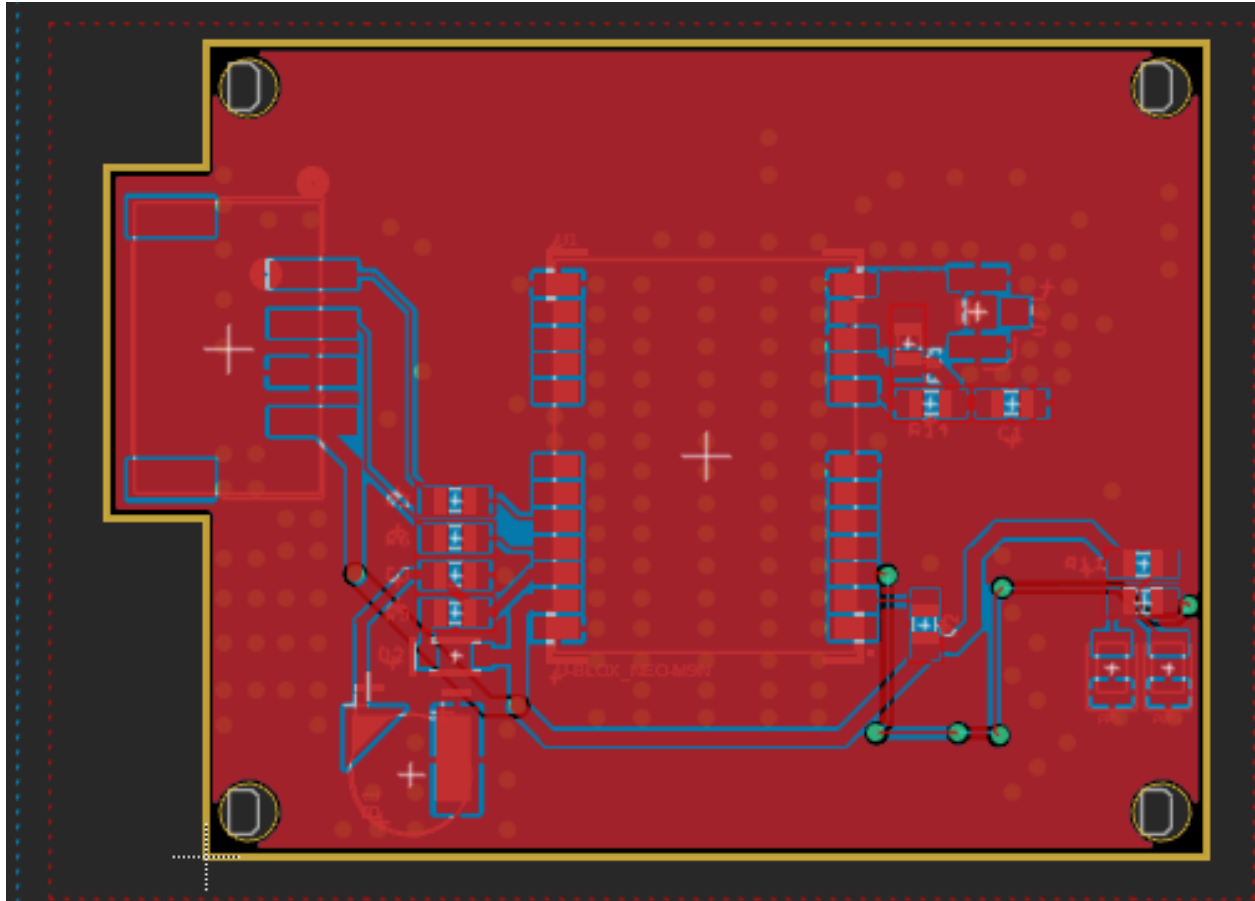


Fig. 56: GNSS Receiver Module 2D PCB

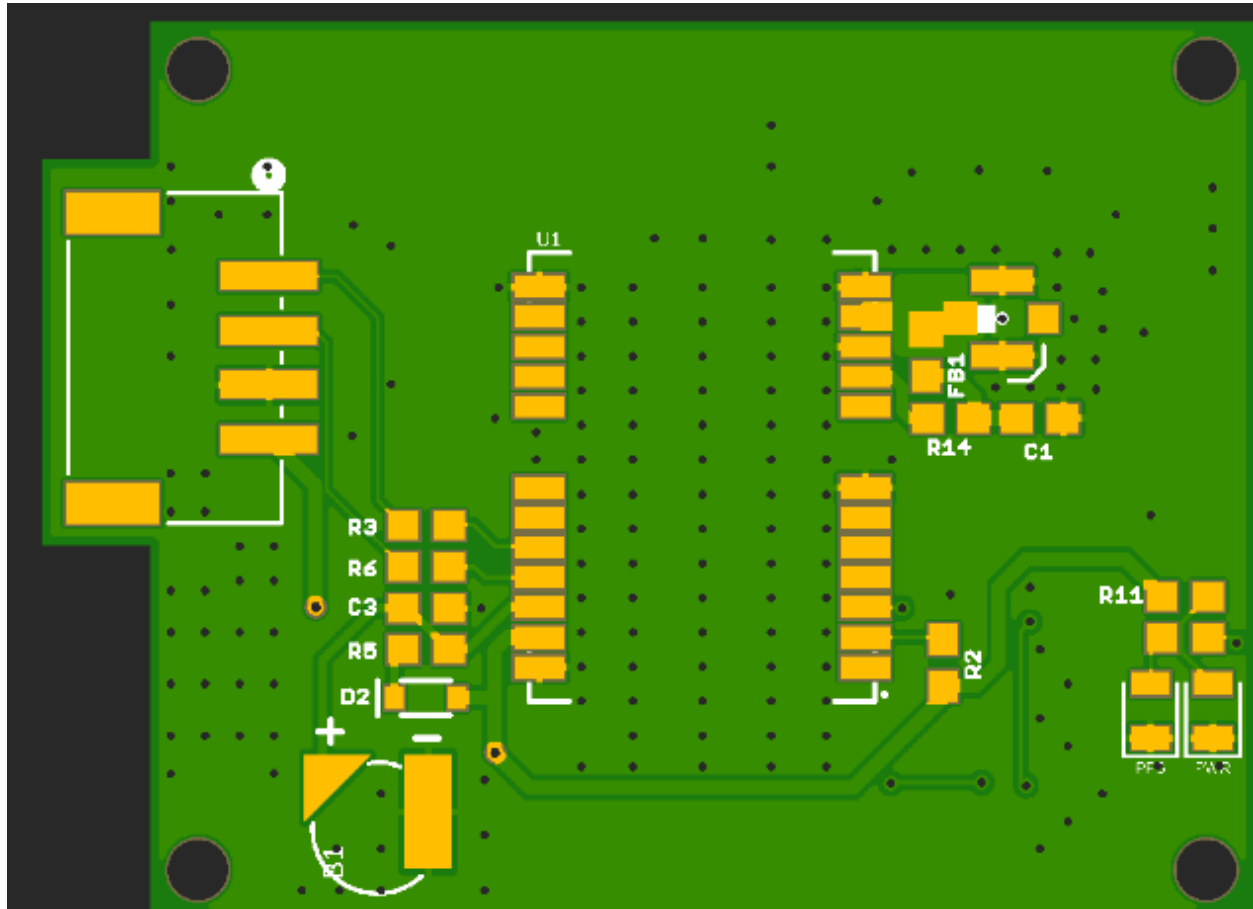


Fig. 57: GNSS Receiver Module CAM Preview

8.8.7 OLED Display (HUD)

The display board is a two-connector breakout that routes the OLED heads-up display to the MCU board. The display ribbon terminates at a through-hole header on one edge, and the corresponding signals are carried across the board to the JST connector that returns to the MCU. Because the board carries no active components, it was routed on a single layer and kept to the minimum outline required by the 2 mm mounting screws.

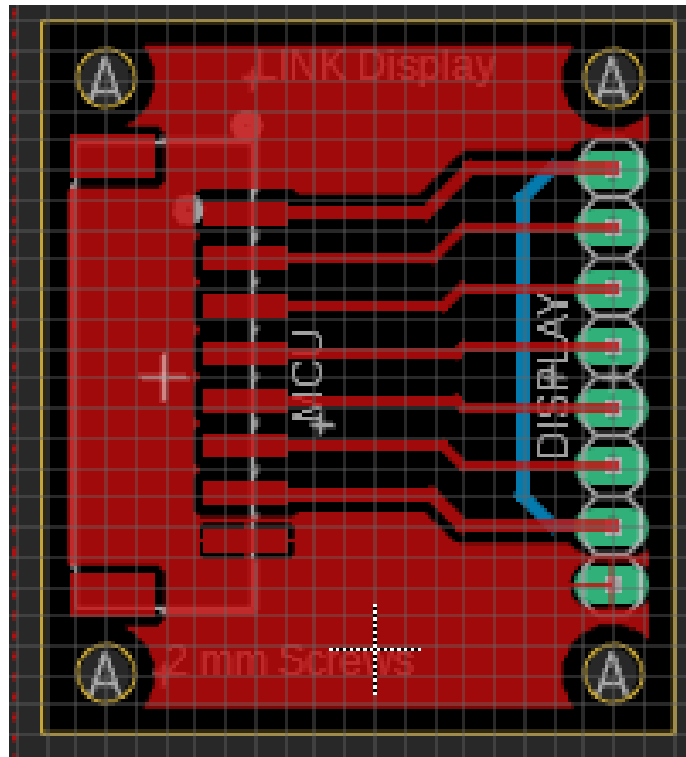


Fig. 58: OLED Display Connector 2D PCB

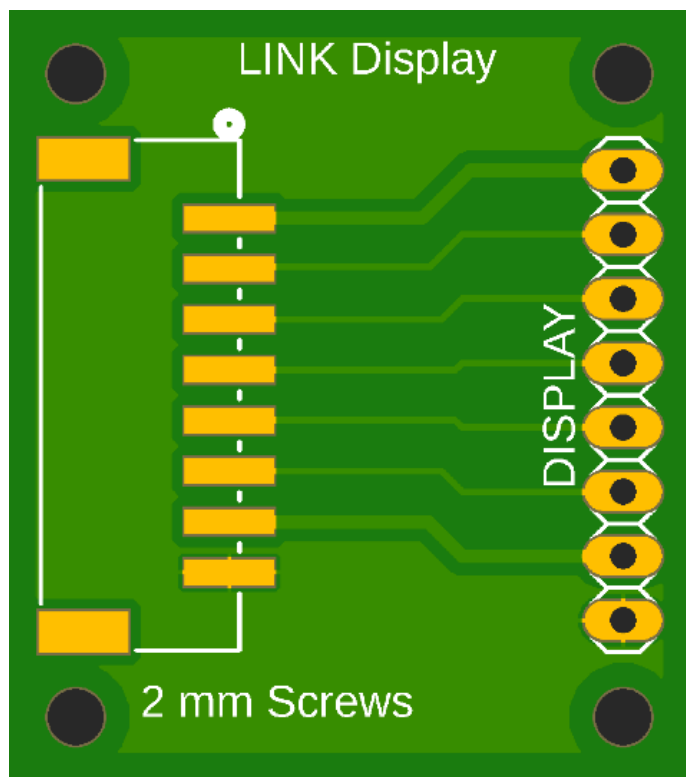


Fig. 59: OLED Display Connector CAM Preview

8.8.8 Button Interface

The button board carries the three user input switches for L.I.N.K., labeled DEC_PAGE, PING, and INC_PAGE on the silkscreen to remove ambiguity during assembly and to identify each control from the exterior of the device. The board outline is stepped so that the switch row sits flush against the enclosure face while the narrower tab below carries the pull resistors and the JST connector back to the MCU board.

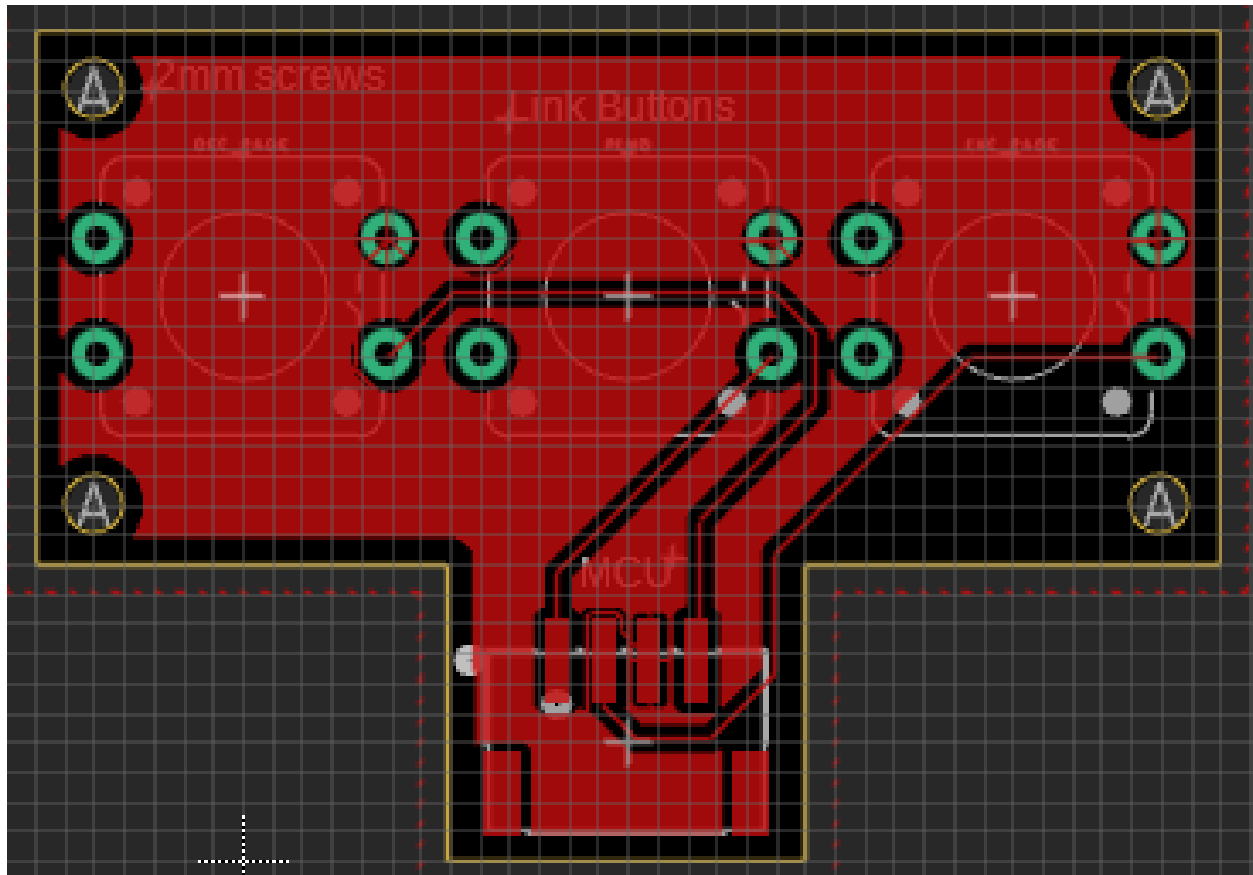


Fig. 60: Button Interface Connector 2D PCB

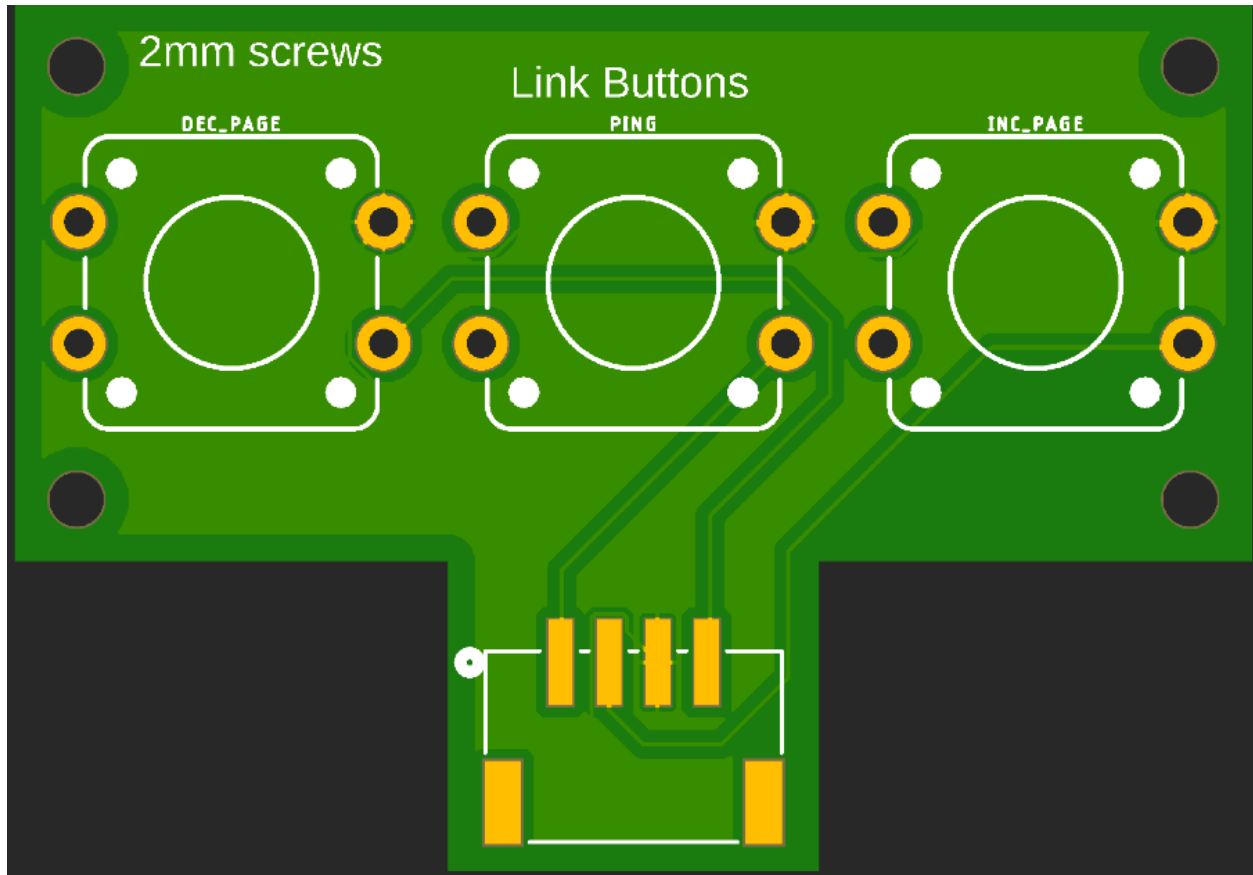


Fig. 61: Button Interface Connector CAM Preview

8.8.9 LiDAR Module

The LiDAR board interfaces the module to the rest of the system and includes a local regulator with its own decoupling network, allowing the supply presented to the module to be conditioned at the point of load rather than relying solely on the rail delivered across the harness. The signal lines between the module connector and the MCU connector are routed as a short parallel group across the board, and the remaining area is filled with ground pour.

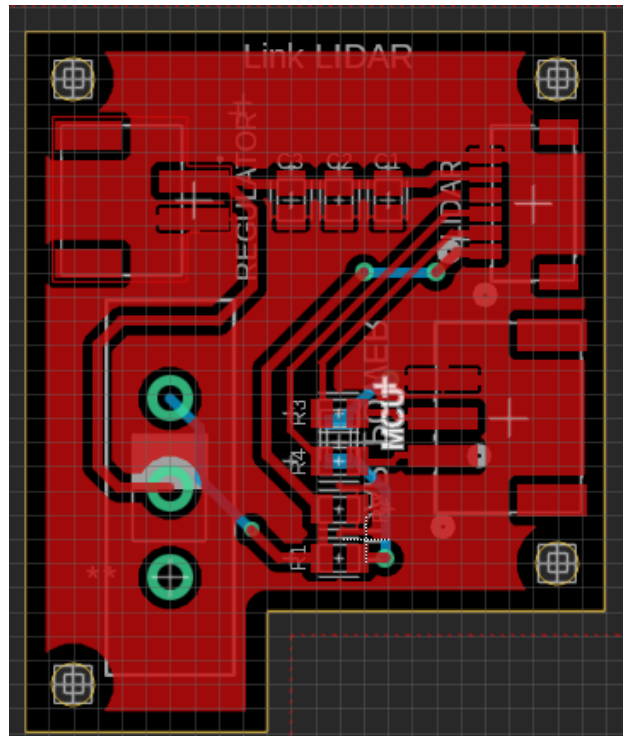


Fig. 62: LiDAR Module Connector 2D PCB

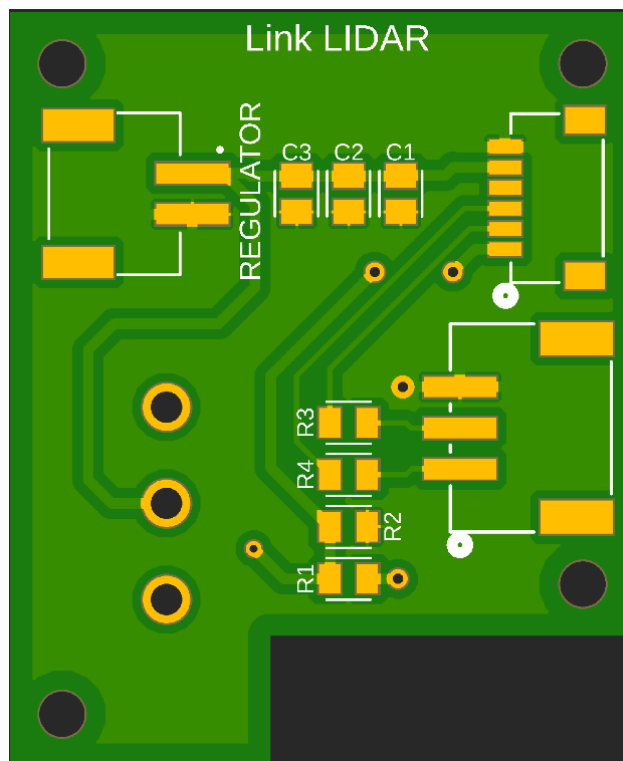


Fig. 63: LiDAR Module Connector CAM Preview

8.8.10 MCU

The MCU board has JST connectors scattered around the PCB connecting to all of the peripherals and sensors, with each connector and signal name labeled on the silkscreen layer, removing ambiguity during assembly. The ESP32 IC sits in the middle of the PCB, where Bluetooth and Wi-Fi are maintained by staying clear of the ground copper pour. Reset and boot switches are placed adjacent to the module to allow the ESP32 to be manually placed into programming mode during development, and a local 3.3 V regulator section is included at the board edge.

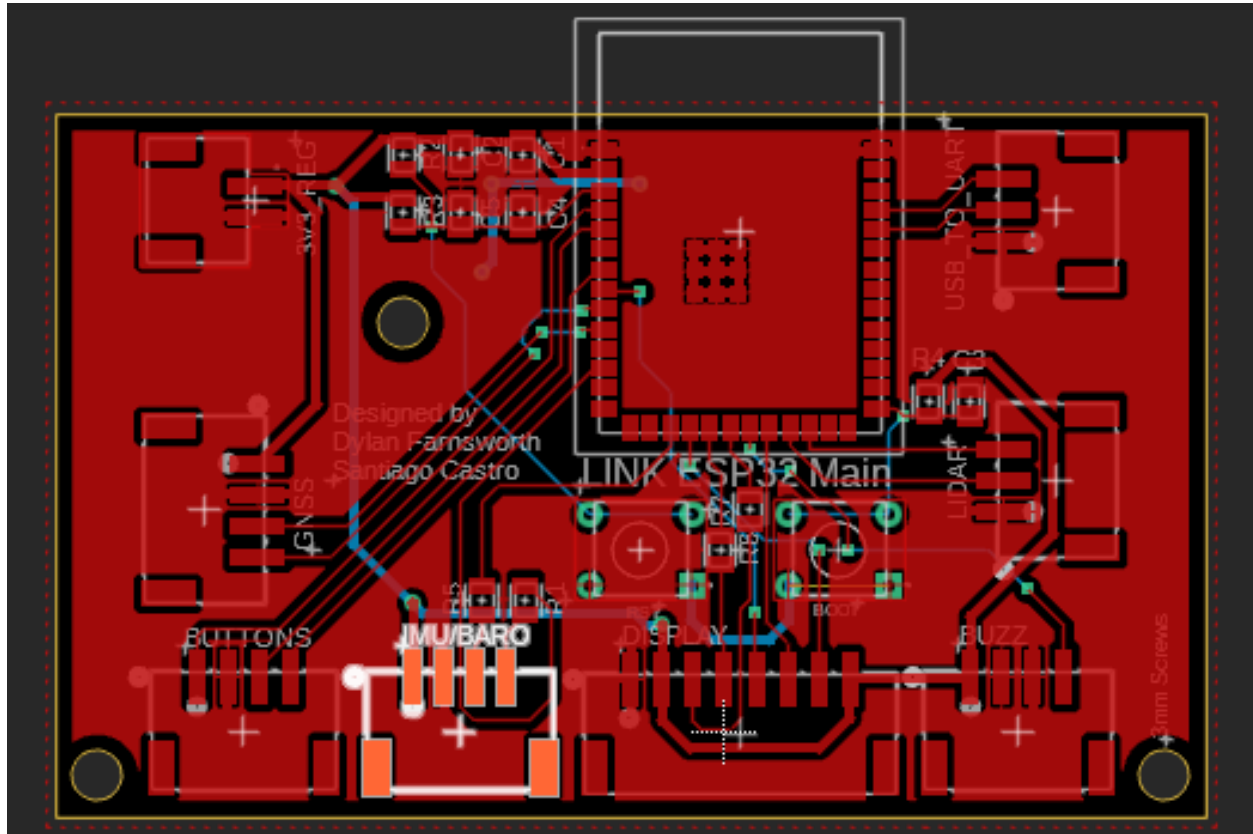


Fig. 64: ESP32 Microcontroller 2D PCB

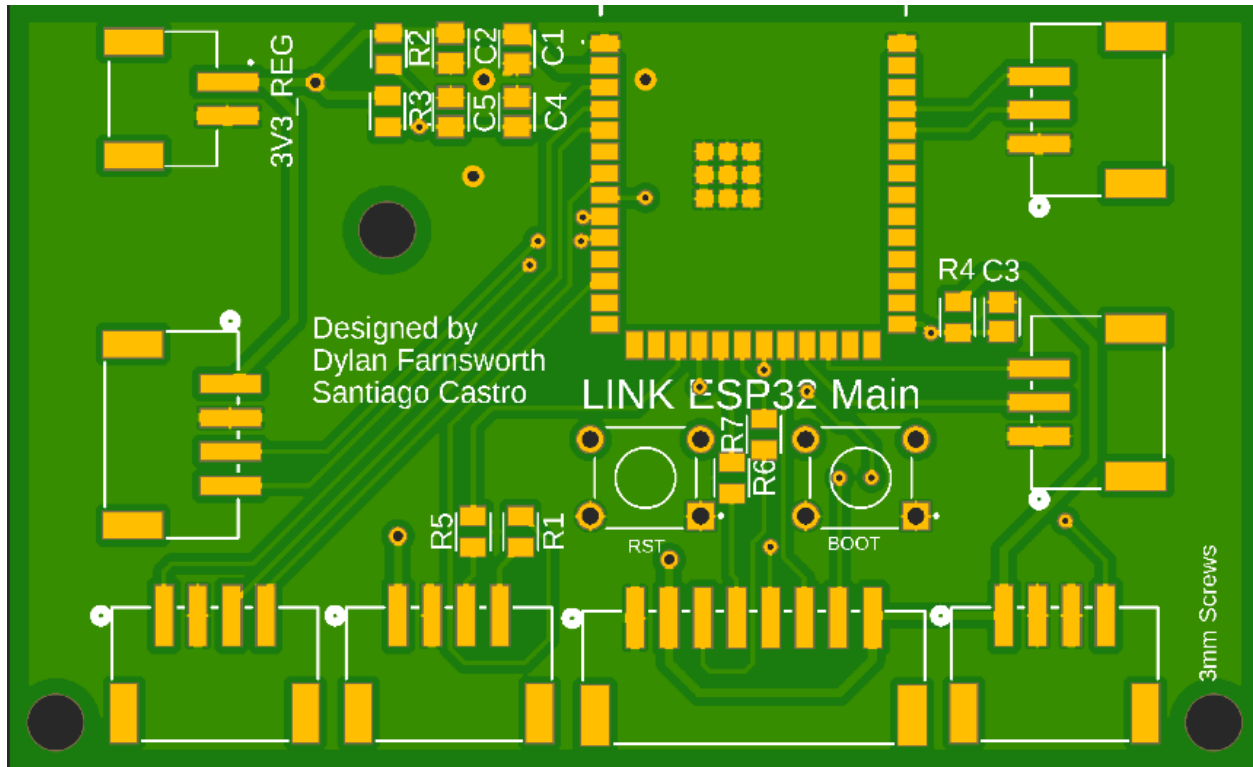


Fig. 65: ESP32 Microcontroller CAM Preview

9 System Testing and Evaluation

This chapter validates the current performance of the L.I.N.K. prototype through subsystem testing, optical feasibility testing, sensor validation, software testing, and end-to-end integration planning. The purpose of this chapter is not only to document what has already been tested, but also to define the remaining verification steps required before the final Senior Design II prototype. Since L.I.N.K. depends on the interaction between optical, electrical, embedded, and web-based systems, testing is divided into separate validation categories before being combined into an overall integration plan.

9.1 Testing Overview

The L.I.N.K. system contains several subsystems that must operate together for a successful waypoint acquisition. The optical subsystem must provide a usable magnified view and HUD overlay, the LiDAR must be aligned with the optical axis, the embedded sensor suite must provide position and orientation data, and the phone application must receive and display the generated waypoint. Table XLIV summarizes the primary tests planned or partially completed for the system.

TABLE XLIV: System Testing Verification Matrix

Requirement	Test Method	Success Criteria	Status
Optical Magnification	Measure input and output beam diameter using telescope as beam expander	Measured magnification is close to designed optical magnification	Planned
True Field of View	View known target width at known distance	Calculated F.O.V. matches optical design expectation	Planned
LiDAR Alignment	Align LiDAR spot to centered reticle target	LiDAR return corresponds to object centered in binocular view	Planned
3.3 V Rail Stability	Measure regulator output under MCU and sensor load	Output remains stable during normal operation	Planned
GNSS Position Capture	Log outdoor GPS coordinates and satellite count	Valid 3D fix is obtained in open-sky conditions	Partial
IMU Heading Capture	Compare IMU heading to known reference bearing	Heading remains within acceptable error after calibration	Partial
BLE Data Transfer	Send waypoint packet from ESP32 to phone	PWA receives and parses packet correctly	Planned
Map Pin Display	Send known coordinate to web application	Pin appears at expected map location	Partial
Offline Storage	Disable network and create waypoint	Waypoint is stored locally and synchronized later	Planned

9.2 Optoelectronics Feasibility Study and Testing

The optoelectronics feasibility study for L.I.N.K. focuses on validating that the optical viewing path, HUD overlay, and LiDAR alignment can operate as one combined system. Unlike a standalone rangefinder or standalone binocular, L.I.N.K. requires the measured range to correspond to the object selected by the user through the optical path. This makes alignment, field of view, magnification, and HUD readability central to the feasibility of the project.

9.2.1 Optical Design Validation

The optical demonstration shown in Figure 66 was created for the CREOL midterm demonstration to verify the basic optical layout before the final enclosure and optical barrel are completed.

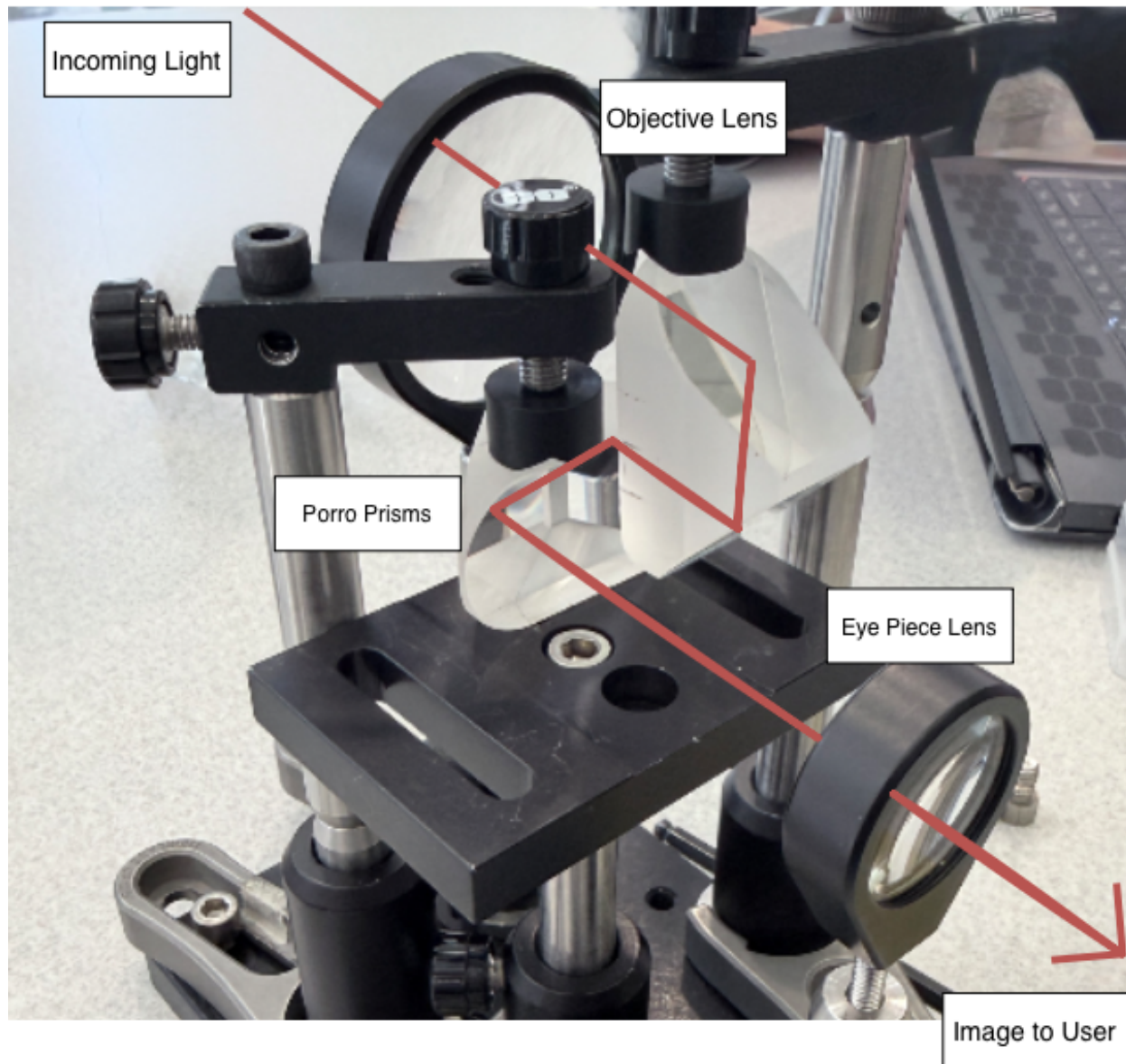


Fig. 66: Annotated Optics Demonstration

Seen above is the optics demonstration made for the CREOL midterm demonstration. This demonstration consists of a basic Keplerian telescope, with a Porro prism set between the lenses. This gives a simple approximation for the system, albeit a far more limited specification version. This demonstration lacks the proper F.O.V. and aberration control that the final system will exhibit. However, it is useful for identifying opto-mechanical design challenges and solutions.

9.2.2 Magnification

Validating the magnification of the system is fairly straightforward, where we will use the secondary function of a telescope: a beam expander. By shooting a laser into the eyepiece lens, the beam will exit the objective lens with a magnified size. By measuring both the initial

and final beam diameter, we can find the system magnification from the equation below.

$$M = \frac{h_{\text{Objective}}}{h_{\text{Eyepiece}}}$$

This can be illustrated by revisiting the paraxial diagram from Chapter 6, to show how the collimated input onto the right side of the system at the eyepiece becomes larger at the output on the left side at the objective. A visualization of this is shown in Figure 67.

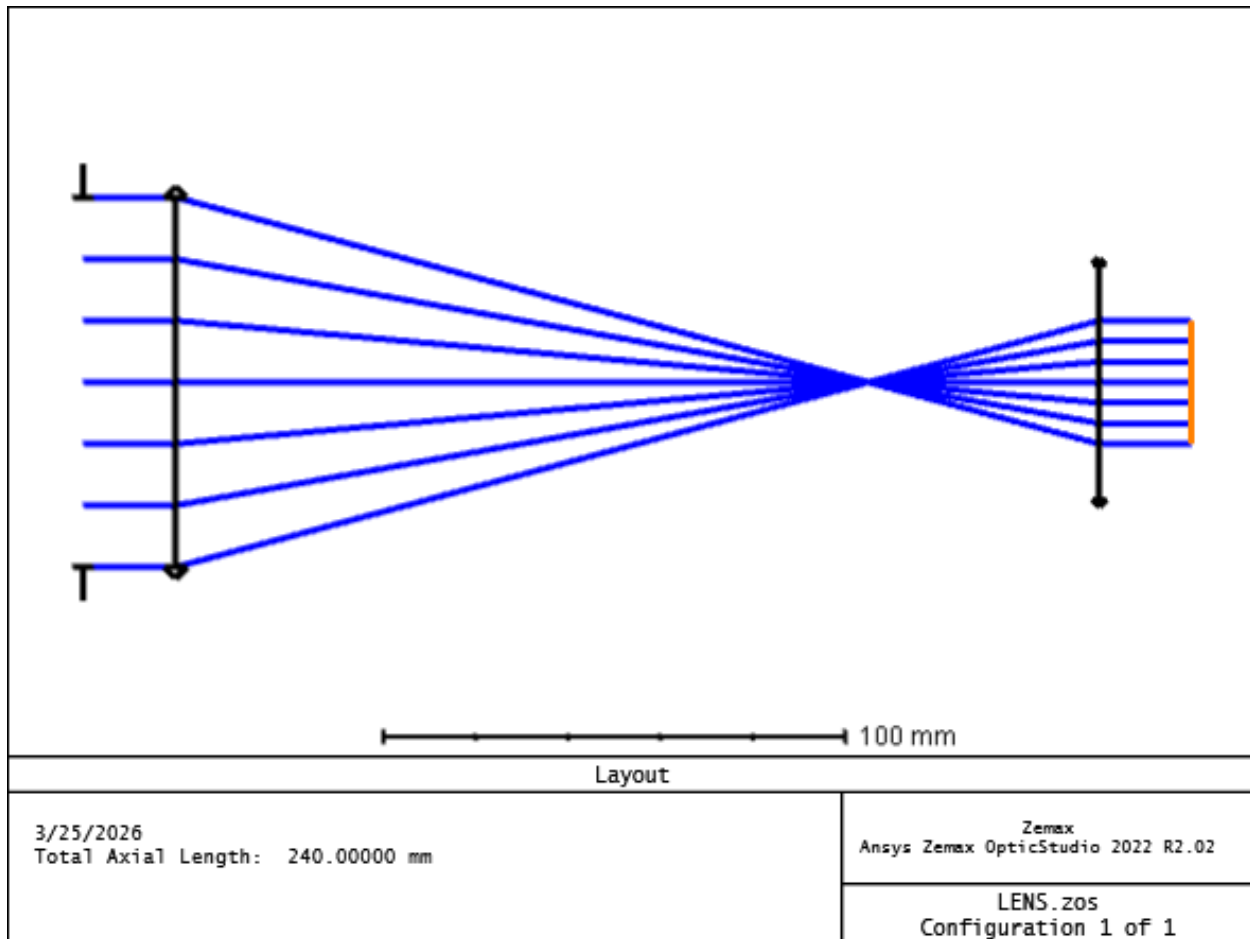


Fig. 67: Beam Expander Diagram

9.2.3 True and Apparent F.O.V.

Measuring the F.O.V. will take more setup, but is fairly trivial. By using one, or several yard sticks laying perpendicular to the optical axis, the system can be placed at a known distance away from the target plane. This known distance is represented as D . The visible width through the binocular system is represented as W . Using these two values, the true F.O.V. can be calculated using the equation below.

$$\theta_{\text{TRUE-FOV}} = 2 \tan^{-1} \left(\frac{W}{2D} \right)$$

The apparent F.O.V. is found by multiplying the magnification M by the true F.O.V., as shown below.

$$\theta_{\text{APPARENT-FOV}} = M \cdot \theta_{\text{TRUE-FOV}}$$

9.2.4 Aberration Testing

The measure of image quality is often subjective given that each person has a unique prescription that causes a different image quality during viewing by person. As such, a more quantitative way to measure image quality is to measure the aberration of the system using interferometry. Using a Zygo interferometer operating at 633nm wavelength, we align the binocular to the optical axis of the interferometer, place a flat mirror at the output the relays the light back through the binocular and into the interferometer to be measured. The measurement from the interferometer provides information regarding the wavefront error, spherical aberration, coma, astigmatism, and optical power. We complete this process individually for both barrels as one contains a beamsplitter that will introduce new optical properties.

9.2.5 HUD Optical Readability

Besides the Bluetooth map integration, the primary way a user can perceive data from the sensor suite is through the integrated H.U.D. system. This is comprised of a micro OLED display displaying sensor data into a beam splitter which directs the image into the optical axis of the binoculars. Great care must then be taken to ensure a high quality viewing experience. We start with our resolution and magnification limitations- we only have so many pixels to shrink screen text, and the 1 F imaging system provides little magnification to the screen. We plan on doing a market-research-style investigation where we poll a focus group on various font sizes. We will ask participants to rank various font sizes on comfort and readability, and will use the size that averages the highest score.

9.2.6 LiDAR Optical Alignment

Ensuring that the LiDAR module is boresighted to the optical axis of the binoculars is critical to ensuring both the functionality and safety of the L.I.N.K. device. To do so, the LiDAR module must be integrated so that its tip and tilt can be adjusted. For this, we take inspiration from standard mirror holders that use tension springs and screws to adjust the tip and tilt, and design a holder for the LiDAR module that follows the same design philosophy and can be easily integrated into the device.

With this mechanism in place, we plan to align the LiDAR using a reticle perfectly centered in the binoculars and an IR camera. Using a printed target at our minimum working range of 50

meters, we plan to align the reticle to the center of the target, then turn on the LiDAR module and measure where the beam hits on the target using the IR camera. We then tune the LiDAR module's alignment to the target's center. This ensures that the LiDAR is aligned and zeroed at a distance of 50 meters. We then slowly move the target back in increments of 50 meters, repeating the alignment process until the laser is no longer visible on the target in the IR camera.

9.3 Hardware Testing

Before any subsystem is asked to play nicely with the others, hardware testing has to confirm that the electrical side can actually carry the embedded device, the sensors, the display, the LiDAR module, and the user input on its own two feet. We split that work into power rail testing, MCU bring-up, bus-level checks, and individual peripheral tests, in that order. The reason for that ordering is risk: power gets validated first so that nothing sensitive is ever connected to a custom PCB rail we have not yet measured.

9.3.1 ESP32-S3 MCU Bring-Up Testing

The ESP32-S3 bring-up test is the gate everything else has to clear: it confirms that the microcontroller will boot cleanly, run firmware, and advertise over BLE before a single sensor is wired in. We do bring-up on the ESP32-S3-DevKitC-1 carrying the WROOM-2 N32R16V module, which is also the module footprint targeted for the custom PCB, so anything that works here should work on the final board. The firmware is built through PlatformIO against the Arduino framework and flashed over the dev board's onboard USB-to-UART bridge.

The procedure walks through four checks in sequence. First, the firmware boots and prints its expected banner over the serial monitor at 115200 baud. That single line tells us the flash image is valid and that none of the GPIOs we picked stomp on a strapping pin or a PSRAM line. The N32R16V routes octal PSRAM through GPIO 33–37, so any peripheral that overlapped that range would refuse to boot at all; the firmware's pin map deliberately steers clear of those pins. Second, the firmware enters its main loop and survives a ten-minute soak with no random resets and no watchdog timeouts. Third, the BLE GATT server initializes, comes up under the device name `LINK-HUD`, and is confirmed visible from the phone via the PWA's BLE pairing screen. Fourth, the firmware's stub-data fallback gets exercised on its own: each sensor flag is disabled in turn, and BLE telemetry is verified to keep streaming at the expected 5 Hz even with no physical sensor attached. Once all four checks pass, every higher-level driver and integration test is unblocked.

9.3.2 I²C Sensor Bus Testing

The very first hardware test we ran on the embedded subsystem was an I²C bus check. The ESP32-S3 was loaded with a small scanner sketch that walks the shared bus and prints every address that responds. The point of doing that before anything else was simple: catch a wiring, pull-up, or power problem before the actual sensor drivers ever get involved. A clean scan that returns the BNO055 IMU, BMP280 barometer, and SSD1306 OLED at their assigned addresses is the green light to start higher-level driver testing.

The shared bus is configured at 400 kHz Fast-mode on GPIO 8 (SDA) and GPIO 9 (SCL). Expected addresses are 0x28 for the BNO055, 0x76 (or 0x77 if the BMP280's SDO line is pulled high) for the BMP280, and 0x3C for the SSD1306. During bring-up all three addresses showed up exactly where we expected, which confirmed two things at once: the 4.7 k Ω pull-ups on SDA and SCL are sized correctly for the actual bus capacitance, and there are no address collisions. We then re-ran the scan with each device disconnected one at a time and watched its address disappear from the output — a quick way to rule out a stuck-bus condition. With the bus electrically validated, we brought up the higher-level drivers one at a time, leaning on each driver's self-test routine to confirm the chip was actually responding to its expected register reads.

9.3.3 UART Peripheral Testing

UART testing covers the two serially attached peripherals: the GT-U7 GNSS receiver and the TFMini-Plus class LiDAR rangefinder. We need to confirm both that the ESP32-S3 enumerates them and that the firmware parses their bytes correctly. The GNSS receiver lives on UART2 at 9600 baud, with GPIO 16 as RX and GPIO 17 as TX, while the LiDAR sits on UART1 at 115200 baud on GPIO 47 (RX) and GPIO 48 (TX). Putting the two on independent UART blocks is deliberate — it stops the high-rate LiDAR traffic from blocking the slower NMEA stream from the GPS.

The procedure for both peripherals is the same in spirit: power the device individually, bridge its UART through to USB on the dev board, and capture the raw byte stream from the serial monitor. For the GNSS receiver the expected output is a 1 Hz stream of NMEA-0183 sentences (`$GPGGA`, `$GPRMC`, `$GPGSA`); once a satellite fix is acquired, the fields populate with real values. With the raw stream confirmed, the TinyGPS++ parser is enabled and verified to reproduce non-zero latitude, longitude, satellite count, and HDOP figures that match the raw NMEA. For the LiDAR the expected output is a fixed-format binary frame opening with the 0x59 0x59 header bytes, repeating at the configured rate. The driver-level parser checks each frame's checksum and returns a range value in centimeters that should grow predictably as the target moves away from the sensor. Both tests need to come back clean before the firmware is allowed to enable either module inside the unified acquisition state machine.

9.3.4 HUD Display and User Input Testing

This test exercises the two pieces of hardware the operator actually touches during a waypoint acquisition: the SSD1306 128 $\times$ 64 OLED that draws the heads-up overlay, and the KY-040 rotary encoder that flips between HUD pages and triggers a ping. The OLED runs over the shared I²C bus at address 0x3C using the `Adafruit_SSD1306` and `Adafruit_GFX` libraries. The encoder takes GPIO 4 and GPIO 5 for its quadrature channels and GPIO 7 for its built-in push-button, all configured with internal pull-ups.

On the display side, we cycle through the five HUD pages the firmware exposes and confirm that each one renders cleanly — no flicker, no partial draws, no clipped characters at the edges. The pages we walk through are heading and pitch, GPS status (with satellite count and fix type), barometer altitude and pressure, LiDAR range and quality flag, and BLE connection status with queued pin count. To pass, a page change has to land on the OLED within one telemetry frame

and the layout has to fit fully inside the screen.

On the input side, we rotate the encoder one detent at a time in both directions and check that the page index advances or backs up by exactly one each time, with no missed steps and no double counts. The push-button is exercised in two modes. A short press is currently a no-op in the firmware, and that is treated as an explicit pass condition rather than a bug. A long press of at least 2.5 seconds is the gesture mapped to a waypoint ping, and the long-press handler is verified by watching the serial log for the snapshot, waypoint compute, and BLE notify messages that arrive in sequence once the threshold is crossed. To keep the encoder switch from chattering, the firmware applies a 250 ms debounce window.

9.4 Sensor Suite Validation

Validating the sensor suite is non-negotiable: every onboard sensor needs to demonstrate that it produces a reliable, repeatable output under conditions the system will actually see in the field. L.I.N.K. carries four sensor modules — the BNO055 IMU, the GT-U7 GNSS receiver, the BMP280 barometer, and a LiDAR rangefinder — and every one of them feeds the waypoint pipeline. Because the final target accuracy is the combined accuracy of all four, we validated each module on its own before trusting their fusion. The firmware’s built-in self-test and module-state reporting (OK, DEGRADED, FAIL) was the day-to-day diagnostic during this work, with the serial debug output capturing raw readings at every stage.

9.4.1 Gyroscope Characterization

The BNO055 9-DOF IMU is the source of the heading, pitch, and roll values that tell the system where the binoculars are pointed. Heading feeds straight into the waypoint math, and that makes even small errors expensive: a 1° heading error compounds into roughly 17.5 m of lateral offset at a 1 km range. That alone is reason enough to put gyroscope characterization near the top of the validation list.

We run the BNO055 in Nine Degrees of Freedom mode, where its onboard processor fuses accelerometer, gyroscope, and magnetometer data into an absolute orientation estimate. The firmware pulls Euler angles over I²C at 20 Hz and inspects the calibration status on every read. Calibration is reported as four scores — system, gyroscope, accelerometer, and magnetometer — each on a 0–3 scale; the firmware flags the module as OK when the system score reaches 2 and as DEGRADED otherwise. For the actual characterization, we mounted the device on a fixed tripod, pointed it at a known landmark whose true bearing we had measured with a handheld compass, and logged the BNO055 heading over 60-second windows for comparison. Once the magnetic calibration had finished its initial settle-in (typically 10–15 seconds of figure-eight motion), the heading readings settled to within roughly $\pm 2^\circ$ of the reference bearing. Pitch and roll were checked by tilting the device to known angles with an inclinometer; both agreed with the inclinometer to within $\pm 1^\circ$. The biggest source of heading drift we saw during testing was magnetic interference from nearby electronics, and the firmware leans on the magnetometer calibration score to catch exactly that case — drop the score below 2 and the module flips to DEGRADED.

9.4.2 Barometer Characterization

The BMP280 gives us pressure, temperature, and a derived altitude estimate. The firmware uses that altitude both as a vertical reference and as a fallback whenever the GNSS receiver is producing a degraded vertical fix. We talk to the sensor over I²C at address 0x76 through the Adafruit BMP280 library, and the firmware exposes pressure in hectopascals, temperature in degrees Celsius, and altitude in meters above sea level. Because raw barometric altitude is only as good as the assumed sea-level reference pressure, the firmware also runs a GPS-assisted calibration routine: when GPS conditions are good enough, it back-solves the local sea-level pressure from a confident GPS altitude and stores that value as the reference for everything that comes next.

The characterization test pokes at three specific properties of that pipeline. The first is short-term stability: leave the device on a bench for 60 seconds and the standard deviation of the reported pressure has to stay under 0.05 hPa, which works out to about 0.4 m of altitude jitter at sea level. The second is calibration convergence: once the firmware has a GPS fix with at least six satellites and a horizontal accuracy of 15 m or better, the calibration routine averages five GPS-derived altitude samples to compute the sea-level reference, and the resulting barometric altitude has to agree with the GPS altitude to within ± 3 m. The third is recalibration cadence: the firmware re-runs that calibration every ten minutes whenever GPS conditions cooperate, and the test confirms that this re-calibration does not produce a visible jump in the reported altitude. If GPS conditions are not met, the firmware falls back to the raw GPS altitude on its own; we exercise that fallback by intentionally degrading the GPS environment and checking that the firmware correctly reports the barometer module as DEGRADED.

9.4.3 GNSS Receiver Characterization

GNSS characterization on the GT-U7 boils down to three properties: time to first fix, satellite count and HDOP under different sky conditions, and NMEA parser correctness. The receiver streams standard NMEA-0183 sentences over UART at 9600 baud, and the firmware leans on the TinyGPS++ library to peel out latitude, longitude, altitude, satellite count, HDOP, and fix age.

Time to first fix is measured from a true cold start: leave the receiver unpowered for at least 30 minutes, then power it back up and time the gap until a 3D fix appears. Under open sky we typically see 30 to 60 seconds, with the receiver locking onto 8–12 satellites and pulling its HDOP under 1.5. Stick the same receiver next to a tall building and that picture changes — time to first fix climbs into the 60–120 second range, satellite count drops to 4–6, and HDOP drifts up to somewhere between 2 and 4. The firmware classifies the resulting fix purely by satellite count: fewer than 3 satellites is no fix, 3–5 is a 2D fix, and 6 or more is a 3D fix.

For parser correctness, we replay a recorded log of NMEA sentences into the firmware over a UART loopback and compare its parsed latitude, longitude, satellite count, and HDOP against a reference parser running on a host computer. Every value has to agree to the last reported decimal digit for the test to pass. The horizontal accuracy figure that gets exposed to the rest of the system is derived by multiplying the reported HDOP by 5; that gives a meters-of-accuracy estimate that is conservative under typical civilian GPS conditions, which is exactly what we

want when the value is feeding a waypoint calculation.

9.4.4 LiDAR Rangefinder Characterization

The LiDAR characterization test exercises three things at once: accuracy, repeatability, and effective range. The module sits on UART1 at 115200 baud and emits binary frames opening with the 0x59 0x59 header, followed by a 16-bit range field in centimeters, a 16-bit signal strength field, and a checksum byte. The firmware parses every frame, validates the checksum, and exposes the result as a range in meters along with a quality flag derived from the signal strength.

The procedure itself is straightforward: clamp the rangefinder onto a tripod aimed at a flat target board, place the board at a series of known distances measured with a calibrated tape, and at each distance log 100 consecutive range frames so we can pull a mean and a standard deviation out of them. To pass, the mean range has to land inside the module's rated accuracy — the larger of ± 5 cm or $\pm 1\%$ of measured range — and the standard deviation across the 100-frame window has to stay within a few centimeters. Effective range is measured by walking the same procedure outward until the signal strength field falls below the module's reliability threshold; at that point the firmware flags the reading as invalid and refuses to feed it into a waypoint computation.

The rated range of the prototype LiDAR module caps the system to relatively short engagements, which is exactly why short-range numbers dominate the waypoint accuracy results presented later in this chapter. For Senior Design II we are planning an upgraded long-range module, and this same procedure will be repeated against extended outdoor reference distances to verify that the higher-range performance carries through.

9.5 Software Testing

Software testing has to cover the embedded firmware, the BLE link, the phone-side PWA, the local storage layer, the backend API, and the map display — essentially every software seam in the project. Because the L.I.N.K. stack is split across the ESP32-S3 and a phone/web application, the work naturally splits the same way: firmware-level tests first, then application-level tests on top.

9.5.1 Embedded Firmware State Machine Testing

This test confirms that the ESP32-S3 firmware moves through its operating states in the right order and that it recovers cleanly when a sensor decides to misbehave. The states we exercise are boot, sensor initialization, idle telemetry streaming, acquisition triggered by a long-press of the encoder, LiDAR read, waypoint computation, BLE notification, and the degraded operating mode that kicks in when one or more sensors fail their self-test.

The whole thing is driven from the serial debug log while the firmware runs through a complete acquisition cycle. Right after reset the firmware needs to print its boot banner, finish I²C and

UART initialization, and run every enabled driver's self-test routine. Each driver returns OK, DEGRADED, or FAIL, and the aggregated state shows up on the HUD status page. With initialization complete, the firmware should drop into the idle state and start streaming the telemetry JSON payload over BLE at 5 Hz while polling sensors at 20 Hz. A long-press on the encoder must move the firmware into the acquire state, where it grabs a synchronized snapshot of the latest GPS, IMU, barometer, and LiDAR readings, computes the target waypoint, parks the result in the local pin queue, and pushes a BLE notification on the pin characteristic. After the BLE acknowledgment comes back, the firmware needs to return cleanly to idle.

Degraded operation is exercised the obvious way: we disable each sensor in turn through its compile-time feature flag and check that the firmware substitutes stub data, marks the affected module as FAIL on the HUD, and keeps streaming telemetry without crashing. The waypoint computation is also gated on the GPS and IMU module states, so the test makes sure that an acquisition attempt with either of those in FAIL is rejected with a logged error — not a crash — and that the firmware as a whole stays responsive afterwards.

9.5.2 Waypoint Calculation Testing

The waypoint calculation test answers a simple question: given an observer location, a measured range, and a heading, does the firmware put the target where it actually is? We feed the calculation a known latitude, longitude, bearing, and range and compare its output against an independently computed reference. This matters because every other piece of the chain can be working flawlessly and the pin can still land in the wrong spot if the math is off.

9.5.3 BLE Communication Testing

This test confirms three things about the BLE link in one go: that the ESP32-S3 advertises under the right device name, that it exposes the GATT service and characteristics we expect, and that it reliably delivers both the streaming telemetry payload and event-driven waypoint notifications all the way to the phone application. The firmware exposes a single primary service with four characteristics. The phone subscribes to the telemetry characteristic for the 5 Hz JSON snapshot. The pin characteristic emits a notification each time a new waypoint is computed. The acknowledgment characteristic is the path the phone writes back over to confirm receipt of a pin. The fourth characteristic is for IMU calibration commands. Both the firmware and the phone-side code carry the same hardcoded 128-bit UUIDs for these so the two ends can never get out of sync silently.

The procedure is to put the device into advertising mode, pair from the phone using the PWA's BLE pairing screen, subscribe to the telemetry and pin characteristics, and then watch the data flow over a fixed window. To pass, the device name `LINK-HUD` has to show up in the advertising scan results, telemetry notifications have to arrive at a rate consistent with the configured 5 Hz interval, and a long-press of the encoder has to produce exactly one pin notification per acquisition. Each pin payload is then validated field by field against the values reported in the most recent telemetry frame, so we can confirm that the snapshot taken at acquisition time really is the one that propagated through the BLE pipeline. Finally, the acknowledgment write path is checked the obvious way: turn off Bluetooth on the phone, watch the firmware retry the unacknowledged pin after the 10-second retry interval, and confirm that the retries stop the

moment the phone re-pairs and writes the ack.

9.5.4 PWA and Server Testing

This test picks up where the BLE communication test leaves off. From the moment the PWA receives a pin payload, it has to persist that payload to the browser's IndexedDB store, attempt an upload to the backend server over HTTPS, and finally render the resulting marker on the Leaflet map. The backend itself is a Node.js Express service backed by a SQLite database, and it exposes REST endpoints for creating pins, listing pins, and acknowledging delivery.

We exercise each piece of that pipeline in isolation before stitching the whole thing together. The map renderer goes first: a hard-coded waypoint is fed into the React component and we check that the marker lands at the expected latitude and longitude with the right popup metadata (timestamp, range, heading, target identifier). The server REST endpoints come next, poked with a tool such as `curl`: a POST to the pin-creation endpoint with a known payload has to return a 200, the SQLite database has to actually contain the new row afterward, and a follow-up GET to the listing endpoint has to include that row in its response. The server uses an `INSERT OR IGNORE` pattern keyed on the pin identifier; we verify that idempotency by sending the same payload twice and confirming that the second request does not produce a duplicate row. Finally, with all the pieces individually proven, we trigger a real acquisition on the ESP32 and confirm that the marker shows up on the map within a small number of seconds, with every metadata field matching what the firmware reported.

9.5.5 Offline Queue and Synchronization Testing

This test answers a question that matters more than any other for outdoor field use: can the system capture waypoints when the network is gone, and can it reliably push them to the server when connectivity comes back? Cellular coverage in the field is intermittent, and an operator should never have to wait for an HTTP round-trip before taking the next reading. Two queues sit on the data path here, and both have to behave: the local queue inside the ESP32 firmware that holds up to 32 pins until a BLE acknowledgment shows up, and a queue inside the PWA that holds pins in IndexedDB until the upload to the server actually succeeds.

We set the test up by putting the phone in airplane mode, manually re-enabling BLE, and loading the PWA from its service-worker cache. From there, three waypoints are captured back-to-back by holding down the encoder on the ESP32. Each pin has to land on the map immediately, with a visual indicator that flags it as pending synchronization, and the IndexedDB store should contain three matching rows when we crack open the browser developer tools. With the queue confirmed populated, we take the phone out of airplane mode and let the PWA retry on its own. Once the retry settles, every row should be marked synchronized, the server's SQLite database should hold three new rows, and a hard reload of the PWA should pull those same three pins back from the backend instead of from local storage. To confirm the server's idempotent `INSERT OR IGNORE` actually does what it says, we deliberately reload during the upload window: even if the PWA retries an upload that has already succeeded, no duplicate rows are produced server-side.

9.6 Accuracy Validation

9.6.1 GPS/User Mapping Accuracy

GPS validation came down to one practical question: are the positions reported by the GT-U7 module and the phone's built-in GPS good enough to drop the user pin on the map within an acceptable margin of error? The GT-U7 talks to the ESP32 over UART at 9600 baud and emits standard NMEA sentences, which are unpacked by TinyGPS++. From there, the firmware estimates horizontal accuracy by multiplying the reported HDOP by 5 (yielding a value in meters) and classifies the fix purely on satellite count: under 3 is no fix, 3–5 is a 2D fix, and 6 or more is a 3D fix.

Out in the field, the GT-U7 picked up a 3D fix within 30 to 60 seconds of power-on under open sky every single time, locking onto 8–12 satellites. We compared the reported positions against known reference coordinates and found them accurate to within roughly 3–5 m horizontally under normal conditions, which lines up with what civilian GPS is supposed to deliver. Step into a degraded environment — next to a building, under heavy tree canopy — and the satellite count dropped to 4–6 and the horizontal accuracy slipped to about 8–12 m, at which point the firmware correctly bumped the module into DEGRADED. On the phone side, the browser's Geolocation API exposes its own accuracy figure through `coords.accuracy`, and the PWA renders it as the blue accuracy circle around the user's marker. With both devices co-located, the phone GPS and the ESP32 GPS consistently reported positions within 5–10 m of each other.

9.6.2 Point of Interest/Target Mapping Accuracy

The one accuracy number that matters most for L.I.N.K. is how close the computed waypoint actually lands to whatever the operator was looking at through the binoculars. That target coordinate is built from three measurements — the observer's GPS position, the IMU heading, and the LiDAR range — which means the final accuracy is a function of three independent error sources stacked on top of each other. GPS error shifts the entire solution rigidly. Heading error shows up as lateral displacement that scales with the range. Range error pushes radially along the line of sight.

The lateral component caused by heading error is well-approximated by:

$$E_{\text{heading}} = R \cdot \sin(\theta_{\text{error}})$$

with R the measured range and θ_{error} the heading error in degrees. That relationship is exactly why IMU heading accuracy matters more and more as targets get further away — a 1° heading error works out to about 4.36 m of lateral error at 250 m and roughly 8.73 m at 500 m. The implication is that final waypoint accuracy has to be evaluated as a combined GPS, IMU, and LiDAR problem, not as any single sensor's spec sheet.

To characterize end-to-end target accuracy, we have a series of field tests planned in which the binoculars are aimed at landmarks whose true GPS coordinates we already know — either from

a reference survey or from a high-quality phone GPS reading taken at the landmark itself. At each test point, an operator will trigger a waypoint via the encoder long-press, and the resulting target coordinates will be compared against ground truth using the haversine distance formula. The plan covers both short ranges around 50–100 m, where heading error contributes only a few meters of lateral offset, and medium ranges of 200–500 m, where heading error becomes the dominant term and the LiDAR’s accuracy starts to matter. Three independent trials are taken at each range, and the mean radial error is reported. Based on the heading and range analysis above and our validated 3–5 m GPS accuracy, we expect short-range targets to come in within roughly 5–10 m of truth and medium-range targets within 10–25 m, with the dominant error source shifting from GPS to heading as range grows.

9.7 Performance Evaluation

The performance evaluation pulls L.I.N.K.’s current measured (or expected) numbers up against the major design goals. At this stage of development some of those numbers are validated end-to-end on dev boards and software prototypes, while others will only land in their final form during Senior Design II once the PCB has been fabricated and the mechanical integration is complete.

TABLE XLV: L.I.N.K. Performance Evaluation Summary

Metric	Target	Measured/Current Result	Evaluation
LiDAR Range	250 m minimum desired	Pending field test	Must be verified outdoors in SD2
GNSS Position Capture	Valid latitude/longitude	Partial module test	Needed before full waypoint testing
IMU Heading Output	Stable heading for waypoint bearing	Partial sensor test	Requires calibration and magnetic interference testing
BLE Transfer Latency	Near real-time transfer to phone	Pending full BLE test	Must be tested with repeated packets
Map Pin Display	Pin appears after acquisition	Partial software prototype	Verify with real ESP32 packet
Offline Operation	Store waypoint without network	Planned	Test with phone network disabled
Power Rail Stability	Stable 3.3 V and 5 V rails	Pending PCB/load test	Required before final prototype
HUD Readability	Telemetry visible in optical path	Planned	Must be tested in indoor and outdoor lighting

9.8 Overall System Integration

The overall integration test is the single most important test L.I.N.K. has to pass: it walks the complete workflow from optical target selection all the way through to map waypoint display. Everything before this point validates one slice of the system in isolation; this test is what proves that the optical, electrical, embedded, and software pieces actually behave like one product instead of four loosely related demos.

TABLE XLVI: End-to-End Integration Test Sequence

Step	Subsystem Tested	Expected Result
1	Power system and ESP32 boot	Device powers on and firmware enters initialization state
2	GNSS, IMU, barometer, and HUD	Sensor status is displayed on HUD
3	Optical aiming and LiDAR acquisition	Valid range value is captured after user input
4	Waypoint computation	Destination coordinate is generated from location, heading, and range
5	BLE communication	Phone receives waypoint packet
6	PWA and server	Waypoint is stored and displayed on map
7	Offline synchronization	Locally stored waypoint syncs when network returns

The integration test follows the same sequence we used during partial bench validation. The device is powered from a fully charged 18650 cell and allowed to finish its boot and self-test sequence; once that completes, we glance at the HUD status page and confirm that the GNSS, IMU, barometer, and LiDAR modules all read OK. The phone is paired over BLE and the PWA opens onto a fresh map view. From there the user aims the binoculars at a pre-surveyed landmark, holds the encoder past the long-press threshold, and the marker should appear on the phone map within a few seconds. The marker's coordinates get compared against the surveyed position to fill in one row of the target mapping accuracy table, and the same procedure is repeated at three different ranges to populate the rest of the table. As a final pass, the entire test is run again with the phone placed in airplane mode at the start of the run — that exercises the offline queue path — and the total elapsed time from acquisition to map display is recorded as the end-to-end latency for that configuration. Cleanly running this entire procedure across all planned trials is the formal pass condition for the L.I.N.K. integrated prototype.

9.9 Plan for Senior Design II

The goal in Senior Design II is to move out of isolated subsystem testing and into full prototype validation. We are tackling that in three stages.

Stage one is PCB fabrication and board bring-up for the power and MCU/sensor boards. Each voltage rail gets tested independently before any sensitive component — the ESP32-S3, the IMU, the GNSS receiver, the OLED display, the LiDAR module — is asked to sit on it. With the electronics proven, we expand the firmware out from individual driver tests into the full acquisition state machine.

Stage two is optical and mechanical integration. The LiDAR module, the HUD display, the beamsplitter, and the binocular optical path all have to be aligned so that the displayed reticle actually points at whatever the LiDAR is ranging. This stage also picks up outdoor range testing, HUD readability checks, and repeated waypoint acquisitions across multiple distances.

Stage three is end-to-end demonstration readiness. The embedded device sends waypoint packets to the phone application over BLE, the PWA stores and displays each waypoint, and the server synchronizes stored pins whenever a network is available. Final testing measures system latency,

waypoint accuracy, battery runtime, and reliability across many repeated acquisition cycles — the kind of numbers that have to hold up under demonstration conditions.

TABLE XLVII: Senior Design II Testing and Integration Plan

Phase	Planned Work	Deliverable
PCB Bring-Up	Assemble and test power PCB and MCU/sensor PCB. Verify 3.3 V and 5 V rails before attaching full system loads.	Functional custom boards
Firmware Integration	Combine GNSS, IMU, barometer, LiDAR, HUD, and BLE drivers into one state machine.	Integrated embedded firmware
Optical Alignment	Align LiDAR, HUD reticle, and binocular optical path. Test HUD readability and target alignment.	Aligned optical/electronic prototype
Software Integration	Connect ESP32 BLE output to the PWA, local storage, REST API, and map display.	End-to-end waypoint transfer
Field Testing	Test range acquisition, waypoint accuracy, BLE reliability, and map pin placement outdoors.	Performance evaluation data
Final Demo Preparation	Repeat full acquisition workflow and resolve reliability issues.	Demonstration-ready prototype

10 Administrative Content

Based on Table, as shown below, the estimated building cost for this project is \$1652.50. There are many estimated general costs, and this value does not include iterations or exact components; therefore, the total budget is set to \$1700 to cover the cost of extraneous variations and variables.

10.1 Project Milestones

Table 1: Project Milestones

Task	Dates	Responsibility
Project proposal assignment	1/12 – 1/23	All
Write and submit D&C first draft	1/23 – 1/30	All
Meet with advisors and discuss necessary changes	2/4	All
Revise D&C and resubmit	2/4 – 2/6	All
Research components for each L.I.N.K. subsystem	2/6 – 2/23	All
Purchase breakout boards and verify via firmware	2/26 – 3/6	EE & CPE
Draft Zemax optical model and optimize	1/30 – 2/27	PSE

Task	Dates	Responsibility
Design 3D model (opto-mechanics + electronics)	1/30 – 2/27	PSE
OSE Early Demo	3/3	All
Finalize optical elements and procure	3/2 – 3/13	PSE
Extended HUD testing	3/2 – 3/16	PSE
Prototype Firmware V1	3/13 – 3/27	CPE
Begin PCB schematics	2/26 – 4/15	EE
Schematic review	4/15 – 4/18	All
Firmware V2	3/20 – 4/3	CPE
Website V1	3/20 – 4/3	CPE
Server/API V1	3/20 – 4/3	CPE
Midterm Report submission	2/23 – 3/27	All
3D printed lens cradle	2/27 – 3/27	PSE
Midterm revisions	3/30 – 4/3	All
End-to-End prototype validation	4/3 – 4/17	All
Offline handling demo	4/10 – 4/24	CPE
Demo video creation	4/3 – 4/28	All
Improved optomechanical demo	4/28	All
Final Report submission	4/3 – 4/28	All
Send Gerber files for fabrication	4/28 – 5/11	EE
Website V2 polish	4/28 – 5/29	CPE
3D print assembly	5/11 – 5/22	PSE
Mount components and reflow	5/18 – 6/1	EE
Mount optical components	5/25 – 6/1	PSE
Test power PCB	6/1 – 6/15	EE
Test sensor outputs	6/1 – 6/8	EE & CPE
Port firmware to custom PCB	6/8 – 6/22	CPE
Full system integration	6/22 – 7/6	All
Power validation testing	7/1 – 7/9	EE
Final End-to-End system test	7/1 – 7/9	All
Final Presentation	7/23	All

10.2 Bill of Materials

Table 2: Bill of Materials

Item	Price	Qty	Cost	Details
Optical Components				
LiDAR Module	\$100	1	\$100	Laser Rangefinder
Achromat Objective Lens	\$181	2	\$362	Binocular Objectives
Eyepiece Lens	\$43.10	2	\$86.20	Viewing Optics
Focal Extender	\$44.25	2	\$88.50	Allows for Field Lens
Field Lens	\$40.50	2	\$81	Aberration Control
Porro Prisms	\$100	2	\$200	Image Inverter
Beam Splitter	\$389	1	\$389	HUD Combiner
Prototype Demo				
ESP32 DevKitC	\$10	1	\$10	Fast firmware bring-up + USB programming
LiPo Battery (3.7V, ~2800mAh)	\$7	3	\$21	Rechargeable demo power
External Battery Charger	\$15	1	\$15	Battery wall charger
OLED 128×64 I ² C Module	\$20	1	\$20	Early HUD testing display
9-DoF IMU Breakout	\$22	1	\$22	Rapid motion sensing tests
Rangefinder Module	\$100	1	\$100	Software integration testing
Jumpers / Wiring / Mounting	\$15	1	\$15	Miscellaneous hardware
Custom PCB — Integrated Circuits & Modules				
ESP32-S3-WROOM-2-N32R16V	\$12.19	2	\$24.38	Final MCU module, 32MB flash
NEO-M9N-00B	\$14.69	2	\$29.38	u-blox GNSS receiver
CP2102-GMR	\$9.54	2	\$19.08	USB-to-UART bridge

Item	Price	Qty	Cost	Details
LM2596S-3.3/NOPB	\$7.83	2	\$15.66	3.3V buck regulator, 3A
LM2596S-5.0/NOPB	\$7.83	2	\$15.66	5V buck regulator, 3A
BME280	\$4.00	3	\$12.00	Pressure / humidity / temp sensor
Custom PCB — Discrete Components				
SRR1260-330M	\$1.05	6	\$6.30	33 μ H shielded inductor, 3A
1N5822	\$0.61	6	\$3.66	Schottky catch diode, 40V 3A
MBR0520LT1G	\$0.35	3	\$1.05	Schottky diode, 20V 500mA
BAT54HT1G	\$0.15	3	\$0.45	Schottky flyback diode
MMBT2222ALT1G	\$0.15	3	\$0.45	NPN buzzer driver transistor
ESD5B5.0ST1G	\$0.14	10	\$1.41	TVS diode, USB ESD protection
Status LEDs (R/G/B)	\$0.19	9	\$1.71	0805 indicator LEDs
Passives Budget	\$5	1	\$5	Caps and resistors
Custom PCB — Peripherals, Power & Interconnect				
SMT-0440-T-R	\$3.76	3	\$11.28	Magnetic buzzer, 3V 4 $\times$ 4mm
W3011	\$2.01	3	\$6.03	1.575GHz GNSS chip antenna
MS621FE-FL11E	\$2.51	3	\$7.53	GNSS backup coin cell, 3V
ASR00050	\$6.95	3	\$20.85	18650 Li-Ion cell, 3.7V 2.5Ah
106990288	\$0.60	2	\$1.20	3-cell 18650 holder
500SSP1S2M2QEA	\$3.52	4	\$14.08	SPDT slide power switch, 5A

Item	Price	Qty	Cost	Details
0480371000	\$1.48	3	\$4.44	Molex USB Type-A plug, R/A SMD
JST PH Headers (2/3/4/8-pos)	\$0.42–0.86	40	\$21.90	2mm R/A board connectors
JST 1mm Display Connector	\$0.12–0.68	10	\$4.00	SM06B-SRSS-TB + SHR-06V housing
OLED Module	\$20	1	\$20	Final HUD display module
PCB Fabrication	\$262.50	1	\$262.50	Prototype PCB batch
Optical Components Subtotal				\$1306.70
Prototype Demo Subtotal				\$203.00
Custom PCB Subtotal				\$510.00
Total				\$2019.70

10.3 Distribution of Work

Table 2: Team Roles and Responsibilities

Role	Name	Responsibilities
Computer Engineer	Santiago Castro	ESP32 firmware Mapping UI + waypoint pipeline Backend/API integration Website design and management
Optics and Photonics Engineer	Brian Corchuelo	LiDAR system design and alignment Optomechanical design and testing 3D printing optimization Proof-reading and consistency checking
Electrical Engineer	Dylan Farnsworth	Design PSU PCB schematic + layout for regulators, MCU + sensor board Assembly and soldering of electrical components Verify voltage values and test functionality

Role	Name	Responsibilities
Optics and Photonics Engineer	Jeremy Goodenough	3D diagram modeling Integrated HUD design and testing Lead editor and submission responsibility Zemax optical design of binocular optics

10.4 Sponsorship

A Significant portion of the Optics in this project were purchased by Edmund Optics. Without this sponsorship, the imaging quality, as well as tolerenceing of the Optics would be far lower, and the aberration control would have been much worse. The parts sponsored by Edmund Optics are listed below:

Table 1: Sponsored Parts

Item	Price	Qty	Cost	Details
Achromat Objective Lens	\$181	2	\$362	Binocular Objectives
Eyepiece Lens	\$43.1	2	\$87	Viewing Optics
Focal Extender	\$44.25	2	\$88.5	Allows for Field Lens
Field Lens	\$40.5	2	\$81	Aberration Control
Beam Splitter	\$389	1	\$389	HUD Combiner

11 Conclusion

Through the L.I.N.K. project, four undergraduate students from distinct ECE and CREOL disciplines collaboratively applied and extended their technical skillsets to address the challenges inherent in a tightly integrated optoelectronic system. This work encompassed custom optical system design, LiDAR-based ranging integration, embedded programming using the ESP32 microcontroller platform, and PCB-level hardware development, all within a multidisciplinary engineering framework. The primary objective of L.I.N.K. is to provide a user-centric system capable of translating real-world target locations into a digitally realized representation, enabling accurate visualization of both user and target positioning across external web-based platforms.

This report documents the complete design process, from initial system requirements and architectural decisions through subsystem implementation and iterative refinement. Particular emphasis is placed on the deliberate incorporation of unfamiliar tools and methodologies, reflecting a

design philosophy centered on both functional performance and technical growth. The resulting system demonstrates successful integration of optical sensing, data acquisition, and computational processing, validating the feasibility of the proposed concept. The students' methodical approach—characterized by rigorous analysis, subsystem testing, and iterative optimization—has ensured steady progress toward a deployable prototype.

Overall, the outcomes of this project indicate that the L.I.N.K. system has met its core design objectives at the prototyping level. The insights gained through this process not only confirm the viability of the system architecture but also establish a strong foundation for future refinement, including improved optical alignment tolerances, enhanced sensor accuracy, and expanded software functionality. Collectively, these results demonstrate that the L.I.N.K. platform represents a credible and scalable solution for real-time spatial awareness and digital environment mapping.

Appendix

A

References

- [1] Safran Vectronix AG, *VECTOR X User Manual*, 2023.
- [2] R. Fisher, "Head-Up Displays," in *Handbook of Optics*, 2010.
- [3] R. Spitzer, *Electro-Optical Systems Design Handbook*, 2000.
- [4] J. Zhang and Y. Li, "GNSS-Based Positioning," 2016.
- [5] L. Meng et al., "Map-based Mobile Services," 2005.
- [6] P. D. Groves, *GNSS Navigation Systems*, 2013.
- [7] Bluetooth SIG, "Bluetooth Core Specification Version 5.0," 2016.
- [8] M. Satyanarayanan, "Edge Computing," 2017.
- [9] Mapbox Inc., "Mapbox GL JS API."
- [10] Espressif Systems, "ESP32-S3 Datasheet."
- [11] Bluetooth SIG, "Bluetooth Core Specification 5.4," 2023.
- [12] T. Bray, "JSON Format," RFC 8259.
- [13] W3C, "Web App Manifest."
- [14] W3C, "Service Workers."
- [15] R. Fielding et al., "HTTP Semantics," RFC 9110.
- [16] Espressif Systems, "ESP32 Technical Reference Manual."
- [17] NXP, "I2C Specification."
- [18] M.-C. Amann et al., "Laser ranging," 2001.
- [19] B. Schwarz, "LiDAR," 2010.
- [20] A. McCarthy et al., "Depth imaging," 2009.
- [21] Y. Li et al., "Fog effects on LiDAR," 2017.
- [22] R. Lange, "ToF camera," 2001.
- [23] A. Kolb et al., "ToF imaging," 2010.
- [24] R. Raschofer, "Radar/LiDAR systems," 2011.
- [25] B. Behroozpour et al., "LiDAR architectures," 2017.
- [26] J. Bosse, "3D scan matching," 2019.
- [27] B. Saleh, *Fundamentals of Photonics*, 2019.
- [28] IEC, "Laser Safety Standard," 2014.
- [29] Samsung SDI Co., Ltd., "ICR18650-28A Product Specification."
- [30] Analog Devices, "LTC3113 Datasheet."
- [31] J. Zhang and X. Zhao, "Solid-state electrolytes for lithium-ion batteries," *Journal of Power Sources*, vol. 196, no. 13, pp. 6007–6013, 2011.

- [32] "Alkaline Manganese Datasheet (LR6/AA)."
- [33] Panasonic Industry Europe GmbH, "NCA-463436A Prismatic Cell Product Specifications."
- [34] Texas Instruments, "TPS61022 Datasheet."
- [35] Texas Instruments, "WEBENCH Power Designer."
- [36] IEEE, "802.11 Standard."
- [37] Arduino, "Arduino Documentation."
- [38] Bosch, "BNO055 Datasheet."
- [39] Bosch, "BMP280 Datasheet."
- [40] Solomon, "SSD1306 Datasheet."
- [41] Adafruit, "SSD1306 Library."
- [42] PlatformIO Labs, "PlatformIO Documentation."
- [43] Espressif Systems, "ESP-IDF Programming Guide," 2026.
- [44] Amazon, "FreeRTOS Kernel Reference Manual."
- [45] D. George et al., "MicroPython Documentation."
- [46] M. Hart, "TinyGPS++."
- [47] NMEA, "NMEA 0183."
- [48] W3C, "Web Bluetooth."
- [49] W3C, "Geolocation API."
- [50] W3C, "IndexedDB."
- [51] Microsoft, "TypeScript."
- [52] MapLibre, "Docs."
- [53] Leaflet, "Docs."
- [54] Meta, "React Docs."
- [55] Node.js, "Documentation."
- [56] Express, "Framework."
- [57] Vite, "Frontend Tool."
- [58] SQLite, "Documentation."
- [59] E. Hecht, *Optics*.
- [60] W. Smith, *Optical Engineering*.
- [61] Thorlabs, "Beamsplitters."
- [62] J. Greivenkamp, *Geometrical Optics*.
- [63] R. Kingslake, *Lens Design*.
- [64] Edmund Optics, "Binocular Design."
- [65] Thorlabs, "Prisms."
- [66] MicroOLED, "OLED Displays."
- [67] UL Standards, "UL 1642: Standard for Lithium Batteries," Underwriters Laboratories, 2012.
- [68] International Electrotechnical Commission, "IEC 62133-2: Safety Requirements for Portable Sealed Secondary Lithium Cells," IEC, 2017.
- [69] United Nations, "UN Manual of Tests and Criteria, Part III, Sub-section 38.3: Lithium Metal and Lithium Ion Batteries," 2020.
- [70] Avantier Inc., "A Comprehensive Guide to ISO 10110 Optical Drawing Standards."
- [71] N. Mohan, T. M. Undeland, and W. P. Robbins, *Power Electronics: Converters, Applications, and Design*, 3rd ed. Hoboken, NJ, USA: Wiley, 2002.
- [72] u-blox, "GNSS Datasheet."
- [73] C. Veness, "Haversine."
- [74] Adafruit, "GFX Library."
- [75] W3C, "Notifications API."
- [76] J. Archibald, "idb."
- [77] OpenStreetMap, "Policy."
- [78] J. Wheeler, "better-sqlite3."
- [79] Analog Devices, "LTC3539 Datasheet."
- [80] D. Malacara, *Optical Testing*.
- [81] C. Padgham, *HUD Design*.
- [82] Zemax, "Optical Tutorials."
- [83] Kaplan, *GPS GNSS*, 2017.
- [84] Titterton, *Inertial Navigation*, 2004.
- [85] M. Jones, "JWT," RFC 7519.
- [86] Motorola, "SPI Guide."
- [87] TI, "UART App Note."

B Component Datasheets

This appendix lists the datasheets for the major hardware components selected during the research and part-selection process described in Sections 2 and 3. Each entry corresponds to a part chosen in a “Final Choice” paragraph of Section 3 and is the authoritative document used to validate electrical, mechanical, and protocol specifications during schematic capture, firmware bring-up, and PCB layout. Full PDF copies of each datasheet are archived in the project repository under `docs/datasheets/`.

- **Espressif ESP32-S3 Series Datasheet** (system MCU, Section 3 Final Choice of MCU Part). Provides the Xtensa LX7 dual-core specifications, Wi-Fi/BLE radio characteristics, GPIO matrix, and I²C/UART/SPI peripheral parameters used to budget the I²C bus and power rails. https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf
- **Espressif ESP32-S3 Technical Reference Manual** (companion to the MCU datasheet). Verifies FreeRTOS interactions, timer/PWM configuration, and the dual I²C controller architecture referenced in the firmware architecture analysis. https://www.espressif.com/sites/default/files/documentation/esp32-s3_technical_reference_manual_en.pdf
- **Pengtu PTYS-12X-251028 905 nm Time-of-Flight LiDAR Module Datasheet** (rangefinder, Section 3 Final Choice of Rangefinder Part). Documents the UART/TTL output frame format (header 0x59 0x59, range and signal-strength fields, checksum), the Class 1 eye-safety classification, the 1200 m daytime / 2000 m nighttime range capability, and the 3.3 V supply requirement.
- **Bosch Sensortec BNO055 Intelligent 9-axis Absolute Orientation Sensor Datasheet** (IMU, Section 3 Final Choice of IMU Part). Provides the I²C address (0x28/0x29), Nine-Degrees-of-Freedom fusion-mode register map, calibration-status byte format, and Euler-angle output ranges used by `imu_driver.cpp`. <https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bno055-ds000.pdf>
- **Bosch Sensortec BMP280 Digital Pressure Sensor Datasheet** (barometer, Section 3 Final Choice of Barometer Part). Specifies the I²C address (0x76/0x77), pressure/temperature compensation formulas, and absolute accuracy figures used to derive the GPS-assisted sea-level calibration routine. <https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmp280-ds001.pdf>
- **u-blox NEO-7 Series GNSS Modules Data Sheet** (reference platform for the GT-U7, Section 3 Final Choice of GNSS Part). Documents the NMEA-0183 sentence set parsed by TinyGPS++, cold/warm/hot start times, sensitivity figures, and the UART default baud rate (9600) wired to the ESP32-S3.
- **Goouuu Tech GT-U7 GPS Module User Manual** (vendor document for the actual module). Lists pinout, on-board active-antenna IPEX connector, USB-to-serial bridge, and 3.3 V/5 V tolerant supply range used during prototype bring-up.
- **Solomon Systech SSD1306 128×64 OLED Driver Datasheet** (HUD display controller, Section 3 Final Choice of Display Part). Documents the I²C/SPI command set, GDDRAM layout, and contrast/brightness command codes. <https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf>
- **Alps Alpine EC11 Series Rotary Encoder with Push Switch Datasheet** (user-input device, Section 3 Final Choice of User Input Method). Documents the quadrature output timing, detent specifications, and integrated push-switch contact ratings used to derive the

firmware debounce constants and long-press capture threshold.

- **Analog Devices (Linear Technology) LTC3113 Buck-Boost DC/DC Converter Datasheet** (3.3 V regulator, Section 3 Final Choice for 3.3V Regulator). Provides efficiency curves at 3.3 V output, switching frequency, output ripple, and inductor/capacitor selection guidance. <https://www.analog.com/media/en/technical-documentation/data-sheets/3113fa.pdf>
- **Analog Devices (Linear Technology) LTC3539/LTC3539-2 Synchronous Step-Up DC/DC Converter Datasheet** [79] (5 V regulator supplying the LiDAR, Section 3 Final Choice for 5V Regulator). Documents the 900 mA output capability, soft-start behavior, and quiescent current that drive the system battery-life calculations. <https://www.analog.com/media/en/technical-documentation/data-sheets/3539fb.pdf>
- **Samsung SDI INR18650-25R Lithium-ion Cell Specification** (representative 18650 cell, Section 3 Final Choice of Battery Selection). Provides the nominal 3.7 V terminal voltage, 2500 mAh nominal capacity, discharge curve shape, and protection-circuit requirements that drive the safety analysis in Section 4 (UL 1642, IEC 62133, UN 38.3).

C Software Source Code Listings

This appendix collects the core source listings referenced throughout the Software Design section (Section 8). The listings below are excerpts of the production code that runs on the L.I.N.K. system; full files are available in the project Git repository. Excerpts have been chosen to expose the algorithms, data structures, and protocol handlers that are most relevant to the design discussion. Comments and formatting are preserved as they appear in the working firmware and PWA codebase.

A.3.1 Firmware: Haversine Forward Projection (`waypoint.h`)

The destination-point computation invoked at the moment the ping button is pressed. Both the firmware and the PWA implement the same closed-form formula on a spherical Earth (see Section 8.5.4).

```

1 // esp32-firmware/src/waypoint.h
2 #pragma once
3 #include <cmath>
4 #include config.h
5
6 struct LatLon { double lat; double lon; };
7
8 inline LatLon destinationPoint(double latDeg, double lonDeg,
9                               float bearingDeg, float distanceM) {
10     const double R      = cfg::EARTH_RADIUS_M;
11     const double lat1 = latDeg      * M_PI / 180.0;
12     const double lon1 = lonDeg      * M_PI / 180.0;
13     const double brng = bearingDeg * M_PI / 180.0;
14     const double dr   = distanceM / R; // angular distance
15
16     double lat2 = asin(sin(lat1) * cos(dr) +
17                       cos(lat1) * sin(dr) * cos(brng));
18     double lon2 = lon1 + atan2(sin(brng) * sin(dr) * cos(lat1),
19                               cos(dr) - sin(lat1) * sin(lat2));
20
21     LatLon result;
22     result.lat = lat2 * 180.0 / M_PI;
23     result.lon = lon2 * 180.0 / M_PI;
24     return result;

```

25 }

A.3.2 Firmware: Local Pin Queue (pin_queue.cpp)

The 32-slot RAM-resident circular buffer that retains unacknowledged pin payloads across BLE disconnections. Enqueue evicts the oldest un-ACK'd entry when the buffer is full; prune runs after every retry cycle to compact the buffer in-place using a two-pointer scan.

```

1 // esp32-firmware/src/pin_queue.cpp (excerpt)
2 static QueuedPin _queue[CFG::MAX_LOCAL_QUEUE];
3 static int      _count = 0;
4
5 bool pinQueue_enqueue(const String& id, const String& json) {
6     if (_count >= CFG::MAX_LOCAL_QUEUE) {
7         Serial.println([QUEUE] full -- dropping oldest un-ACK'd pin);
8         for (int i = 0; i < _count; i++) {
9             if (!_queue[i].acked) {
10                 for (int j = i; j < _count - 1; j++) {
11                     _queue[j] = _queue[j + 1];
12                 }
13                 _count--;
14                 break;
15             }
16         }
17         if (_count >= CFG::MAX_LOCAL_QUEUE) return false;
18     }
19
20     QueuedPin& e = _queue[_count++];
21     e.id = id; e.jsonPayload = json;
22     e.attempts = 0; e.lastSentMs = 0; e.acked = false;
23     return true;
24 }
25
26 void pinQueue_ack(const String& id) {
27     for (int i = 0; i < _count; i++) {
28         if (_queue[i].id == id) { _queue[i].acked = true; return; }
29     }
30 }
31
32 void pinQueue_prune() {
33     // Two-pointer compact: copy survivors down over the gaps
34     int write = 0;
35     for (int read = 0; read < _count; read++) {
36         if (!_queue[read].acked) {
37             if (write != read) _queue[write] = _queue[read];
38             write++;
39         }
40     }
41     _count = write;
42 }

```

A.3.3 Firmware: BLE GATT Server Setup (ble_server.cpp)

The L.I.N.K. custom GATT service is constructed at boot, wiring the Telemetry and Pin Payload notify characteristics together with the writable ACK and Calibration characteristics. The ACK write callback parses the JSON message from the phone and hands the pin ID off to the main loop, which marks the corresponding queue entry as acknowledged.

```

1 // esp32-firmware/src/ble_server.cpp (excerpt)
2 class AckWriteCallback : public BLECharacteristicCallbacks {
3     void onWrite(BLECharacteristic* pChar) override {
4         std::string raw = pChar->getValue();
5         if (raw.length() == 0) return;

```



```

6   JsonDocument doc;
7   DeserializationError err = deserializeJson(doc, raw);
8   if (err) {
9       Serial.printf([BLE] ACK parse error: %s\n, err.c_str());
10      return;
11  }
12  if (doc[ack].as<bool>() && doc.containsKey(pinId)) {
13      String pinId = doc[pinId].as<String>();
14      if (_ackCb) _ackCb(pinId);
15  }
16  }
17  };
18  };
19
20 void ble_init() {
21     BLEDevice::init(cfg::BLE_DEVICE_NAME);
22     BLEDevice::setMTU(512); // request large MTU for JSON
23
24     pServer = BLEDevice::createServer();
25     pServer->setCallbacks(new ServerCallbacks());
26
27     BLEService* pService =
28         pServer->createService(cfg::BLE_SERVICE_UUID);
29
30     pTelemetryChar = pService->createCharacteristic(
31         cfg::BLE_TELEMETRY_CHAR_UUID,
32         BLECharacteristic::PROPERTY_NOTIFY);
33     pTelemetryChar->addDescriptor(new BLE2902());
34
35     pPinChar = pService->createCharacteristic(
36         cfg::BLE_PIN_CHAR_UUID,
37         BLECharacteristic::PROPERTY_NOTIFY);
38     pPinChar->addDescriptor(new BLE2902());
39
40     pAckChar = pService->createCharacteristic(
41         cfg::BLE_ACK_CHAR_UUID,
42         BLECharacteristic::PROPERTY_WRITE |
43         BLECharacteristic::PROPERTY_WRITE_NR);
44     pAckChar->setCallbacks(new AckWriteCallback());
45
46     pService->start();
47     ble_startAdvertising();
48 }

```

A.3.4 PWA: Haversine Forward Projection (geo.ts)

The TypeScript mirror of the firmware's destination-point formula. Both implementations consume identical inputs (observer latitude/longitude in degrees, bearing in degrees, range in meters) and produce identical outputs to within 1×10^{-6} degrees.

```

1 // hud-app/src/lib/geo.ts
2 export function destinationPoint(
3     latDeg: number, lonDeg: number,
4     bearingDeg: number, distanceM: number) {
5     const R = 6371000;
6     const lat1 = (latDeg * Math.PI) / 180;
7     const lon1 = (lonDeg * Math.PI) / 180;
8     const brng = (bearingDeg * Math.PI) / 180;
9     const dr = distanceM / R;
10
11     const lat2 = Math.asin(
12         Math.sin(lat1) * Math.cos(dr) +
13         Math.cos(lat1) * Math.sin(dr) * Math.cos(brng),
14     );
15
16     const lon2 = lon1 + Math.atan2(
17         Math.sin(brng) * Math.sin(dr) * Math.cos(lat1),

```

```

18   Math.cos(dr) - Math.sin(lat1) * Math.sin(lat2),
19   );
20
21   return {
22     lat: (lat2 * 180) / Math.PI,
23     lon: (lon2 * 180) / Math.PI,
24   };
25 }

```

A.3.5 Server: REST Endpoint and Validation (`routes/pins.js`)

The Express handler that closes the end-to-end delivery loop. The INSERT OR IGNORE pattern keyed on the pin `id` makes the endpoint idempotent: a phone retry of an already-stored pin (caused by a missed ACK) silently no-ops instead of producing a duplicate.

```

1  // binocular-hud/server/src/routes/pins.js (excerpt)
2  import { Router } from express;
3  import db from ../db/pool.js;
4  const router = Router();
5
6  const insertPin = db.prepare(`
7    INSERT OR IGNORE INTO pins (
8      id, received_at,
9      observer_lat, observer_lon, observer_alt, observer_acc,
10     target_lat, target_lon, target_alt,
11     bearing_deg, pitch_deg, range_m, lidar_quality,
12     label, telemetry
13   ) VALUES (
14     @id, @receivedAt,
15     @observerLat, @observerLon, @observerAlt, @observerAcc,
16     @targetLat, @targetLon, @targetAlt,
17     @bearingDeg, @pitchDeg, @rangeM, @lidarQuality,
18     @label, @telemetry
19   )
20 `);
21
22 router.post(/, (req, res) => {
23   try {
24     const p = req.body;
25     const errors = [];
26     if (!p.id) errors.push(missing id);
27     if (!p.observer?.lat || !p.observer?.lon)
28       errors.push(missing observer position);
29     if (!p.target?.lat || !p.target?.lon)
30       errors.push(missing target position);
31     if (Math.abs(p.target.lat) > 90)
32       errors.push(target.lat out of range);
33     if (Math.abs(p.target.lon) > 180)
34       errors.push(target.lon out of range);
35
36     if (errors.length) {
37       return res.status(400).json({
38         ok: false, pinId: p.id ?? null,
39         error: errors.join(', '),
40       });
41     }
42
43     insertPin.run({
44       id: p.id,
45       receivedAt: p.receivedAt ?? Date.now(),
46       observerLat: p.observer.lat,
47       observerLon: p.observer.lon,
48       observerAlt: p.observer.altM ?? null,
49       observerAcc: p.observer.accM ?? null,
50       targetLat: p.target.lat,
51       targetLon: p.target.lon,
52       targetAlt: p.target.altEstM ?? null,

```

```

53     bearingDeg: p.aiming?.bearingDeg ?? null,
54     pitchDeg:   p.aiming?.pitchDeg ?? null,
55     rangeM:     p.aiming?.rangeM ?? null,
56     lidarQuality: p.aiming?.lidarQuality ?? null,
57     label: p.label ?? null,
58     telemetry: p.telemetry
59               ? JSON.stringify(p.telemetry) : null,
60   });
61
62   return res.json({ ok: true, pinId: p.id });
63 } catch (err) {
64   return res.status(500).json({
65     ok: false, pinId: req.body?.id ?? null,
66     error: internal server error,
67   });
68 }
69 });

```

D ChatGPT Prompts and Outcomes

1a - 1g:

[1a] **Voltage Regulator Comparison Prompt:** “Explain a technical comparison between multiple voltage regulators for an embedded battery-powered system. The comparison should evaluate efficiency, cost, footprint, complexity, and suitability for a compact handheld device.”

[1b] **ChatGPT Outcome:** “Voltage regulators are a critical component in embedded battery-powered systems, as they ensure stable voltage delivery under varying load and battery conditions. Linear regulators offer low cost and small footprint but suffer from poor efficiency due to heat dissipation. Buck regulators provide high efficiency for step-down applications and are well suited for digital subsystems, though they introduce switching noise and increased design complexity. Boost regulators enable voltage step-up but require higher input current and careful design considerations. Buck-boost regulators provide the most flexible solution by maintaining a constant output across varying input voltages, making them ideal for single-cell battery systems. In compact handheld devices, a hybrid approach using switching regulators for primary power conversion and linear regulators for noise-sensitive components is often optimal.”

[1c] **ChatGPT Conversation Link:** <https://chatgpt.com/share/69c2c1c2-fc90-832f-ad0b-c728db936307>

[1d] **Claude Outcome:** “The comparison of voltage regulators highlights trade-offs between efficiency, cost, and design complexity. Linear regulators provide low cost and minimal footprint but suffer from poor efficiency due to heat dissipation. Buck regulators offer high efficiency for step-down applications and are well suited for powering digital subsystems, though they introduce switching noise and require additional components. Boost regulators enable voltage step-up but increase input current demand and design complexity. Buck-boost regulators provide the most flexible solution by maintaining a stable output voltage across varying input conditions, making them ideal for battery-powered handheld systems.”

[1e] **Claude Conversation Link:**

<https://claude.ai/share/7cb880d3-44cd-4107-b66d-c7bd0e32ac97>

[1f] ***Gemini Prompt:*** “Explain a technical comparison between multiple voltage regulators for an embedded battery-powered system. Evaluate efficiency, cost, footprint, complexity, and suitability for a compact handheld device.”

[1g] ***Gemini Outcome:*** “Voltage regulators for embedded battery-powered systems present trade-offs across efficiency, cost, footprint, and complexity. Linear regulators provide a compact and low-cost solution with low noise, but suffer from poor efficiency due to energy dissipation as heat. Buck regulators offer high efficiency for step-down applications and are well suited for digital systems, though they require additional components and careful layout. Boost regulators enable voltage step-up but increase input current demand and design complexity. Buck-boost regulators provide the most versatile solution by maintaining stable output voltage across varying battery conditions, making them highly suitable for compact handheld devices.”

[1h] ***Gemini Conversation Link:***

<https://gemini.google.com/share/fc6568bb013c>

2a - 2g:

[2a] ***Types of LiDAR for Long Distance Ranging Prompt:*** “Provide an in depth comparison between different LiDAR types for long distance ranging. Ensure that range, complexity, cost, and any other relevant factors are accounted for.”

[2b] ***ChatGPT Outcome:*** “Long-distance LiDAR performance is fundamentally dictated by the ranging principle used—primarily Time-of-Flight (ToF), phase/amplitude-modulated continuous wave (often grouped as AMCW/phase-shift), and Frequency-Modulated Continuous Wave (FMCW)—each of which trades off range capability, system complexity, cost, and robustness in different ways.

Time-of-Flight (ToF) LiDAR is the most widely deployed architecture for long-range applications because it directly measures the round-trip travel time of short laser pulses, making it conceptually simple, scalable, and relatively low cost to implement. Its hardware typically consists of pulsed laser diodes, fast photodetectors (APDs/SPADs), and timing electronics, which are mature and manufacturable at scale. As a result, ToF systems dominate automotive and surveying applications where ranges on the order of hundreds of meters to beyond 1 km are achievable with sufficient aperture and pulse energy. However, the achievable range is fundamentally constrained by eye-safety limits on peak optical power and by signal attenuation, meaning that pushing to extreme distances requires large optics or sophisticated scanning strategies. Additionally, ToF systems are susceptible to noise from ambient light and multi-path reflections, and they cannot directly measure velocity, requiring frame-to-frame estimation instead. Despite these drawbacks, their low complexity and cost-effectiveness make them the default choice for long-range ranging when system budgets are constrained.

Phase-shift or AMCW LiDAR operates by modulating the intensity of a continuous laser beam and measuring the phase difference between emitted and received signals. This approach simplifies timing requirements compared to pulsed ToF and enables compact electronics, but it introduces ambiguity because phase wraps periodically, limiting the maximum unambiguous range unless multiple modulation frequencies are used. As a result, AMCW systems are typically restricted to short-to-medium distances and are rarely used for kilometer-scale ranging without complex multi-frequency architectures. While they can achieve high precision at shorter ranges and benefit from continuous-wave operation (which can improve signal averaging and reduce peak power requirements), their need for multi-frequency disambiguation and susceptibility to phase noise increases system complexity without delivering the long-range performance of pulsed ToF or FMCW. Consequently, for long-distance applications, AMCW is generally considered less favorable despite moderate cost and integration advantages.

FMCW LiDAR represents a more advanced coherent detection approach in which a continuously emitted laser is frequency-chirped, and distance is extracted from the beat frequency between transmitted and received signals. This coherent detection provides several intrinsic advantages for long-range operation, including higher sensitivity, better rejection of ambient light, and the ability to operate at higher effective optical powers without violating eye-safety limits due to its distributed energy over time. As a result, FMCW systems can achieve excellent range resolution and improved performance in low-reflectivity or high-glare environments, while also directly measuring radial velocity via the Doppler effect. However, these benefits come at the cost of significantly increased system complexity: FMCW requires narrow-linewidth lasers, coherent mixing, precise chirp linearity, and heavy digital signal processing. These requirements historically made FMCW expensive and difficult to scale, although recent advances in silicon photonics integration are beginning to reduce size and cost. Even so, manufacturing challenges and system design complexity still limit widespread adoption compared to ToF.

When comparing these technologies specifically for long-distance ranging, ToF remains the most practical and widely used solution due to its balance of achievable range (hundreds of meters to kilometer scale), relatively low complexity, and cost efficiency. FMCW, while theoretically superior in sensitivity, noise rejection, and velocity measurement, is still emerging and tends to be more expensive and complex, though it shows strong potential for future high-performance systems where cost constraints are relaxed. Phase-shift/AMCW systems, by contrast, are generally unsuitable for true long-range applications due to ambiguity limits and reduced maximum range, despite offering compact and moderately priced implementations. Overall, the trade space is dominated by a classic engineering compromise: ToF offers the best balance of range and practicality today, FMCW offers the highest performance ceiling at the expense of complexity and cost, and phase-based systems occupy a niche where moderate range and compact design are prioritized over extreme distance capability.”

[2c] **ChatGPT Conversation Link:** https://chatgpt.com/s/t_69c616d6f45c8191879e9b0e0ba61bad

[2d] **Gemini Outcome:** “Long-distance LiDAR for ranging beyond 200 meters essentially splits into two competing detection philosophies: Time-of-Flight (ToF) and Frequency Modulated Continuous Wave (FMCW). ToF remains the industry workhorse because of its relatively low computational complexity and established manufacturing base. In a ToF system, a laser pulse is

emitted, and the distance d is calculated by measuring the round-trip time Δt using the formula $d = \frac{c\Delta t}{2}$, where c is the speed of light. While effective, ToF systems face a significant trade-off between range and eye safety. Operating at the 905 nm wavelength allows for cheap silicon-based sensors, but the power must be kept low to avoid retinal damage. To push past 200 meters, many high-end ToF systems shift to the 1550 nm wavelength, which is "eye-safe" at much higher power levels because the light is absorbed by the cornea before reaching the retina. However, 1550 nm systems require expensive Indium Gallium Arsenide (InGaAs) detectors, which significantly drives up the cost compared to their 905 nm counterparts.

In contrast, FMCW LiDAR, often marketed as 4D LiDAR, offers a more technically sophisticated approach by emitting a continuous stream of light rather than discrete pulses. By modulating the frequency of the emitted light—usually in a "chirp" pattern—the system can determine range and velocity simultaneously through coherent detection. The range is derived from the beat frequency f_b generated when the reflected light is mixed with the local oscillator, while the velocity v is captured directly via the Doppler shift using the relationship $f_D = \frac{2v f_c}{c}$, where f_c is the carrier frequency. This method provides several advantages for long-range applications, including a much higher sensitivity that allows for long-distance detection with lower peak power and a natural immunity to interference from the sun or other LiDAR sensors. Despite these perks, FMCW is significantly more complex to implement. It requires ultra-stable lasers with long coherence lengths and heavy-duty digital signal processing to handle the fast Fourier transforms needed for every pixel, which has historically kept its cost high and its pixel throughput lower than that of high-speed ToF systems.

The mechanism used to steer the laser beam adds another layer of complexity and cost to the comparison. Mechanical LiDAR, characterized by the iconic "spinning bucket" on top of early autonomous vehicles, remains the gold standard for long-range performance and a full 360-degree field of view. By physically rotating the entire laser and receiver assembly, these systems maintain a consistent beam quality across the entire scan, avoiding the signal degradation often seen at the edges of stationary sensors. However, the reliance on moving parts introduces mechanical wear, increases the physical footprint, and keeps unit costs in the thousands of dollars. Micro-Electro-Mechanical Systems (MEMS) offer a "semi-solid-state" compromise, using tiny vibrating mirrors to steer the beam. While MEMS units are smaller and cheaper, they often struggle with long-range detection due to the small size of the mirrors, which limits the amount of reflected light they can collect and makes them sensitive to vehicle vibrations.

Purely solid-state technologies, such as Optical Phased Arrays (OPA) and Flash LiDAR, represent the current frontier for reliability and mass-market scalability. Flash LiDAR functions like a high-speed camera with a flash, illuminating the entire scene at once. While this provides exceptional frame rates and no moving parts, the light energy dissipates rapidly over distance, making it generally unsuitable for long-range highway speeds unless paired with extremely high-power 1550 nm lasers. OPA, on the other hand, uses electronic phase shifts to steer the beam without any moving parts at all. While theoretically the ultimate solution for reliability and cost-efficiency through silicon photonics, OPA systems in 2026 still grapple with "sidelobes"—wasted light energy that bleeds out of the main beam—which reduces the effective range and resolution compared to mechanical or high-end ToF systems.

When evaluating these types for long-distance use, the decision ultimately hinges on a balance

between "truth" and "economy." Mechanical ToF systems provide the most reliable data for high-precision mapping and long-range detection but are becoming less attractive for consumer automotive due to their bulk and maintenance needs. FMCW is the rising star for long-distance perception because it adds the "fourth dimension" of instantaneous velocity, which is critical for predicting the movement of distant objects, but its price tag and processing requirements remain barriers to entry. For 2026 and beyond, the market is shifting toward hybrid approaches: 1550 nm ToF for the raw range and resolution required for Level 3 autonomy, and increasingly integrated FMCW on silicon chips for high-performance safety applications. While we aren't quite at the point of a \$100 long-range LiDAR, the move away from mechanical parts toward integrated photonics is rapidly closing the gap between high-end performance and mass-market affordability.

Would you like me to analyze the specific LiDAR specifications of a leading manufacturer, such as Hesai or Innoviz, to see how these theories are being applied in the field today?"

[2e] **Gemini Conversation Link:** <https://gemini.google.com/share/eb37ab483c12>

[2f] **Copilot Outcome:** "Different LiDAR architectures for long-distance ranging—most commonly time-of-flight (ToF) pulsed LiDAR, frequency-modulated continuous-wave (FMCW) LiDAR, and hybrid or application-specific variants such as terrestrial long-range scanners—differ substantially in achievable range, system complexity, cost, and operational constraints. Pulsed ToF LiDAR remains the most widely used for long-range applications because high-energy pulses can reach hundreds of meters with relatively straightforward signal processing, though performance depends heavily on beam divergence, detector sensitivity, and environmental conditions. These systems tend to scale in cost with required range, as higher-power lasers and more sensitive detectors are needed for long-distance returns, but they remain simpler and more mature than other approaches. FMCW LiDAR, by contrast, offers coherent detection, enabling superior velocity measurement, improved noise rejection, and potentially longer effective range for a given optical power; however, FMCW systems are significantly more complex due to the need for narrow-linewidth lasers, stable frequency chirping, and coherent receivers, which increases cost and limits commercial availability. Terrestrial long-range LiDAR scanners used in surveying can reach hundreds of meters to over a kilometer, but they often trade field-of-view coverage and acquisition speed for extreme range, requiring mechanical scanning and careful setup. These systems also vary widely in cost, with high-end long-range terrestrial scanners commanding premium prices due to precision optics, rotation mechanisms, and calibration requirements. Across all types, range performance is tightly linked to optical power, atmospheric conditions, and scanning geometry, while complexity and cost scale with the sophistication of the detection scheme and mechanical design. Selecting the optimal long-range LiDAR therefore requires balancing measurement distance, accuracy, field-of-view needs, environmental robustness, and budget, as emphasized in long-range LiDAR selection guides that highlight range, accuracy, acquisition speed, and operating environment as key decision factors."

[2g] **Copilot Conversation Link:** <https://copilot.microsoft.com/shares/i5hVdptVA4vYbDuSvF8Nz>

3a - 3g:

[3a] **Type of Prism for binoculars Prompt:** “Give me a list and in depth comparison of different prism configuration for optical binoculars. Take into account cost, weight, and any optical parameters you think are necessary when picking a prism type.”

[3b] **ChatGPT Outcome:** “Binocular optical systems use prisms to erect the inverted image produced by the objective lens and to fold the optical path so the instrument can remain compact. The prism configuration strongly influences size, weight, optical transmission, cost, alignment tolerance, and overall performance. Most binocular designs fall into two broad families: Porro prism systems and roof prism systems. Roof prisms are further divided into Schmidt–Pechan and Abbe–Koenig configurations, with a few uncommon variants occasionally appearing in specialized optical instruments.

The classic Porro prism configuration uses two right-angle prisms arranged so the optical axis is laterally displaced between the objective and eyepiece. Light entering the system undergoes multiple reflections within the prisms, which rotate the image by 180 degrees while also shifting the beam sideways. This geometry produces the familiar zig-zag optical path seen in traditional binoculars. Because most reflections occur through total internal reflection rather than mirror coatings, Porro systems achieve very high optical transmission. Another advantage is stereoscopic depth perception. The objective lenses are spaced farther apart than the eyepieces, increasing the effective baseline between the two optical channels and improving depth cues when viewing distant objects. Porro binoculars also often provide a relatively wide field of view and high image brightness. However, the lateral displacement of the optical path results in a bulkier mechanical layout. The housings are wider, the prism blocks are larger, and the overall instrument tends to be heavier. Waterproofing can also be more difficult compared to roof prism designs because of the more complex mechanical structure. Manufacturing cost is generally low, since Porro prisms are simple to produce and do not require phase-correction coatings.

A variation known as the reverse Porro prism inverts the typical geometry so the objectives are closer together than the eyepieces. This arrangement reduces the lateral footprint of the instrument and allows very compact binocular bodies. Reverse Porro designs are common in inexpensive compact binoculars and travel optics. The optical performance remains similar to the traditional Porro concept, including high transmission due to total internal reflection. However, the reduced spacing between objectives decreases the stereoscopic effect and therefore reduces perceived depth. In practice, reverse Porro binoculars are typically built for portability rather than maximum optical performance.

Roof prism binoculars represent a different optical architecture in which the objective and eyepiece remain aligned along a straight optical axis. This produces the narrow, streamlined “H-shaped” form factor common in modern field binoculars. Roof prisms achieve image erection by reflecting the beam across a pair of intersecting surfaces known as the roof edge. This geometry splits the incoming wavefront into two halves that recombine after reflection. Because the two halves travel slightly different optical paths, a phase shift occurs between them, which can reduce contrast and resolution. Modern roof prism binoculars therefore require phase-correcting coatings to restore the wavefront alignment. The need for extremely precise prism geometry and specialized coatings makes roof prism binoculars more expensive to manufacture than Porro systems, but the compact mechanical layout and improved durability make them very popular.

The Schmidt–Pechan roof prism is the most common roof prism configuration used in binoculars today. This system consists of two prism elements that produce multiple internal reflections, folding the optical path tightly while rotating the image 180 degrees. The design is highly compact, allowing manufacturers to produce shorter and lighter binocular bodies. The geometry does require reflective mirror coatings on some prism surfaces because total internal reflection cannot occur at every interface. These coatings introduce small optical losses, so Schmidt–Pechan systems generally have slightly lower transmission than Porro or Abbe–Koenig prisms. Despite this limitation, the compactness and mechanical robustness of the Schmidt–Pechan design make it the dominant configuration in most modern consumer binoculars used for birding, hiking, and general outdoor observation.

The Abbe–Koenig roof prism represents a higher-performance but less compact roof prism design. This configuration uses two elongated prism elements arranged so that light undergoes four reflections before exiting the system. Most of these reflections occur through total internal reflection, allowing the system to maintain very high optical transmission without requiring mirror coatings. Because of this efficiency, Abbe–Koenig prisms are often used in binoculars designed for low-light conditions such as hunting or twilight observation. However, the geometry requires a longer optical path, which leads to longer binocular housings and larger prism assemblies. Manufacturing these prisms is also more complex and expensive than Schmidt–Pechan designs, so they are typically found only in premium binocular models.

Several other prism systems exist but are rarely used in binoculars. The Amici prism, sometimes called a roof-edge erecting prism, produces a correctly oriented image with proper left-right orientation. It is commonly used in telescope diagonals and spotting scopes rather than binoculars. The roof edge can introduce diffraction artifacts that reduce image quality at high magnifications, which limits its use in binocular designs. Another historical design is the Uppendahl prism, which also belongs to the roof prism family. It provides an erect image and compact optical path but requires complicated internal geometry and precise manufacturing. Because of this complexity and cost, it has largely been replaced by the Schmidt–Pechan configuration in modern binocular manufacturing.

When selecting a prism configuration for binocular design, engineers typically evaluate several competing parameters. Optical throughput is determined by the number of reflections and whether reflective coatings are required. Porro and Abbe–Koenig systems generally provide the highest transmission because most reflections occur through total internal reflection. Size and packaging constraints often favor roof prism systems, since they allow the objective and eyepiece to remain coaxial and produce a slimmer housing. Manufacturing cost depends strongly on prism geometry and required coatings. Porro prisms are the least expensive to produce, while Abbe–Koenig prisms are typically the most costly. Mechanical alignment tolerances also play an important role. Roof prisms require extremely precise roof angles and phase-correcting coatings to avoid degradation of image contrast. Finally, ergonomic considerations often influence design choices, since compact roof prism binoculars are easier to carry and more resistant to environmental exposure.

In practical engineering applications, three configurations dominate the binocular market. Porro prisms offer the best optical performance for the lowest manufacturing cost and are frequently used in astronomical or marine binoculars where size is less critical. Schmidt–Pechan roof prisms

provide compact size and ruggedness, making them the most common choice for consumer field binoculars. Abbe–Koenig prisms deliver the highest optical transmission among roof prism systems and are typically used in premium binoculars intended for low-light observation. Together, these three prism architectures represent the primary optical design options available for modern binocular systems. ”

[3c] **ChatGPT Conversation Link:** <https://chatgpt.com/share/69c6079f-1e5c-8333-85e9-d9ed76898578>

[3d] **Gemini Outcome:** “The most traditional configuration is the Porro prism, which uses a pair of right-angle prisms offset from one another to flip the image. This design is highly efficient because it utilizes total internal reflection, meaning no light is lost to mirror surfaces, resulting in a very bright image with excellent 3D depth perception. However, the offset lenses make Porro prism binoculars bulky and heavy, which can be a drawback for long-distance trekking. Because they are easier and cheaper to manufacture to high optical standards, they offer the best performance-to-cost ratio for budget-conscious users.

The Schmidt-Pechan prism is the most common type of roof prism and is responsible for the modern, straight-barrel shape of many popular binoculars. While these are much more compact and lightweight than Porro designs, they are technically more complex. One of the glass surfaces does not naturally reflect all light, so it must be treated with expensive mirror coatings, such as silver or dielectric layers, to maintain brightness. They also require phase-correction coatings to keep light waves in sync. Because of these requirements, a high-quality Schmidt-Pechan binocular is usually more expensive than a comparable Porro model.

For those seeking the highest possible light transmission in a roof-prism format, the Abbe-Koenig prism is the premium choice. Unlike the Schmidt-Pechan, the Abbe-Koenig design uses total internal reflection on all surfaces, which eliminates the need for lossy mirror coatings and results in a noticeably brighter image, especially in low-light conditions like dawn or dusk. The trade-off is physical size; the prism itself is quite long, which leads to longer and heavier binocular barrels. Due to the difficulty of manufacturing these to precise tolerances, they are typically found only in top-tier, professional optics.

A more modern and specialized hybrid is the Perger-Porro prism, which is found in high-end luxury optics. This configuration attempts to bridge the gap between designs by offering the superior depth perception and light efficiency of a Porro prism while maintaining a more ergonomic, curved shape that fits comfortably in the hand. While it provides an exceptional viewing experience, it is currently one of the most expensive prism types on the market. When choosing between these types, the decision usually comes down to whether you prioritize the compact portability of a Schmidt-Pechan or the optical efficiency and value found in Porro or Abbe-Koenig designs.

Would you like me to find specific binocular models that use these different prism types so you can compare their prices?”

[3g] **Copilot Conversation Link:** <https://m365.cloud.microsoft/chat/entity1-d870f6cd-4aa5-4d42-9626-ab690c04eyJpZCI6IlZYTmxjbFl4ZkdoMGRIQnpPaTh2YzNWaWMzUnlZWfJsTFdsdWRDNXZabVpwWTJVdVkyOXF>

[3d] **Copilot Outcome:** ” Porro prisms use an offset, zig-zag light path created by two right-angle prisms. This design produces the classic wide-bodied binocular shape and offers outstanding optical performance. Porro systems deliver very high brightness because their simpler internal reflections minimize light loss, resulting in strong image illumination even without advanced coatings. They also offer exceptional stereoscopic depth perception due to the wider spacing of the objective lenses, typically around four to six inches in common 42 mm models. This creates a pronounced three-dimensional effect that is ideal for birdwatching, field identification, and astronomy. Such prisms maintain strong contrast and sharpness without the need for phase-correction coatings, contributing to their reputation for excellent optical quality. However, Porro configurations tend to be bulkier and heavier than roof-prism alternatives, and their offset mechanical structure makes them more challenging to waterproof and less durable in rugged environments. They also benefit from lower manufacturing complexity, making them the most affordable option while often providing the best optical performance per dollar. [

revellphot...graphy.com

](<https://revellphotography.com/roof-prism-vs-porro-prism/>), [

foreseenoptics.com

](<https://www.foreseenoptics.com/roof-prism-vs-porro-prism-binoculars-what-you-should-know>), [

opticsboom.com

](<https://opticsboom.com/porro-prism-vs-roof-prism-binoculars-full-comparison/>)

Schmidt–Pechan roof prisms use a straight-through optical layout with six internal reflections, some of which require mirror coatings to maintain adequate brightness. These roof prisms produce compact, lightweight binoculars that are easy to carry and ideal for hiking, travel, and general outdoor use. Their slim barrels also improve durability because the straight-line design is easier to waterproof and mechanically reinforce. While they provide good optical performance, Schmidt–Pechan systems suffer from lower inherent light transmission than Porro or Abbe–König prisms. They depend heavily on advanced phase-correction and dielectric coatings to maintain contrast and image sharpness, as the roof edge introduces phase-shift problems that degrade detail without corrective coatings. The high precision required in their manufacturing increases production costs, making Schmidt–Pechan binoculars generally more expensive while remaining the most portable option. [

vectoroptics.com

](<https://www.vectoroptics.com/Academy/AK-PRISM-VS-ROOF-PRISM-BINOCULAR-WHICH-ONE-TO-CHOOSE.html>), [

opticsboom.com

](<https://opticsboom.com/porro-prism-vs-roof-prism-binoculars-full-comparison/>), [

ico – optics.org

](<https://www.ico-optics.org/the-physics-of-porro-vs-roof-prism-configurations/>)

Abbe–König roof prisms also use a straight-line optical path but differ from Schmidt–Pechan

designs because all reflections occur through total internal reflection. This eliminates the need for mirror coatings and allows Abbe–König prisms to achieve the highest light transmission of any roof-prism system, reaching values around ninety-three percent. As a result, Abbe–König binoculars are particularly effective in low-light environments such as twilight hunting or night-time wildlife observation. They offer excellent brightness, sharpness, and contrast while still benefiting from the durability and waterproofing advantages of roof-prism construction. However, these prisms are physically longer than Schmidt–Pechan prisms, resulting in longer barrel designs and moderate increases in weight. Because Abbe–König prisms require extremely precise manufacturing and are usually paired with other high-end optical elements, they appear almost exclusively in premium binoculars and carry a high price. [

vectoroptics.com

](<https://www.vectoroptics.com/Academy/AK-PRISM-VS-ROOF-PRISM-BINOCULAR-WHICH-ONE-TO-CHOOSE.html>)

Choosing between these prism configurations depends on your priorities. Porro prisms provide the best depth perception and raw optical brightness at the lowest cost, but their bulky size and less rugged construction limit portability. Schmidt–Pechan roof prisms are the most compact and lightweight, offering excellent versatility for mobile outdoor use, though their optical performance depends heavily on expensive coatings and precise alignment. Abbe–König roof prisms occupy the high-end segment, providing the brightest roof-prism images with superior low-light performance while maintaining durability, though at the cost of longer barrels and significantly higher pricing. Each configuration excels in specific scenarios, and understanding these differences helps determine the ideal prism type for activities such as astronomy, birding, hunting, travel, or general observation. ”

[3g] **Copilot Conversation Link:** https://m365.cloud.microsoft/chat/conversation/c869e7c9-8dcf-4813-ae5d-de4a-auth=2&home=1&from=ShellLogo&username=je661406%40ucf.edu&login_hint=je661406%40ucf.edu

4a - 4g:

[4a] **Cooperative Loop vs. FreeRTOS Tasks Prompt:** “What are the trade-offs between using a cooperative timer-based main loop versus FreeRTOS tasks for polling multiple I2C sensors at 20 Hz on an ESP32-S3, considering real-time constraints, power consumption, code complexity, and watchdog timer management?”

[4b] **ChatGPT Outcome:** “One practical point already: at 20 Hz you are nowhere near the ESP32-S3’s timer limits, so the decision is less about raw timing capability and more about isolation, sleep behavior, and how you want failures to be contained.

For multiple I2C sensors at 20 Hz on one ESP32-S3, I would usually not start with ‘one FreeRTOS task per sensor.’ I’d start with either a cooperative periodic poller or a single periodic FreeRTOS task that owns the bus. At 20 Hz, your period is 50 ms, and Espressif’s esp_timer periodic timers are meant for much shorter intervals than that; the key caveat is that timer callbacks run from the esp_timer task and should be kept short. That makes a timer-triggered poller a good fit, but the actual I2C reads should happen in a task, not inside the timer callback.

Real-time behavior: a cooperative poller gives you the most predictable sequencing because you decide exactly when each sensor is read and in what order. The downside is that one slow or stuck transaction can delay the whole cycle. FreeRTOS tasks give better isolation and prioritization, but on a shared I2C bus they do not create real parallel sensor I/O, because I2C itself is serial and half-duplex. On the ESP32-S3 you have two I2C controllers, so tasks only buy real bus-level parallelism if you actually split devices across two controllers. Otherwise, tasks mostly add scheduler jitter and locking overhead around the same single bus.

Power consumption: both approaches can be efficient if they block properly between samples. ESP-IDF's auto light-sleep uses FreeRTOS tickless idle and only enters sleep when tasks are blocked/suspended and power-management locks are released. So a cooperative loop that sleeps until the next period, or FreeRTOS tasks that wait on delays/notifications, can both be low power. The power loss comes from busy polling, not from the label 'main loop' versus 'RTOS.' Also, ESP-IDF power management can add interrupt latency, but Espressif documents the worst extra latency as up to 40 μ s, which is tiny relative to a 50 ms sample period.

Code complexity: the cooperative design wins easily. One scheduler, one bus owner, one timing source, fewer stacks, fewer race conditions, and simpler debugging. FreeRTOS becomes worth it when sensors have different rates, different timeout/fault behavior, or you also have unrelated work like networking, logging, UI, or control loops that should not be blocked by sensor polling. On ESP32-S3, IDF FreeRTOS is dual-core SMP, and tasks can be pinned to Core 0, Core 1, or left unpinned. That helps when you need isolation, but it also means you now have to think about affinity, priority inversions, and cross-core behavior.

Watchdog management: this is where task-per-sensor designs often get messier. Espressif's Task Watchdog Timer primarily watches the idle tasks and detects tasks that run too long without yielding; you can also explicitly subscribe tasks or 'users' and reset the watchdog from them. In a cooperative poller, watchdog handling is straightforward: each cycle either completes, times out, or trips a recovery path in one place. In a multi-task design, every polling task has to block or yield correctly, and every path that can stall on I2C or retries has to be bounded. Espressif also notes that a task waiting on a notification can block without consuming CPU time, which is useful if you do choose an RTOS design.

My practical recommendation for your exact case: Use one periodic sensor-poll task and keep it cooperative. Trigger it with either `vTaskDelayUntil()` or a short `esp_timer` callback that just posts a flag/notification. Let one task own the I2C bus. Add FreeRTOS per-sensor tasks only if you have a real need for fault isolation, mixed sample rates, or parallelism across the ESP32-S3's second core or second I2C controller. On a shared bus at 20 Hz, 'many tasks' is usually more complexity than benefit."

[4c] **ChatGPT Conversation Link:** <https://chatgpt.com/share/69c5592c-4bdc-832c-88dd-28f95fd2f1b7>

[4d] **Gemini Outcome:** "When deciding how to poll multiple I2C sensors at 20 Hz (a 50 ms period) on an ESP32-S3, you are fundamentally choosing between simplicity and scalability. It is important to remember that the ESP32-S3 natively runs on FreeRTOS via the ESP-IDF (even if you are using the Arduino core, your `main loop()` is just running inside a FreeRTOS task).

Attempting to force a bare-metal cooperative loop onto an RTOS-based chip often fights the underlying architecture.

1. **Real-Time Constraints (Timing & Jitter):** At 20 Hz, you have a 50 ms window to read all sensors, process the data, and handle other system tasks. Cooperative Main Loop (Superloop): High vulnerability to jitter. If one sensor takes slightly longer to respond, or if you use a blocking I2C call (e.g., waiting for an ACK), the entire loop is delayed. FreeRTOS Tasks: Excellent deterministic timing. You can use `vTaskDelayUntil()` to guarantee the task wakes up exactly every 50 ms, regardless of how long the previous read took (as long as it took less than 50 ms). The ESP32-S3 is dual-core. FreeRTOS allows you to pin the I2C polling task to Core 1 while Core 0 handles Wi-Fi/Bluetooth, completely isolating your sensor timing from heavy network interrupts.

2. **Power Consumption:** Cooperative Main Loop: Generally worse for power unless meticulously managed. To keep precise timing in a loop, developers often use busy-waiting (e.g., `while(millis() - lastTime < 50)`). This keeps the CPU awake and drawing power. FreeRTOS Tasks: Built-in power efficiency. When an ESP-IDF I2C read is called, the RTOS puts the calling task in a 'Blocked' state while the I2C peripheral hardware (via DMA/interrupts) does the work. If you enable Tickless Idle in menuconfig, FreeRTOS will automatically put the ESP32-S3 into automatic light sleep when all tasks are blocked, drastically reducing power without writing any manual sleep logic.

3. **Code Complexity:** Cooperative Main Loop: Easy to set up initially, but scales poorly. Polling one sensor with a timer is trivial. Polling multiple I2C sensors on the same bus cooperatively requires writing a massive state machine to ensure you don't start a read while another is finishing. FreeRTOS Tasks: Higher upfront complexity. You must learn RTOS primitives. Because multiple sensors likely share the same I2C bus, you cannot have multiple tasks writing to the bus simultaneously. You must implement a Mutex (`xSemaphoreCreateMutex()`) to lock the I2C bus during a transaction. However, once the Mutex and Task architecture is set up, adding a new sensor is as simple as spawning a new task or adding a clean sequential block within a single sensor task.

4. **Watchdog Timer (WDT) Management:** Cooperative Main Loop: Risky. The ESP32 utilizes a Task Watchdog Timer (TWDT). If your main loop uses a blocking I2C call that hangs (e.g., a sensor pulls the SDA line low and freezes), the loop gets stuck, the TWDT is not fed, and the ESP32 resets. FreeRTOS Tasks: Safer and more granular. Because I2C calls in the ESP-IDF are RTOS-aware, a blocked I2C call yields to the scheduler. You can set explicit hardware timeouts on the I2C transaction itself. If the sensor hangs, the RTOS unblocks the task, returns an `ESP_ERR_TIMEOUT`, and you can gracefully reset the I2C bus without crashing the entire microcontroller.

The Verdict: For the ESP32-S3, FreeRTOS tasks are the objectively better choice. The hardware and the underlying ESP-IDF are explicitly designed to leverage the RTOS. Writing a cooperative loop on an ESP32 is essentially fighting the framework and leaving the dual-core and power-saving capabilities on the table."

[4e] **Gemini Conversation Link:** <https://gemini.google.com/share/2d842d9d8755>

[4f] **Claude Outcome:** “This is a great embedded systems architecture question. Here’s a comprehensive breakdown of the trade-offs for your ESP32-S3 scenario:

1. Real-Time Constraints: Cooperative Timer-Based Main Loop: Determinism — You tightly control timing (e.g., `esp_timer_get_time()` to poll every 50 ms). Since tasks yield voluntarily, the execution order is fully predictable. Jitter — Minimal as long as each loop iteration completes well within the 50 ms budget. But if any sensor read blocks (e.g., clock stretching on I2C), all other sensors are delayed. Worst case — A single slow sensor stalls everything. FreeRTOS Tasks: Determinism — The scheduler handles preemption. Each sensor can use `vTaskDelayUntil()` for precise periodic wakeups. Jitter — Better isolation; if one sensor blocks in its task, others continue on schedule (assuming separate tasks per sensor or sensor group). Worst case — I2C bus contention still matters (it’s a shared bus), but the software scheduling is more resilient. Verdict: FreeRTOS wins if sensors have variable or unpredictable response times. Cooperative wins if all reads are fast and predictable.

2. Power Consumption: Cooperative Timer-Based Main Loop: Easier sleep integration — You can poll, then explicitly enter light sleep until the next 50 ms window. Less overhead — No context switching, no scheduler ticks consuming power. Deep sleep friendly — Straightforward to implement wake-poll-sleep cycles. FreeRTOS Tasks: Idle task overhead — The FreeRTOS tick interrupt (typically 1 kHz on ESP-IDF) keeps firing, preventing the deepest sleep states unless you enable tickless idle (`configUSE_TICKLESS_IDLE`). Context switching cost — Minor but nonzero; each task switch has a small CPU/power cost. Possible with tuning — ESP-IDF supports automatic light sleep in FreeRTOS mode, but it requires careful configuration. Verdict: Cooperative loop is simpler and more power-efficient by default. FreeRTOS can match it with tickless idle, but adds configuration complexity.

3. Code Complexity: Cooperative Timer-Based Main Loop: Pros — Single execution context, no mutexes, no priority inversion, trivial to debug. Cons — Becomes unwieldy with many asynchronous tasks (networking, display, logging). State machines can proliferate. (Included a concrete code example using `esp_timer_get_time()` with sequential sensor reads in approximately 10 lines.) FreeRTOS Tasks: Pros — Modular, scales well, each sensor is self-contained, natural fit for concurrent operations (WiFi, BLE, etc.). Cons — Must manage mutexes for I2C bus, stack sizing per task (approximately 4 KB each adds up), potential for priority inversion and deadlocks. (Included a concrete code example using `xSemaphoreCreateMutex()` and `xTaskCreatePinnedToCore()`.) Verdict: Cooperative is simpler for 5 or fewer sensors with no other concurrent work. FreeRTOS scales better when you add networking, logging, or UI.

4. Watchdog Timer Management: Cooperative Timer-Based Main Loop: Simple — One `esp_task_wdt_reset()` call in the loop. If anything hangs, the watchdog fires; easy to detect and debug. Risk — A single blocking I2C read kills the whole system, but that’s also easy to diagnose. FreeRTOS Tasks: Per-task watchdog — ESP-IDF lets you subscribe individual tasks to the Task Watchdog Timer (TWDT) via `esp_task_wdt_add()`. Idle task pitfall — By default, ESP-IDF monitors the idle task. If your tasks block but the idle task still runs, the watchdog won’t fire; a hung sensor task may go unnoticed. Best practice — Explicitly add each sensor task to the TWDT. Verdict: Cooperative is simpler. FreeRTOS requires explicit per-task watchdog registration, but gives finer-grained monitoring.

Also provided a summary comparison table covering: real-time (20 Hz), power consumption, code complexity, watchdog management, I2C bus sharing, failure isolation, and RAM usage.

Recommendation for Your Case: For 20 Hz polling of multiple I2C sensors on ESP32-S3: Start with cooperative if your system is sensor-polling focused, power-sensitive, and doesn't run WiFi/BLE concurrently. Move to FreeRTOS tasks if you need concurrent networking, variable sensor latencies, or plan to scale the system. Consider a hybrid — Use a single FreeRTOS task for all I2C polling (cooperative within the task, sequential reads, no mutex needed), and separate tasks for networking/processing. This gives you the best of both worlds: deterministic I2C timing with FreeRTOS scalability.”

13-

[4g] **Claude Conversation Link:** <https://github.com/copilot/share/c05641a4-02e4-88f1-9952-040180304891>