List of Deliverables

Client: Harry Zhang (Icon Lab)

Autonomous Drone

Hardware Upgrade Project

TL 101: Arunark, Peter, Rylan, Nabiha, Ryan and Spencer

Date Last Updated: August 5, 2025

Final List of Deliverables

1. Source Code to be run on the Raspberry Pi
2. Source Code to be run on the Remote Client Laptop
3. Mechanical Schematics for the Final Drone
4. Electrical Schematics for the Final Drone
5. Bill of Materials
6. Finalized Drone
7. User Manual for Finalized Drone
8. Comprehensive Documentation Package detailing the Design Process
9. Capstone Video

Current State of Deliverables

1. Source code for the Raspberry Pi 4

2. Source code for the Remote Client Laptop

See the GitHub repository for the current state of all source code at the following link: https://github.com/raeditio/ELEC491_TL101

3. Mechanical Schematics for the Final Drone

All .STL files for the mounting design will be provided in M4 deliverables files on Canvas: https://canvas.ubc.ca/groups/720960/files/folder/M4%20deliverable%20files

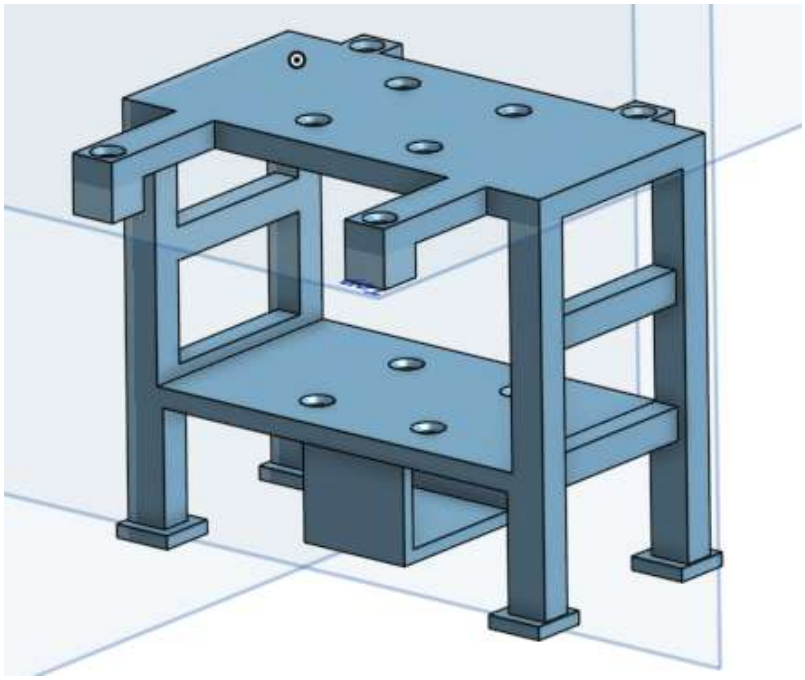


Figure 1: Drone bottom mount

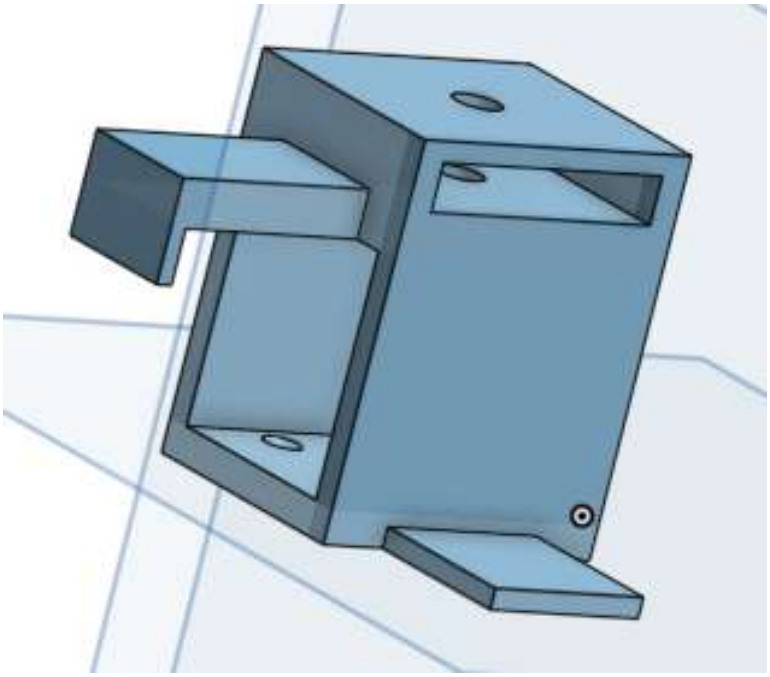


Figure 2: Drone top mount

4. Wiring Diagram for the Finalised Drone

The Electrical wiring diagram can be found in the M4 Deliverables files on Canvas and below: https://canvas.ubc.ca/groups/720960/files/folder/M4%20deliverable%20files

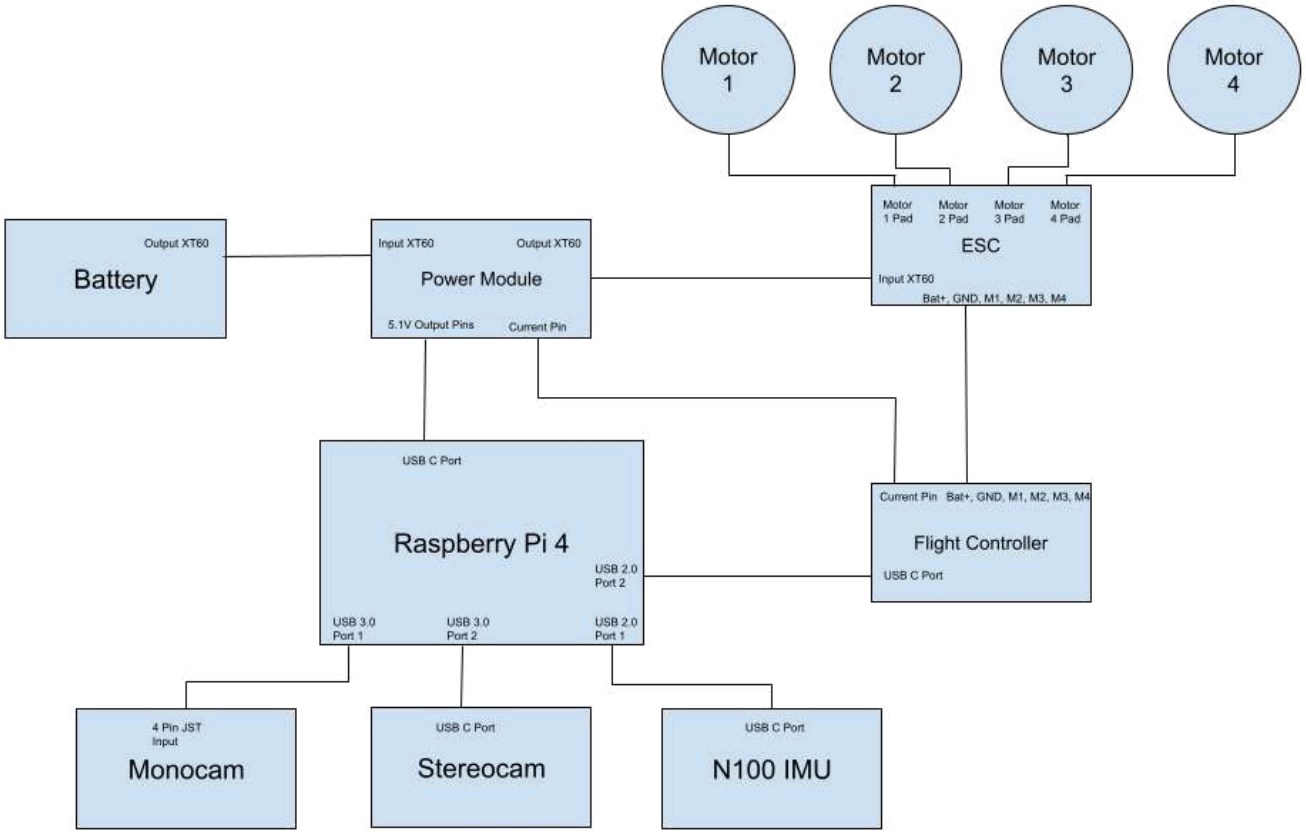


Figure 3. Drone Wiring Diagram

5. Bill of materials

The Bill of materials can be found in the M4 Deliverables files on Canvas and below:
https://canvas.ubc.ca/groups/720960/files/folder/M4%20deliverable%20files

Bill of Materials	
Item Name	**Quantity**
Kakute H7 mini flight controller	1
Raspberry-Pi 4	1
HAKRC 45A 4in1 ESC	1
CRIUS Power module 28V 90A	1
2200mAh battery	1
RealSense D435i depth camera	1
Monocam	1
Wheeltec N100 IMU	1
Drone frame top mount	1
Drone frame bottem mount	1
VCI Spark 1404 3750kV Motors	4
GF 63-5 Propellers	4
Radiolink R12DSM receiver	1
AT9S Pro transmitter	1
M2 bolts (Motors, Propellers, Frame Mounting, Raspberry Pi Fastener)	34
M2 nuts (Frame Mounting, Raspberry Pi Fastener)	10
M2.5 bolts (Raspberry Pi Fastener)	2
M2.5 nuts (Raspberry Pi Fastener)	2
M2 Split Lock Washers (Raspberry Pi Fastener)	2
M2 Rubber Standoffs (Flight Controller & ESC Mounting)	4
M2.5 Split Lock Washers (Raspberry Pi Fastener)	2
M3 Standoffs (Flight Controller & ESC Mounting)	4
5pin JST Connection (Flight Controller -> ESC)	1
3pin Jumper (Flight Controller to Raidiolink)	1
USB C-USB A 2.0 6" Cable (Raspberry Pi -> IMU)	1
USB C - USB A 3.0 6" Cable (Raspberry Pi -> Flight Controller)	1
USB C - USB A 3.0 12" Cable (Raspberry Pi -> Intel Realsense)	1

Table 1: Bill of Materials

6. Finalized Drone

The drone will be handed off to the client after the final M4 presentation, an image can be seen below.

1. Electronic Speed Controller & Motors
2. Mono – Colour Camera
3. Raspberry Pi4
4. Intel RealSense Depth Stereo Camera
5. Radio Transceiver
6. Power Module
7. Flight Controller
8. Frame
9. Mounts
10. Battery

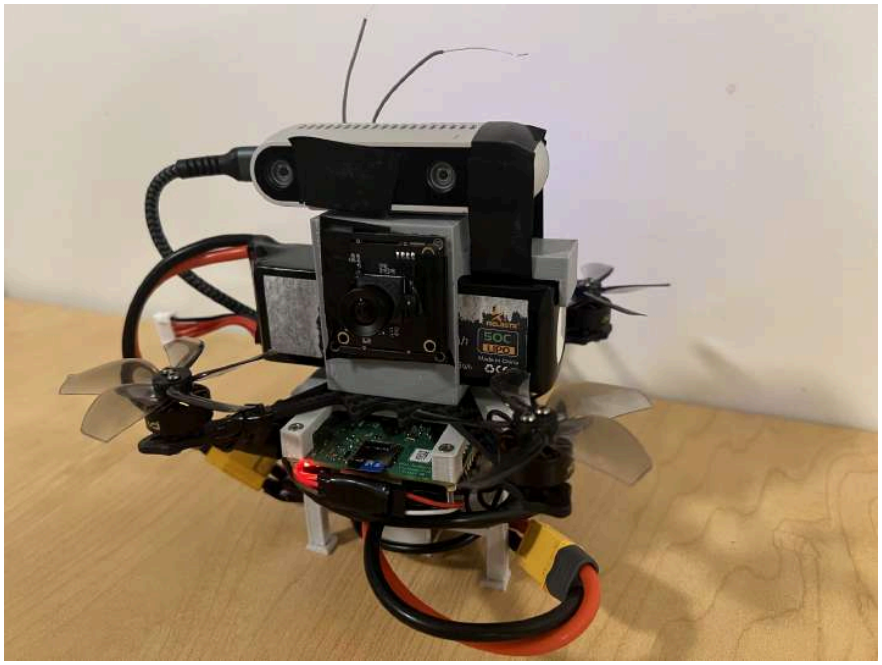


Figure 4: Finalised TL101 Drone

7. User Manual For Finalised Drone

The user manual can be found in the M4 Deliverables files on Canvas:
https://canvas.ubc.ca/groups/720960/files/folder/M4%20deliverable%20files

8. The capstone video you produced

Appendix

Appendix A. Team Roles

Name	Initials	Tech Lead	Management Lead
[student name]	[Identifying Initials]	[Responsibilities]	[Responsibilities]
Rylan Bowen-Colthurst	RBC	Power Management	Lead Client Support
Arunark Singh	AS	Microprocessor and Communication	Lead Admin + Treasurer
Spencer Tipold	ST	Simulation + Hardware Design	Lead Lab & Validation
Ryan Jung	RJ	Software and Sensor	Lead Project Organizer
Nabiha Khan	NK	Simulation + Software Design	Lead Communications
Peter Kim	PK	Power Management +Firmware	Lead Data & Record Tracking

Appendix B. Deliverables Contributions

Document Section	**Major Design**	**Minor Design**	**Major Testing**	**Minor Testing**
[number & title]	[Initials]	[Initials]	[Initials]	[Initials]
1 Source code to be run on the Raspberry Pi	RJ, AS, NK	NK	NK, AS, RJ	RBC, ST
2 Source code to be run on the Remote Client Laptop	AS, RJ	RJ	NK, AS, RJ	RBC, ST
3 Mechanical Schematics for the final drone	ST	RBC	ST, RBC	
4 Wiring Diagram for the Final Drone	RBC	PK, ST	RBC	PK, ST
5 Bill of Materials	ST, RBC		ST, RBC	
6 Finalized Drone	All	All	All	All
7 User Manual for Finalised Drone	RBC, ST, AS, RJ			
8 Capstone Video	PK	NK	All	All

Requirements Document

Client: Harry Zhang (Icon Lab)

Autonomous Drone Hardware Upgrade Project

TL 101: Arunark, Peter, Rylan, Nabiha, Ryan and Spencer

Date Last Updated: August 5, 2025

Change Log

Date	Author	Location	Change
2025-02-09	All	Whole Document	First Draft Created
2025-06-16	RJ	2.7 F5 Depth Data Acquisition	Clarified change in the software pipeline
2025-06-16	RBC	2.1 F1	Removed Tight space navigation as a function requirement due to inability to test, F1 now represents manual control and manual takeover
2025-06-16	RBC, PK	2.7 F7	Reclassified the of loss of communication and low battery behaviour as a functional requirement
2025-06-16	NK	2.8 F8	Introduced functional requirement for path planning in place of tight space navigation as this requirement is much more verifiable.
2025-08-05	RBC, PK	2.3, 2.6	Functional Requirement 2.3 was revised to reflect the team's decision to use a legacy power module instead of a custom-designed BMS. Functional Requirement 2.6, "Sensor Data Acquisition," has been removed. After consulting with the client and presenting the project timeline, it was decided to focus on delivering the Minimum Viable Product (MVP) by not including the additional sensor capabilities. Non Functional Requirement 3.2 was removed due to a deliberate design decision to exclude propeller guards, see *Design Document section 2.1.2* for more details

Table of Contents

1 Overview **5**
1.1 Problem Statement 5
1.2 Project Outcome 5
2 Functional Requirements **6**
2.1 F1: Manual Controls and Manual Takeover 6
2.2 F2 Communication Latency and Bandwidth 6
2.3 F3 Battery Safety 7
2.4 F4 Image Sensor Requirements 7
2.5 F5 Depth Data Acquisition 8
2.6 F6 Manual/Autonomous Switching Procedure 8
2.7 F7 Path Planning & Perimeter Mapping 8
3 Non-Functional Requirements **10**
3.1 NF1 Operating System Compatibility 10
4 Constraints **11**
4.1 C1 Frame Size 11
4.2 C2 Room Size coverage 11
Appendix A. Contributions **12**
5 Glossary **13**
6 References **16**

1 Overview

1.1 Problem Statement

The ICON Lab's primary research focuses on automating building inspection using fully autonomous indoor drones. The core problem this work addresses is that "keeping an up-to-date 3D representation of buildings is a crucial yet time-consuming step for Building Information Modeling (BIM) and digital twins," as existing Scan-to-BIM methods rely heavily on manual operations.

To solve this, the original ICON drone was designed to autonomously navigate indoor environments and generate semantic-rich point clouds. This approach leverages 3D reconstruction to automate the data acquisition and semantic segmentation steps of the Scan-to-BIM process, which would otherwise be performed manually. Furthermore, using a drone provides better coverage and avoids the terrain constraints of ground-based robots.

Despite its success, the original ICON drone has several key limitations: its size and weight restrict its flight time and ability to operate in confined spaces, and its sensing capabilities are limited to only an RGB camera. The current project aims to address these limitations by upgrading the drone's hardware to reduce its size and weight and integrate additional sensors while preserving its existing autonomous capabilities. This hardware upgrade is essential for improving the drone's utility and expanding its application scope.

1.2 Project Outcome

The successful completion of this project will provide ICON Lab with a more compact and agile drone capable of navigating tighter spaces and flying for longer periods. The addition of a thermal camera and air quality sensors will allow the drone to perform more complex inspection tasks, providing greater value to clients who require detailed data for structural diagnostics and maintenance. Furthermore, the project will provide ICON Lab with an upgraded drone system that will support its vision of advancing indoor autonomous inspection technologies by transferring computing power from an onboard mainframe to a wireless local host. This upgrade enables ICON Lab to further enhance the drone's capabilities, allowing for the deployment of more advanced reconstruction models that were previously limited by the onboard computer's system and weight limitations. The project's results could also serve as a foundation for future research and development, potentially leading to innovations in building modeling technology.

A relevant case example is the use of drones equipped with thermal cameras for building inspection by FIXAR[1]. In December 2021, FIXAR deployed a drone with an advanced EO/IR dual sensor camera and a long-range data transmission module to inspect several buildings under tough winter conditions. The drone covered 2 km² in just 9 minutes, efficiently identifying issues such as thermal leaks in walls, roofs, and window structures. This drone-based approach allowed more detailed and rapid data collection compared to traditional inspection methods, improved operator safety,

minimized exposure to adverse weather, and provided clients with comprehensive diagnostic information.

2 Functional Requirements

The final deliverable for this project is a fully functional autonomous drone platform. The following functional requirements (F1 through F8) and non-functional requirements (NF1 and NF2) describe the essential features, capabilities, and safety protocols this drone must possess to be a suitable and effective tool for the ICON Lab's research. Each requirement specifies a critical aspect of the drone's design, from its control systems and communication to its safety features and sensor suite, all of which are necessary to meet the project's overall objectives.

2.1 F1: Manual Controls and Manual Takeover

The drone must support manual piloting via radio controller in addition to autonomous flight, with a fast and reliable manual override for critical situations. This is essential for both operational flexibility and safety in confined indoor environments.

- Manual Flight Response
    - The drone must respond to RC joystick inputs via the radio controller.
    - ***Test Methods*:**
        1. In the Gazebo SITL simulation, verify flight controller accepts and executes RC joystick commands.
        2. With hardware assembled (motors connected), perform a manual flight test and confirm that real-world stick commands result in expected motion.

- Failsafe Protocol (Manual Takeover)
    - Operators must be able to trigger manual override during autonomous navigation to take over manual controls from autonomous navigation. Operators must also be able to trigger a kill switch to completely disarm the drone in the event of emergencies.
    - ***Test Methods**:* Implement both kill switch and manual takeover switch in real flight; verify immediate transfer of control.

2.2 F_2 Communication Latency and Bandwidth

Establishing a reliable connection with low latency is critical to the function of the drone, as autonomous navigation relies on near-instantaneous feedback from sensors and rapid transmission of control commands. To navigate autonomously, the drone must maintain a stable and fast connection with the local host, ensuring that even the smallest delay does not compromise its responsiveness. Additionally, our design must support streaming high-quality video footage over Wi-Fi as the drone travels across rooms and through varied environments, enabling real-time remote monitoring and decision-making. Any latency or interruption in the video stream could hinder effective obstacle

detection and path planning, which is why achieving both low latency and high-bandwidth connectivity is a top priority for our system.

- Based on the requirements of the Software Navigational Models (Like VINS and FUEL), the maximum latency accepted is 3 ms. Additionally, to achieve such a latency with large amounts of data, the theoretical bandwidth is to be >100 Mbps.
- ***Test Methods:*** Verify that all data are published via ROS at required frequencies (e.g. > 100 Hz for IMU)

2.3 F_3 Battery Safety

There are safety risks when using lithium batteries, the worst of which is fire. To eliminate this risk, it is critical that the drone monitors the battery to ensure safe usage.

- The voltage of the battery must be monitored and ensure that it does not fall below safe levels. Draining a cell below 3.0V can result in permanent damage to the battery and cause it to catch fire in the worst of cases. To mitigate this, the drone should land when cell voltages fall to 3.5V. For this reason the battery, a 4S Lipo, must not drop below 14.0V.
- The total current drawn from the battery. The total draw must be below 80 % of the battery's maximum discharge rate for an adequate safety margin.
- To achieve this, the voltage must be measured with an accuracy greater than $\mp$0.05 V and current with an accuracy of $\mp$ 1.0 A
- ***Test Methods:*** Bench tests using lab bench power supplies to simulate battery voltages, ensuring accurate measurements. Pass measurements to the flight controller with motors detached from the drone to verify the flight controller's response

2.4 F_4 Image Sensor Requirements

The camera quality is another critical requirement that the drone must satisfy. Since it was decided previously to keep a single camera module responsible for both the path-planning and the recording of the footage later to be used for the 3D reconstruction, the quality of the image sensor should satisfy the input requirements of each respective algorithm. To generate accurate and in-depth information using the depth estimation algorithm, it is advantageous to have a clear image. Certain considerations should be made for a reliable SLAM[1] as well.

- Resolution: 1280x720. Although higher resolution is better, it will require higher computational power
- Framerate: 30 fps–a framerate lower than this will not be able to track the movement of the drone
- Global Shutter: limits image warping in cases of rapid movements
- Auto-exposure and support for a wide dynamic range are preferred to handle varying light conditions
- Fixed focus performs better in SLAM

- ***Test Methods:*** Collect sample footage under varying lighting and motion; verify sensor meets algorithm input needs.

1: simultaneous localization and mapping. Used to map an environment and determine the position of a robot in that map[2]

2.5 F_5 Depth Data Acquisition

To ensure safe and reliable autonomous navigation, both the accuracy of the SLAM algorithm and the quality of its input sensor data are critical. These factors directly impact the drone's ability to localize and map its environment in real time. In particular, the depth information must closely match the performance of the original stereo camera system, with a target disparity of less than 1.5 meters under standard indoor conditions, as specified by the client.

- $< \pm$ 1.5 m error
- Reliable performance in varying light conditions
- Reliable performance in varying motion
- ***Test Methods****:* Measure depths against a wall using the sensors. Compare the manually measured depths (distances) against the depths sensed at varying lights and motions

2.6 F_6 Manual/Autonomous Switching Procedure

The drone must be capable of switching to manual control in the event of critically low battery or sustained communication loss with the local host. This ensures the drone is not damaged and does not endanger its surroundings.

- Switch Conditions: The drone must switch to manual control automatically when one of the battery cells reaches 3.6V or if communication is lost for >5 seconds.
- ***Test Methods****:* Simulate both low-battery and link-loss conditions without motors connected; verify that the control switch happens.

2.7 F_7 Path Planning & Perimeter Mapping

The drone must autonomously plan and execute a flight path that maps at least 90% ± 10% of the indoor room perimeter. Mapping and path planning should be performed using an onboard SLAM system (VINS-Fusion and FUEL) with the resulting route visualized in RVIZ.

- The perimeter route must be computed incrementally by the planner (e.g., frontier-based regional growth, hierarchical planning).
- Planner outputs waypoints and coverage path, shown in RVIZ for operator verification.

Test Methods

- Simulation
    - Integrate VINS-Fusion or FUEL in SITL simulation.
    - Visualize the planned perimeter path in RVIZ.
    - Quantify perimeter coverage; acceptable range is 80–100%.

- Real-World Flight (Indoor, Hardware)
    - Load planned perimeter waypoints onto the flight controller.
    - Execute a waypoint mission indoors.
    - Record flight path coverage; confirm it lies within 80–100% of the measured room perimeter.

3 Non-Functional Requirements

3.1 NF_1 Operating System Compatibility

To eliminate compatibility concerns for the final product, the entire project (including the drone and client PC) uses the same operating system, i.e., Ubuntu 20.04.6 LTS. Such a requirement will guarantee a hassle-free use of the final product by the client's organization members, with minimal need for additional training to use the software. Additionally, this reduces issues when the client intends to further develop the product, as they have shown interest in extending the project's scope.

- ***Test Methods:*** Validate all software modules and launch scripts executed successfully in the specified OS environment.

4 Constraints

4.1 C_1 Frame Size

In the initial stages of the project, the client asked that the drone frame size be constrained to less than five inches. Drone frames below five inches are available on the market, however, these drones are not autonomous and are incredibly lightweight

.

- After a feasibility study was performed, it was discovered that the additional hardware for autonomous navigation would prove too heavy for a drone of this size. After discussing with the client further, the new constraint size for the drone is under eight inches
- ***Test Methods:*** Measure the physical frame from the longest point (diagonally); verify it does not exceed 8 inches after assembly.

4.2 C_2 Room Size coverage

The previous drone that ICON Labs produced was capable of surveying a room with a size of roughly 100 m^2. The drone that will be created by the end of this must meet or exceed the performance of ICON's previous drone; therefore, the final design must be capable of surveying a room greater than or equal to 100m^2.

- To meet this constraint, we specified a drone flight time of over 10 minutes. The previous drone achieved only 3–5 minutes of flight time. This increase provides a generous margin for flight path coverage, allowing for slower, more stable flight and ensuring the constraint is exceeded.
- **Test Methods:**
    - Time continuous flight operation on a full charge under simulated inspection conditions.
    - Use mission logs or RViz mapping output to verify spatial coverage ≥ 100 m².

Appendix A. Contributions

Document Section	**Major Content**	**Minor Content**	**Author**	**Review**
[number & title]	[Initials]	[Initials]	[Initials]	[Initials]
1.1 Problem Statement	PK		NK	RJ, PK, AS
1.2 Project Outcome	PK	NK	RBC	RJ, PK, AS
2.1 Manual Controls and manual Takeover	RBC	NK	RBC	AS
2.2 Communication Latency and Bandwidth	AS	NK	AS	RJ, RBC
2.3 Battery Safety	RBC	PK	RBC	RJ, ST
2.4 Image Sensor Requirements	RBC	PK	RBC	RJ, PK
2.5 Depth Data Acquisition	RJ	NK, AS	RJ	RBC, ST
2.7 Manual/Autonomous Switching Procedure	PK	RBC	PK	RBC
2.8 Path Planning and Perimeter Mapping	NK	AS	NK	RBC
3.1 Operating System Compatibility	RJ, AS	NK	RJ	RBC
3.2 Propeller Guarding	ST	RBC	RBC	RJ, PK
4.1 Frame Size	ST	RBC	RBC	RJ, PK
4.2 Room Size Coverage	NK		RBC	RJ

5 Glossary

- **Air Quality Sensor:** A device that detects and measures pollutants or particulate concentration in the air; used for inspection mapping.
- **Autonomous Flight**: Drone operation without human control, relying on software and sensor data to navigate.
- **Bandwidth:** The maximum rate of data transmission, affecting real-time video and sensor data streaming.
- **Battery Management System (BMS):** Monitors voltage and current to prevent battery over-discharge and fire risks.
- **Clearance:** The space between propeller blades and guards to ensure safety during collisions or deflections.
- **Communication Latency:** Delay between sending and receiving data, critical for real-time autonomous control.
- **Constraint:** A strict requirement or limitation imposed on the design, such as frame size or flight area.
- **Depth Estimation:** The process of calculating the distance of objects from the drone using camera data.
- **Depth-Anything-V2:** A deep learning model used to infer depth from a single image frame.
- **Disarm:** A failsafe function that stops motor output immediately in emergencies.
- **Docker:** A platform used to containerize software for compatibility and consistency across systems.
- **Drone Frame:** The physical structure that holds all drone components; size affects agility and payload.
- **FOV (Field of View):** The extent of the observable world captured by a camera at any moment.
- **Frame Size:** The diagonal measurement of the drone from motor to motor; impacts maneuverability.
- **Frame rate:** The number of frames captured per second by the image sensor, affects SLAM accuracy.
- **Global Shutter:** A camera feature that prevents image distortion during fast movement by capturing all pixels simultaneously.
- **Hardware-in-the-Loop (HITL):** Simulation that allows real-time testing of software on physical hardware without actual flight.
- **Joystick Input:** Manual control signals sent via radio transmitter to pilot the drone.
- **Kill Switch:** An emergency manual override that immediately cuts off all drone motor outputs.
- **Latency Threshold:** The maximum allowable communication delay for safe and responsive flight operation.
- **Local Host:** An offboard computer used to run heavy computation tasks like mapping and navigation.
- **Manual Flight:** Drone operation using direct human control via RC transmitter.
- **Manual Override:** The ability to switch from autonomous to manual control in real-time.
- **Mission Log:** A digital record of drone telemetry and position data during flight.
- **Monocular Camera:** A single-lens camera used for both SLAM and video recording, paired with deep learning for depth.
- **Multi-Mode Control:** Capability to switch between autonomous and manual operation modes seamlessly.
- **ORB-SLAM3:** A visual SLAM system supporting monocular, stereo, and RGB-D input, used for localization and mapping.

- **Operating System Compatibility:** Requirement that all software runs on Ubuntu 20.04.6 LTS for client-side deployment.
- **Path Planning:** Algorithm that computes an efficient route for the drone to follow while avoiding obstacles.
- **Perimeter Mapping:** The process of autonomously recording the edges or boundaries of a room or structure.
- **Ping Test:** A tool used to measure latency and connection stability between drone and local host.
- **Pixel Disparity:** Difference between pixels in stereo vision used to compute depth.
- **Pose Estimation:** The real-time computation of the drone's position and orientation in space.
- **Power Budget:** The total allowable electrical consumption across all drone subsystems.
- **Propeller Guarding:** Protective structures to prevent propeller damage and injury during collision.
- **RC Transmitter:** Handheld device that sends joystick signals to the drone's receiver.
- **RGB-D Input:** A data stream consisting of both color (RGB) and depth (D) information.
- **RViz:** ROS visualization tool used to observe planned paths and sensor output in real time.
- **Room Size Constraint:** Requirement that the drone must map rooms ≥ 100 m² to match or exceed legacy performance.
- **SLAM** (Simultaneous Localization and Mapping): Technique used to map unknown environments while tracking the drone's position within it.
- **Safety Margin:** A conservative threshold set below maximum capacity to ensure safe operation.
- **Sensor Integration:** The process of combining thermal and air quality sensors with the drone's data pipeline.
- **Sensor Suite:** Collection of all active onboard sensors including thermal, visual, and air quality modules.
- **Simulated Inspection Conditions:** Controlled test scenarios that mimic real-world autonomous missions during testing.
- **Software Pipeline:** Sequence of processing steps for image, depth, and telemetry data to produce flight commands.
- **Switch Condition:** A trigger event (e.g., low battery) that changes the drone's operating mode.
- **Test Coverage:** The percentage of a room's perimeter or area the drone successfully maps.
- **Thermal Imaging:** Capturing temperature data to create heat maps for structural diagnostics.
- **Ubuntu 20.04.6 LTS:** The Linux operating system used by both the development team and client for compatibility.
- **Waypoint Mission:** A pre-planned flight path made up of specific 3D coordinates the drone must visit

6 References

- [1] FIXAR, "Building Inspection with Thermal Camera," FIXAR, 2021. [Online]. Available: https://fixar.pro/casestudies/building-inspection-with-thermal-camera/ [Accessed: 5-Aug-2025].
- [2] MathWorks, "Simultaneous Localization and Mapping (SLAM)," *MathWorks*, 2024. [Online]. Available: https://www.mathworks.com/discovery/slam.html. [Accessed: 30-Mar-2025].

Design Document

Client: Harry Zhang (Icon Lab)

Autonomous Drone Hardware Upgrade Project

TL 101: Arunark, Peter, Rylan, Nabiha, Ryan and Spencer

Date Last Updated: August 5, 2025

Table of Contents

Abstract 4
1.0 High-Level Overview 5
1.1 Existing ICON Drone Architecture 5
1.2 TL101 MVP Drone Architecture 5
1.3 MVP Drone Subsystems 7
2.0 Manual Flight Subsystem 9
2.1 Manual Flight Hardware 9
2.1.1 Motor, Propellers, and Battery Selection 9
2.1.2 Frame/Mounts 11
2.1.3 4-in-1 Electronic Speed Controllers (ESC) 15
2.1.4 Flight Controller 15
2.1.5 Receiver & Transmitter 16
2.2 Manual Flight Software - 16
2.2.1 PX4 Firmware on Flight Controller 16
2.2.2 Hardware-in-Loop Simulation 16
3.0 Autonomous Flight Subsystem 18
3.1 Autonomous Flight Hardware 18
3.1.1 Onboard Raspberry Pi 4 and Remote Client Connection 18
3.1.2 Intel RealSense Camera 18
3.1.3 Inertial Measurement Unit (IMU) 19
3.1.4 Raspberry Pi Compatible Mono Camera 19
3.1.5 Power Module 20
3.2 Autonomous Flight Software 22
3.2.1 Autonomous Navigation Pipeline 23
3.2.2 Wireless Communication and ROS Node Connection 25
3.3 Autonomous Drone Control Simulation 28
3.3.1 Simulation Design Decisions and Development Strategy 28
3.3.2 XTDrone as the Simulation Platform 29
3.3.3 Summary of Important Algorithms & Libraries 29
3.3.4 Simulation Development Workflow 30
3.3.5 Final Simulation Description 31
3.3.6 Performance of Autonomous Navigation Algorithms 32
3.3.7 Future Considerations 33
4 Summary 34
Appendix A. Contributions 35
Appendix B. Power Distribution Board and BMS Board Design Alternative 37
Power Distribution Boards 37
Battery Management Board 39
Appendix C Upgraded Drone Architecture Proposals 41
Re-explore Shift to Raspberry Pi Zero 41
Mono Cam Only Alternate System 41
Stereo Cam Only Alternate System 42
5.0 Glossary 44
6.0 References 47

Abstract

This report presents the final deliverable of the TL101 drone project. The system was redesigned into a compact 6-inch platform with full manual and autonomous flight capability. Compute-heavy tasks were offloaded to an external PC, reducing onboard weight. A wireless ROS-based communication link connected the Raspberry Pi to the host. Manual flight was validated through hardware-in-loop testing and physical flights. Autonomous navigation used the RealSense D435i, VINS-Fusion for localization, and FUEL for path planning. A virtual simulation was built to test the autonomy stack without risking hardware. It includes all key components of the legacy client codebase, with final path planning algorithms still in development. The simulation serves as a benchmark for expected behavior in physical flight.

1.0 High-Level Overview

1.1 Existing ICON Drone Architecture

The legacy ICON platform is a mid-size, 10-inch-class quad-rotor (≈ 450 mm motor-to-motor diagonal) designed for indoor mapping and autonomous exploration. It employs four high-thrust motors driven by a compact four Electronic Speed Controllers (ESCs) and supplied by a medium capacity, high-power Li-Po battery pack.

The drone is intended to acquire high-fidelity video footage of interior spaces to be later used by the client's 3-D reconstruction algorithm. Manual drone flight operated by a pilot at close proximities to structures is both labour-intensive and hazard-prone. Furthermore, the input to the 3-D reconstruction algorithm requires stabilised video with negligible motion-induced artefacts. The legacy ICON drone carries both the Intel RealSense D435i and GoPro Hero 11 Mini cameras to mitigate these challenges and enables autonomous flight with high-quality video capture, respectively.

Figure 1 - Legacy ICON Drone

The D435i is a stereo (dual-lens) camera that comes with an embedded depth-processing Application Specific Integrated Circuit (ASIC). Synchronized left/right RGB frames from the D435i and inertial measurements from an external IMU feed VINS-Fusion, a multi-sensor SLAM estimator. Simultaneously, depth images from the D435i ASIC drive the FUEL ("Fast UAV ExpLoration") path planning algorithm for autonomous flight. Both software stacks run under Robotic Operating System (ROS) on Ubuntu 20.04, hosted on an Intel NUC micro-PC featuring an embedded-grade Core i7 processor. At ~0.53 kg, the PC is a powerful yet compact payload that supports the computational demands of the VINS-Fusion and FUEL algorithm. As a result, it was a viable on-board flight computer whose weight could be supported by the large frame and powerful motors.

1.2 TL101 MVP Drone Architecture

The revised ICON prototype targets a compact 6-inch-class airframe (≈ 280 mm motor-to-motor diagonal). Four VCI Spark 1404 – 3750 KV motors drive the craft, each regulated by a 4in1 ESC (Electronic Speed Controller), powered by the same 4S LiPo battery. Compared with the legacy 10-inch platform, this downsized frame yields a lighter, more agile vehicle while still supplying the thrust margin required for stable indoor hover and waypoint flight. The PixHawk 6C flight controller was also replaced with a significantly smaller Holybro Kakute H7 Mini that runs the same firmware to ensure software compatibility.

The sensor suite preserves the Intel RealSense D435i because its on-board depth engine is indispensable for the FUEL exploration planner. The previous GoPro, however, is replaced by a Raspberry Pi–compatible global-shutter monocamera that records raw RGB directly to a Raspberry Pi 4 Model B. Depth frames from the D435i stream over Wi-Fi to an off-board workstation running VINS-Fusion to compute visual–inertial odometry and FUEL to then generate collision-free waypoints, which are transmitted back to the Pi. This relays attitude and position set-points to the flight controller. In effect, computational load migrates from an on-board Intel NUC to an off-board host, reducing payload mass without sacrificing SLAM or planning performance.

The 4S Li-Po battery's voltage exceeds the operational requirements of the Raspberry Pi and camera modules, necessitating a dedicated power distribution and battery management equipment. The power distribution equipment steps down the voltage to the levels required by each subsystem, while the battery management equipment monitors the battery's live current and voltage to prevent discharge below a critical threshold. For this application, the Crius Power Module (6-28V, 90A) for Pixhawk APM/APM2.8 was selected. This component is not only compatible with the updated flight controller but was also provided by the client, which significantly reduced project costs and lead time.

This configuration defines the team's minimum viable product (MVP). Smaller form-factor alternatives are explored later in the report but will only be pursued after the MVP has been flight-proven.

Sub-system	**Legacy Platform**	**Current MVP**	**Action**
Frame	10-inch X-quad	6-inch X-quad	Downsized
Motors	4 × 2550 KV T-Motor	4 × 3750 KV VCI Spark	Replaced
ESC	4 × EMAX 45A	1 × HAKRC 45A 4-in-1	Replaced
Flight Controller	Pixhawk 6C Mini	Holybro Kakute H7 Mini	Replaced (PX4 Retained)
Depth Camera	Intel RealSense D435i	*Same unit*	Re-used
RGB Camera	GoPro 11 Mini Action	Pi-compatible global-shutter mono-cam	Replaced
On-Board Computer	Intel NUC i7	Raspberry Pi 4 Model B	Replaced
Off-Board Computer	N/A	Lenovo Legion Laptop	Added
RC Links	Radiolink AT9S Pro + R9DS	*Same link*	Re-Used

Table 1: Summary of Legacy Icon Drone Components and Their Use in the Upgraded Drone

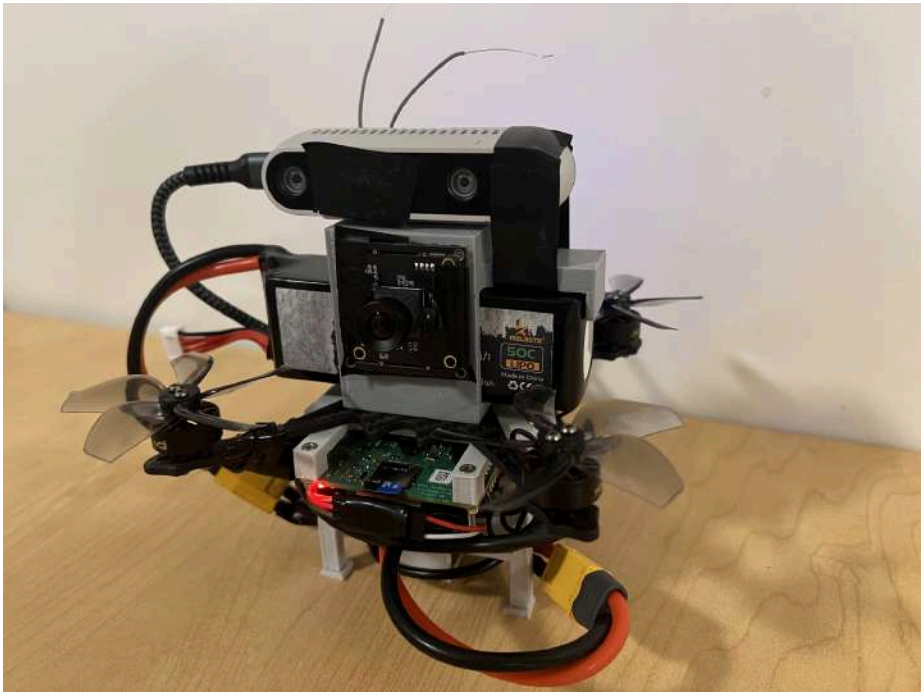


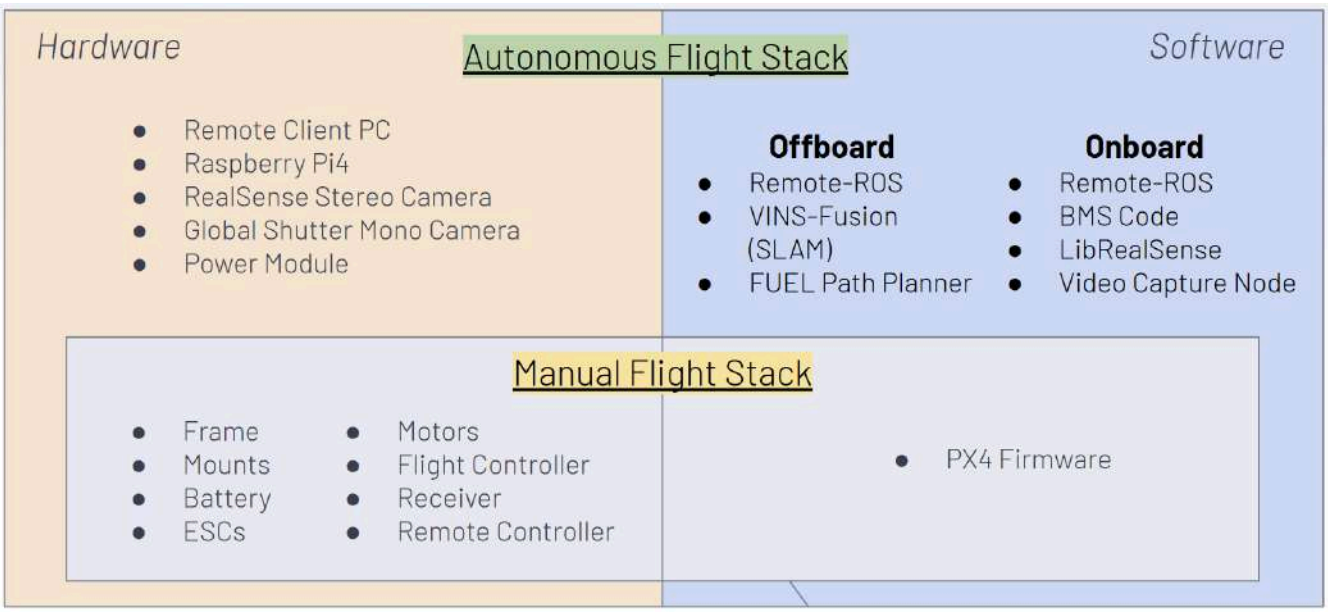
Figure 2. TL101 Upgraded Drone Assembly

1.3 MVP Drone Subsystems

Figure 2 distils the entire platform into two nested sets of capabilities:

1. **Manual-flight core (inner set).**
   This is the bare minimum ensemble of hardware and firmware required for a human-piloted quad-rotor. It comprises the physical airframe and mounts, propulsion stack (battery, ESCs, motors), radio receiver, and remote transmitter—all governed by the **PX4 firmware**. If these items alone are present, a pilot can arm, fly, and land the vehicle by RC.

2. **Autonomous-flight overlay (outer set).**
   Autonomy can only exist on top of a working manual drone; hence, the inner set nests inside the outer. The additional hardware adds perception, compute, and system monitoring. Software is split by execution location:
   • On-board (runs on the Pi)
   • Off-board (runs on a local host PC)
   A bidirectional Wi-Fi link relays depth frames outward from the Pi to the local host and waypoint commands back from the local host to the Pi. This enables closed-loop exploration while lightening the payload relative to the NUC.



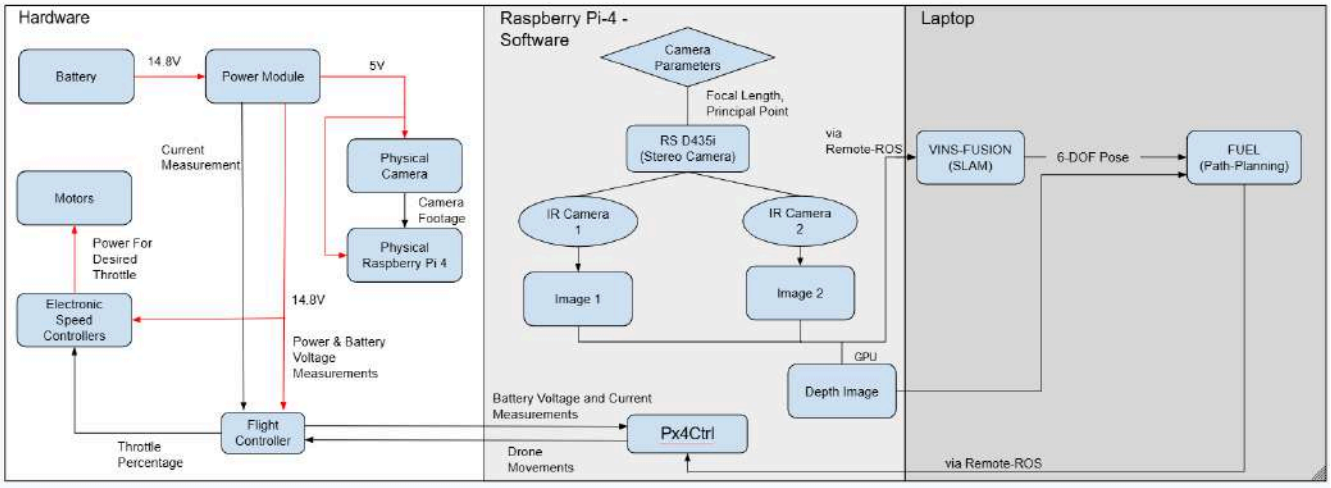


Figure 3: TL101 System visual V3 - July 10, 2025 & Autonomous vs Manual Flight Stack

This system diagram illustrates the full integration of hardware and software components in the autonomous drone platform, highlighting the interaction between onboard and offboard computation. The hardware subsystem (left) includes the battery, motors, ESCs, flight controller, and sensors, all coordinated through the Raspberry Pi 4, which handles camera input and power management. The Intel RealSense D435i stereo camera provides synchronized IR and depth images that are processed onboard and then transmitted to a remote laptop via ROS. The laptop runs VINS-Fusion for SLAM, which outputs a 6-DOF pose to the FUEL path-planning algorithm. Finally, path commands are sent back to the flight controller through the Raspberry Pi, completing a closed-loop autonomous navigation system.

2.0 Manual Flight Subsystem

Since Milestone 2, the drone hardware has undergone several design iterations. This section follows evolution and outlines key decisions that were made to overcome challenges in power-train sizing, thermal management, and weight reduction, among others. The section concludes with a Hardware-In-Loop simulation with the flight controller that was used to de-risk new configurations. Comprehensive bench-test logs are catalogued separately in the *Verification & Validation* document under the same respective sections.

2.1 Manual Flight Hardware

This section covers the minimum hardware required to fly a drone manually.

2.1.1 Motor, Propellers, and Battery Selection

The design decisions for which motor, propeller, and battery to select for the drone are very intertwined and are dependent on size, weight, and flight time restrictions. The heavier the drone is, the larger the propellers, motors, and batteries have to be. In order for the drone to be able to effectively maneuver, it requires a thrust-to-weight ratio of between 2-3 [4].

The weight of the drone was estimated to be about 590 grams in total, however due to the addition of an external IMU and datasheet inaccuracies with the 4in1 ESC weight the actual drone weight is 658 grams in total. Motor propeller and battery selection was based upon achieving a 10 minute flight time with the anticipated weight, the increase in actual weight led to a slight decrease in expected flight time. The breakdown can be found in Table 1 below.

Anticipated Weight	Current Components	Weight (g)
	Frame + mounts	56
	Battery	226
	Reciever + FC (extra wire)	13
	Intel Real Sense Sterocamera	72
	Power Module (with usbc)	32
	Monocamera	50
	Raspberry Pi 4	48
	ESC + Motors + Propellers	54.5
	USB Cables	25
	Mounting Hardware	15
	Total	**591.5**

Actual Weight	Current Components	Weight (g)
	Frame + mounts	67
	Battery	226
	Reciever + FC (extra wire)	13
	Intel Real Sense Sterocamera	72
	Power Module (with usbc)	32
	Monocamera	50
	Raspberry Pi 4	48
	ESC + Motors + Propellers	88
	USB Cables	37
	Mounting Hardware	15
	IMU	10
	Total	**658**

Table 2: Drone Weight, anticipated vs actual

Motor thrust was determined using the manufacturer's thrust–power curves, which include test data for various propeller options. We chose motors based on the thrust produced at around 50% throttle, then scaled that value for all four motors. This approach guarantees a minimum 2:1 thrust-to-weight ratio, enabling the drone to hover comfortably at half throttle.

The selected motor is the VCI Spark 1404 3750 KV, paired with a 63 mm (2.5″) propeller, see Figure 5. Motor thrust and power was approximated to behave linearly between datapoints, therefor at around 57% throttle, each motor delivers 165 g of thrust, yielding a combined 660 g from the four motors, sufficient for the design. At full throttle, total thrust reaches 1308 g, which corresponds to a thrust-to-weight ratio of approximately 2:1, comfortably within the target range. This ensures both responsive lift-off capability and efficient hovering performance.

Testing Dates

Model	Propeller	Voltage (V)	Throttle (%)	Current (A)	Thrust (g)	Efficiency (g/W)	Power (W)	Temperature (C°)
SPARK 1404 3750KV	GF D63-5	16	30	1.26	72	3.57	20.16	82
		16	40	2.19	107	3.05	35.04	
		16	50	3.22	144	2.80	51.52	
		16	60	4.54	172	2.37	72.64	
		16	70	6.10	213	2.18	97.60	
		16	80	7.62	244	2.00	121.92	
		16	90	9.85	280	1.78	157.60	
		16	100	12.27	327	1.67	196.32	

Note:
1、The above data obtained by VCI company's test platform is only for reference;
2、Motor temperature is the spindle temperature after the throttle is tested from 0 to 100 (static test) .

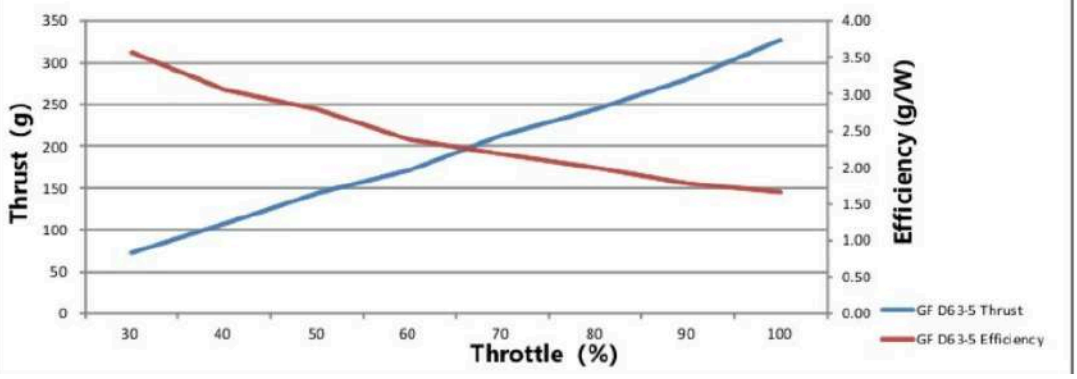


Figure 5: VCI 1404 3750kV 59Spark Thrust-Power Test Data

The voltage required for the VCI Spark 1404 is 16V, a 4S Lipo battery. Capacity requirements were calculated based on the requirement of a 10-minute flight time and total power demand. The total power demand of the drone can be found in Table 2. The average power for the motors is calculated by combining the hovering power of all four motors. Power consumption of the flight controller and electronic speed controller was not specified by the manufacturers; however, they are assumed to be negligible in relation to the power demand of the motor.

	Anticipated Weight Demand (W)	Actual Weight Demand (W)
Battery Voltage Devices		
Motors	206.04	248.32
Flight Controller	Unavailable	Unavailable
ESCs	Unavailable	Unavailable
5V Devices		
Raspberry Pi	3.9	6.25
Real Sense Camera	3.5	3.5
Monocamera	0.7	1.4
Total	214.14	259.47
Current Draw (Battery) (A)	12.74642857	15.44464286
Flight Time (min)	10.35584197	8.546652792

Table 3: Total Drone Power Demand with anticipated vs actual drone weight

Based on the anticipated average power demand of 214W, the required battery capacity for a 10-minute flight time is 2110mAh. Based on this requirement, the selected battery for use in the drone is a 2200mAh 4S Lipo battery provided by the client. As can be seen above with the difference between anticipated vs actual drone weight our actual expected flight time is closer to 8.5 minutes.

Component Selection Summary:

- Motor & Propeller
  - VCI Spark 1404 3750KV with a 63mm Propeller
  - Provides 660g of thrust at 57% throttle for hovering
  - Provides 1308g of thrust at full throttle, giving a 2:1 thrust-to-weight ratio
- Battery
  - 4S LiPo Battery
  - 2200mAh capacity to support a 10-minute flight time (with anticipated weight)

2.1.2 Frame/Mounts

The drone frame was downsized from a 10-inch to a 6-inch X-configuration to minimize system footprint and mass while preserving thrust-to-weight performance for stable indoor flight. A lightweight carbon fibre airframe was selected for its high stiffness-to-weight ratio and mechanical resilience, particularly in environments with high vibration or sudden deceleration forces. This transition resulted in a 43% reduction in diameter and a 51% reduction in total weight relative to the legacy ICON drone, enabling significantly improved flight time and agility.

To accommodate the revised component layout within the smaller frame constraints, a custom two-part mounting system was designed and fabricated using fused deposition modelling (FDM) 3D

printing. This system comprises a top mount and a bottom mount, which together integrate all major components into a rigid, compact, and easily serviceable structure.

The top mount performs several structural and integration roles. It includes a central cavity dimensioned precisely to the 2200 mAh 4S LiPo battery, providing a friction fit that holds the battery securely during flight without straps or fasteners. An integrated retention arm with a hook at the top constrains the battery laterally and ensures consistent centring.

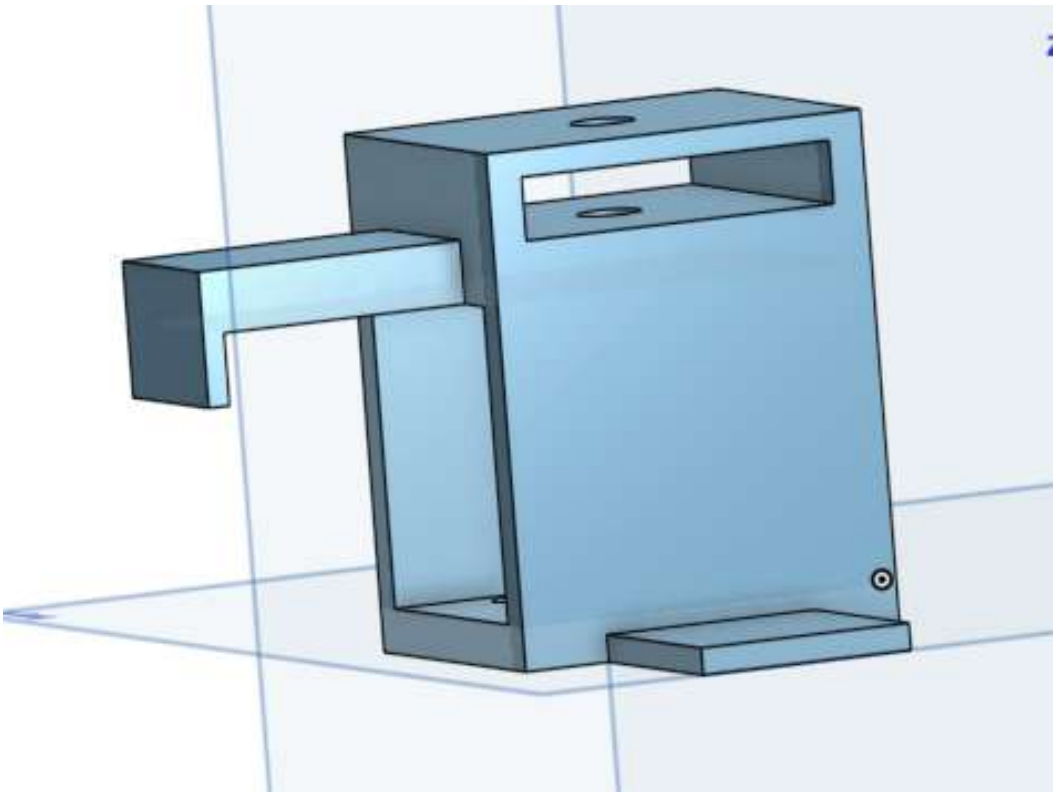


Figure 7: Top mount side view

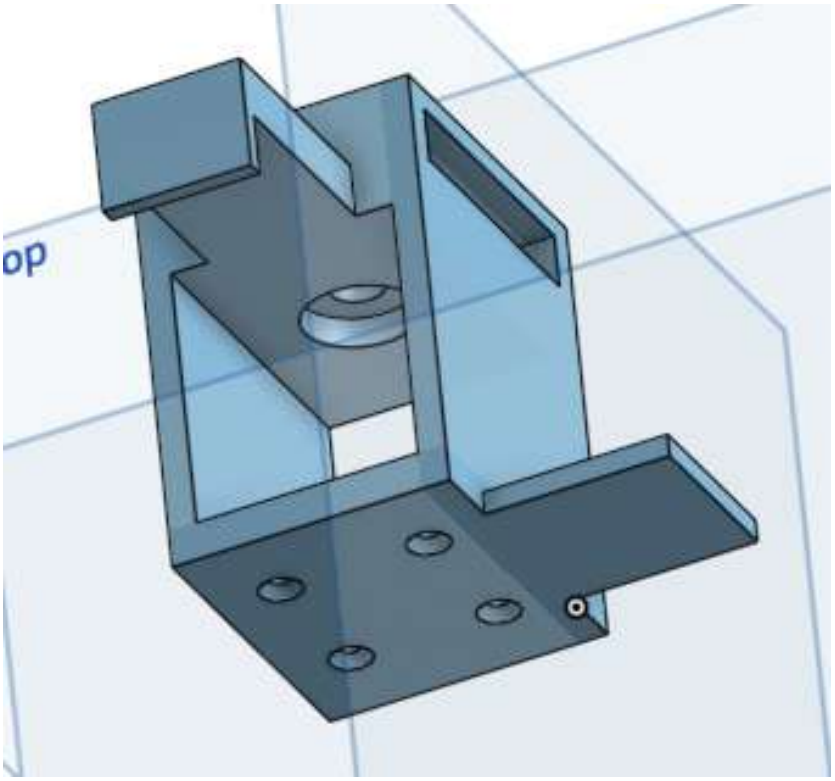


Figure 8: Top Mount bottom view

Above the battery cavity, a shelf supports the mounting bolt for the Intel RealSense D435i depth camera, while preserving clearance for the battery. Adjacent to this shelf, a vertical cable channel allows clean routing of signal and power lines from the monocamera to the Raspberry Pi 4 located below. This cable management prevents connector strain and interference during flight. The front face of the mount was dimensioned to match the width of the selected monocamera, enabling direct attachment using double-sided adhesive without additional brackets or fixtures.

A flat protruding plate on the underside serves as a mounting surface for the external inertial measurement unit (IMU). Positioned near the drone's center of mass, the IMU platform is dimensioned for adhesive attachment and helps reduce rotational artifacts in motion estimation. The mount attaches to the frame using four M2 countersunk screws, recessed to avoid obstructing the battery cavity.

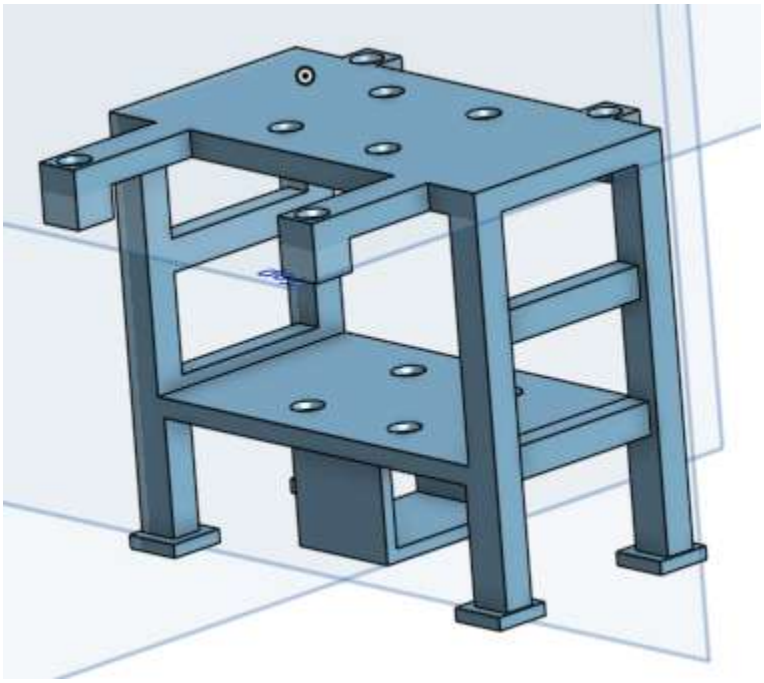


Figure 9: bottom drone mount

The bottom mount, placed directly beneath the top structure, supports the compute and control subsystems. The upper plate features two long front arms and two shorter rear arms with countersunk holes precisely aligned for mounting the Raspberry Pi 4. The lower platform includes four mounting holes for standoff-mounted installation of the 4-in-1 ESC and flight controller stack.

Four additional countersunk holes on the upper surface of the bottom mount align with the carbon fibre frame for rigid attachment. Beneath the lower plate, a printed clip secures the power module, suspending it under the frame for efficient space usage and thermal dissipation. Material was strategically removed from the sidewalls to reduce print weight, while cross-braces were added to preserve strength and prevent lateral flexing.

Propeller guards were intentionally excluded from the final design. While they offer basic protection to the propellers during collisions, their added weight significantly reduces flight time and thrust-to-weight efficiency, two of the project's key performance priorities. Additionally, the cost of propeller replacement is minimal, and the drone will only operate in environments where the pilot maintains a safe distance during flight, eliminating major safety concerns. This tradeoff reflects a deliberate decision to prioritize efficiency, maneuverability, and simplicity.

Together, the top and bottom mounts eliminate the need for discrete brackets and fasteners, streamlining the build while preserving modularity. This design approach reduces assembly complexity, supports iterative upgrades, and ensures mechanical compatibility with all onboard hardware. The integrated mount system directly supports the project’s goals of lowering form factor, improving structural efficiency, and simplifying long-term maintenance.

2.1.3 4-in-1 Electronic Speed Controllers (ESC)

To reduce wiring complexity and improve weight distribution within the constrained 6-inch frame, the team transitioned from the legacy configuration of four discrete ESCs to a compact 4-in-1 unit. The selected ESC is the HAKRC BLHeli_32 45A 4-in-1, which integrates four independent ESC channels into a single printed circuit board.

This design choice eliminates the need for four separate mounting locations and lengthy signal or power wires, freeing up interior space for cleaner component routing. The 4-in-1 ESC is positioned directly beneath the flight controller on the bottom mount using M3 standoffs. This vertical stack simplifies mechanical integration and minimizes electrical noise by shortening the signal and ground lines between the ESC and flight controller.

The ESC supports the DShot digital protocol, allowing for precise and low-latency throttle control. It also provides per-motor telemetry, including current, RPM, and temperature, all of which are compatible with PX4 firmware. The board can drive each motor with up to 45 amps of continuous current, providing ample power headroom for the VCI Spark 1404 3750KV motors, even during high-thrust operation.

Thermal testing confirmed that the ESC remains within safe operating temperatures during hover and moderate maneuvering without the need for active cooling. Heat dissipation is aided by direct airflow over the board during flight.

Overall, the 4-in-1 ESC contributes to the system’s goals of reducing weight, improving reliability, and increasing maintainability through a simplified electrical layout.

2.1.4 Flight Controller

The flight controller used in this system is the Holybro Kakute H7 Mini. It is a compact and powerful board designed for high-performance drones, offering ample processing capability and support for advanced firmware, such as PX4. The board is based on the STM32H743 microcontroller, which features a 32-bit ARM Cortex-M7, making it suitable for demanding real-time flight control tasks. It includes multiple UART ports and a dedicated SBUS-compatible input, allowing direct integration with digital receivers without the need for additional hardware. The Kakute H7 Mini weighs only 5 grams, making it the lightest flight controller on the market that can support PX4. Its small size and low weight made it the right choice to align with the client’s goal of reducing the drone’s form factor.

2.1.5 Receiver & Transmitter

A Radiolink R12DSM receiver was connected to the Kakute H7 Mini via the SBUS protocol using one of its UART inputs. The receiver was physically mounted on the frame and electrically connected to the flight controller's RX pad. It was bound to an AT9S Pro transmitter, which provided reliable manual control throughout the development process. All trims and sub-trims on the transmitter

were reset to zero to prevent drift, and channel responsiveness was confirmed using QGroundControl's radio calibration interface.

2.2 Manual Flight Software -

This section outlines the minimum software requirements for manually flying a drone.

2.2.1 PX4 Firmware on Flight Controller

PX4 is an open-source flight control firmware widely used in research, commercial, and hobby drone applications. It supports a variety of flight modes, autonomous mission planning, and hardware configurations, and interfaces seamlessly with ground control software such as QGroundControl. For this project, PX4 was chosen as a robust and configurable platform that supports both manual flight testing and future autonomous integration with the offboard computer.

Setting up PX4 on the Kakute H7 Mini required several non-standard steps due to limited support in the default firmware flashing pipeline. The board arrived preloaded with Betaflight, which prevented it from being recognized by QGroundControl. To address this, STM32CubeProgrammer was used to erase the flash memory and completely remove the existing firmware. With the board cleared, the PX4 bootloader was flashed manually using dfu-util. This command-line utility uploads firmware to microcontrollers via USB when they are in DFU (Device Firmware Upgrade) mode. This tool allowed us to bypass unsupported graphical tools and directly load the required bootloader onto the Kakute H7 Mini.

Once the bootloader was successfully installed, the Kakute H7 Mini was recognized by QGroundControl. However, the official PX4 firmware build system did not provide direct support for this board, so the PX4 source code was cloned, compiled locally, and manually flashed to ensure compatibility. After installation, QGroundControl was used to complete all necessary setup procedures, including accelerometer, gyroscope, and magnetometer calibration, ESC configuration, and motor layout verification. The process involved multiple stages but resulted in a fully functional PX4 firmware installation tailored to the Kakute H7 Mini, ready for both manual and autonomous flight integration.

2.2.2 Hardware-in-Loop Simulation

To verify the integrity of the control signal pathway (from RC transmitter through receiver and flight controller to actuator outputs), a full Hardware-in-the-Loop (HITL) simulation was established using the Radiolink AT9S Pro transmitter and an alternate flight controller. This setup provided a critical intermediary step before engaging physical motors, reducing the risk of unintended behavior during initial configuration. The workflow began with the legacy flight controller before transitioning to the Pixhawk 6C Mini, which allowed debugging and streamlining the HITL process across both hardware platforms. Notably, configuring PX4 firmware for HITL operation differed significantly from its manual flight configuration. Numerous safety checks and failsafe features, such as GPS lock, sensor

readiness, and motor interlocks, had to be bypassed or overridden to permit actuator output in simulation since PX4's assumption of real hardware when issuing motor commands.

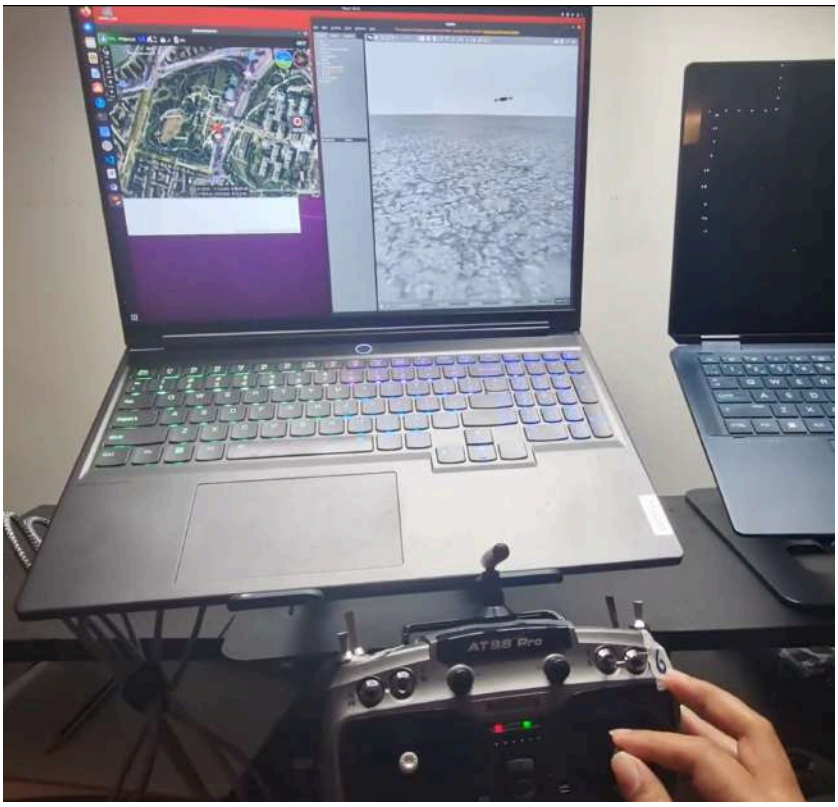


Figure 10: All drone parameters are configured and ready for manual flight

After significant debugging, which involved a custom vehicle configuration and tuning of multiple parameters, MAVROS was able to arm, command, and monitor the drone. The system reliably responded to all mapped manual inputs and successfully simulated manual flight, establishing confidence in the firmware configuration ahead of physical testing.

3.0 Autonomous Flight Subsystem

This section covers all the additional hardware and software stack required to fly the drone autonomously and safely

3.1 Autonomous Flight Hardware

This section is about the hardware to fly safely. It consists mainly of the flight computer and camera. However, there are additional boards to support the power demand required.

3.1.1 Onboard Raspberry Pi 4 and Remote Client Connection

To lighten the computational burden on the Raspberry Pi Model B (RPi–4), whose quad-core ARM Cortex-A72 CPU and 8 GB of RAM are quickly saturated by simultaneous SLAM, computer-vision, and high-rate telemetry tasks, the heavy processing pipeline was migrated to a more powerful ground-station PC. In this architecture, the RPi-4 remains the drone's real-time I/O hub, handling video streaming and flight controller control, while the PC executes resource-intensive algorithms such as Path Planners and Map Fusions.

To facilitate a split, a dedicated wireless communication link is established using 802.11ac (Wi-Fi 5) over the 5 GHz band, which offers significantly higher throughput (up to 867 Mbps on RPi-4 with supported routers) and lower latency that the 2.4 GHz band, while being less prone to interference from common household devices.

Both machines run Ubuntu 20.04 to keep package versions, environments, kernel modules, and Robot Operating System (ROS) Noetic middleware consistent, for ease of use and control. The split architecture offers several advantages:

1. Computational Headroom:
   The PC's multi-core CPU and discrete GPU accelerate perception and learning tasks that would otherwise throttle the RPi-4.
2. Thermal and Power Relief:
   Off-loading reduces on-board heat and power draw, extending flight time and allowing a lighter battery pack.
3. Rapid Iteration:
   Algorithms can be recompiled and profiled on the PC without redeploying firmware to the drone.
4. Scalability:
   Additional PCs or cloud instances can be added for swarm coordination or heavier AI workloads without modifying the aerial platform.

3.1.2 Intel RealSense Camera

The Intel RealSense D435i is a high-performance stereo depth camera featuring dual global‑shutter imagers paired with an onboard Vision Processor D4 ASIC. Each imager delivers

synchronized left/right video at up to 1280×720 resolution and 90 fps, with a 95° diagonal depth field of view and operational range from approximately 0.3 m to 3 m, making it highly suited to a tight indoor environment. Depth output is provided as 16-bit-per-pixel Z-maps (depth frames) at resolutions up to 848×480 at 90 fps, representing per-pixel distance measurements in millimeters. The Vision Processor D4 ASIC performs stereo disparity matching, IR pattern decoding, depth triangulation, and edge-preserving post-processing in real time. This dramatically reduces host CPU load and delivers dense and high-speed depth maps directly over USB 3.1.

Stereo vision provides two distinct optical viewpoints. This makes it particularly valuable for autonomous drone navigation as accurate depth cues emerge from triangulation between the left and right images. As a result, obstacle detection and scene understanding are more robust than monocular approaches, which rely solely on temporal cues or learned patterns. The D435i's wide baseline (~50 mm) improves depth estimation even on low-feature surfaces, while the high frame rate captures fast drone maneuvers with minimal motion blur. This capability was vital for seamless integration with the software stack. The dense depth maps directly inform the FUEL path planner algorithm, which enables real-time frontier detection, obstacle avoidance, and 3D map generation.

Though it is among the more expensive depth-sensing options, the RealSense camera was a justified investment as its plug-and-play depth performance significantly accelerated development time and ensured reliable autonomous operation out of the box. For this purpose, replacing this singular component would be a time-costly endeavour as the processing from the depth ASCI would have to be emulated in software. This introduces significant risk with software compatibility and synchronisation. Because of this, for the MVP, the RealSense camera is being used in the upgraded design as it would mitigate a lot of these challenges. In the future, two alternative systems are proposed in Section 4 for the replacement of this camera, as it would significantly reduce the payload weight on the drone, allowing for further downsizing and a more compact form factor.

3.1.3 Inertial Measurement Unit (IMU)

An IMU is a crucial sensor responsible for measuring the drone's motion and orientation in 3-D space. It typically contains a 3-axis accelerometer and a 3-axis gyroscope. The accelerometer measures linear acceleration along the X, Y, and Z axes, allowing the system to detect motion, tilt, and gravitational forces. The gyroscope measures the angular velocity, or rotation of the drone around each axis, which is essential for estimating its orientation, commonly referred to as roll, pitch, and yaw.

The internal IMU of the flight controller was configured at 50 Hz. Therefore, to provide a higher frequency data at 100 Hz to smoothly operate VINS for path-planning purposes, the 200 Hz Wheeltec M100 external IMU from the client's original prototype was installed on the final drone.

3.1.4 Raspberry Pi Compatible Mono Camera

The Arducam OV9782 USB Global Shutter Camera was selected as the monocular vision system responsible for recording aerial footage used in 3D reconstruction. Our application required a

camera capable of capturing colour video at a minimum of 1280×720 resolution and ideally 30 frames per second. It also needed to support a global shutter to avoid motion distortion during rapid flight, be lightweight, and run reliably on Ubuntu 20.04. The camera had to interface with the Raspberry Pi 4, either through CSI or USB.

However, global shutter colour cameras that are compact enough for a drone are extremely limited. Many options were grayscale-only, and most colour models used CSI interfaces that required the Raspberry Pi OS (Raspbian), which is incompatible with our Ubuntu-based setup. This significantly narrowed our options. Our initial choice, the Raspberry Pi Global Shutter Camera, was ultimately rejected due to its weight and budget constraints.

The OV9782 camera offered a practical solution. It uses a USB-UVC interface that works out-of-the-box on Ubuntu and is recognized as a standard webcam. It integrates easily with OpenCV. It supports 1280×800 resolution at up to 100 fps in MJPEG mode and includes features like auto-exposure, white balance, a wide field of view, and a fixed-focus lens, all of which are ideal for 3D reconstruction. The camera also provides a 68 dB dynamic range and comes in a compact 38 × 38 mm form factor, making it highly suitable drone mounting. Given all constraints this camera was deemed the most suitable option for capturing reliable aerial color footage within our platform's limitations.

3.1.5 Power Module

The power module serves two functions: it steps down the battery's voltage for the Raspberry Pi and its connected Camera modules and external IMU, which require a 5.1V 3A power supply. It also monitors the battery current and voltage to prevent critical discharge during drone operation. This component replaces the previous designs of the custom power distribution board and BMS board, see Appendix B for details on these previous solutions. The exact module used is the CRIUS Power Module 28V 90A and can be seen in figure 11 below. Additionally the Raspberry Pi 4 power requirements can be found in table 2.

Figure 11. CRIUS Power Module 28V 90A

Component	**Max Current Draw (mA)**
Raspberry Pi 4	1250
Intel RealSense - Stereo Cam	700
Arducam - Global Shutter Monocam	200
Wheeltec N100 IMU	40
Total	**2190**

Table 3: 5.1V Supply Current Draw

The Crius Power Module receives power directly from the battery and supplies it to the 4-in-1 ESC through the yellow XT60 connectors shown in Figure 11. The pins connected to the JST connector serve multiple functions. First, they provide a regulated 5V 3A output via an onboard surface-mount buck converter. Second, they deliver an analog voltage signal to the flight controller proportional to the total current drawn from the battery, using a sensing resistor circuit. This signal ranges from 0 to 5 volts, with a scaling factor of approximately 18.3 mV per amp.

After testing and verifying both the current measurements and 5V power source, it was seen that the Crius Power Module dramatically outperforms our previous custom BMS PCB and power distribution board by consolidating voltage regulation and power monitoring into a single, compact unit. This off-the-shelf component is compact, uses surface-mounted components, and is low-risk, as it requires no complex firmware design or clock signal configuration. In contrast, our previously suggested custom modules laid out in Appendix B would require more and heavier components, and debugging would be more complex due to multiple wiring connections that are prone to physical damage and are points of error. However, a custom module would provide greater customization, such as integrating Python scripts for data acquisition/presentation or extending the system with other components.

Ultimately, the team selected the Crius Power Module (6-28V, 90A). Both the team and the client agreed that for a small autonomous drone, a lighter and more compact physical footprint was more beneficial than the customization offered by custom-built modules. The compact wiring layout with the power module can be seen in figure 12.

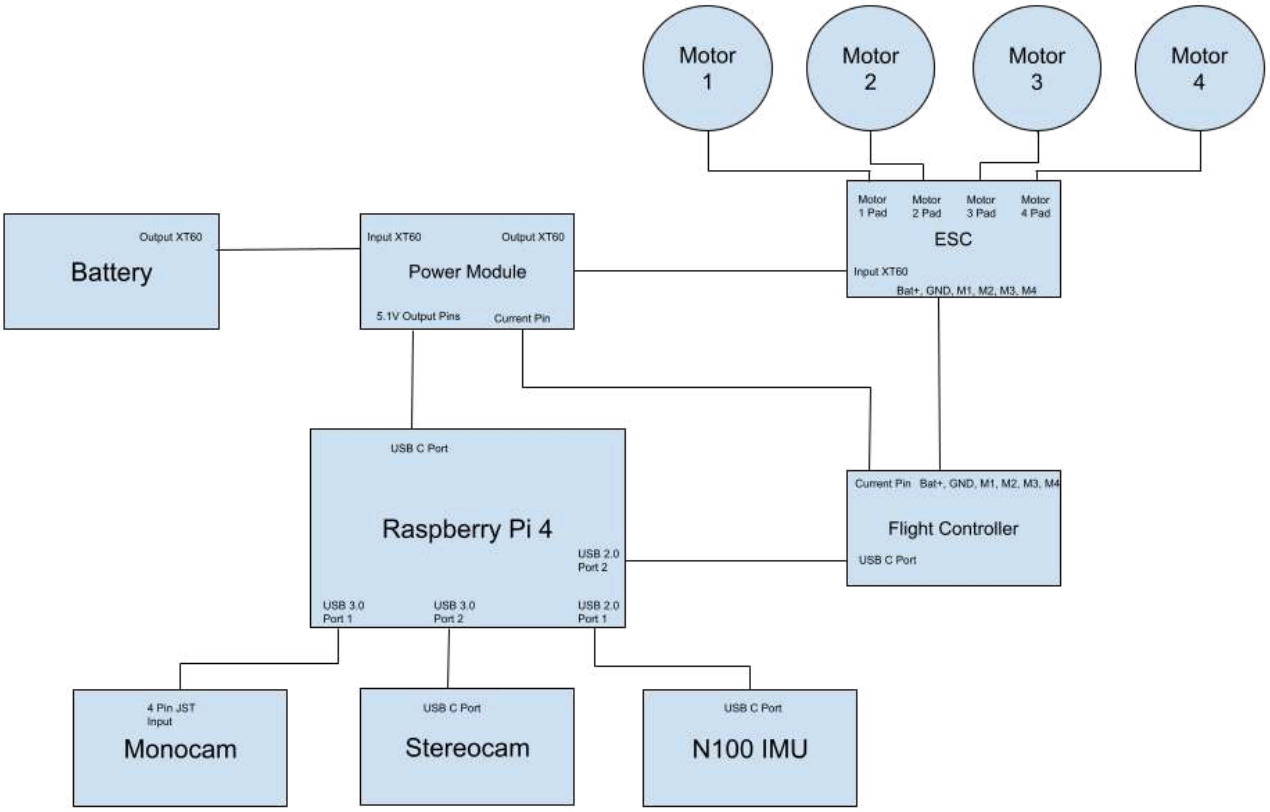


Figure 12. Wiring Diagram with Power Module

3.2 Autonomous Flight Software

This section goes over how all the software is run over ROS on the drone.

3.2.1 Autonomous Navigation Pipeline

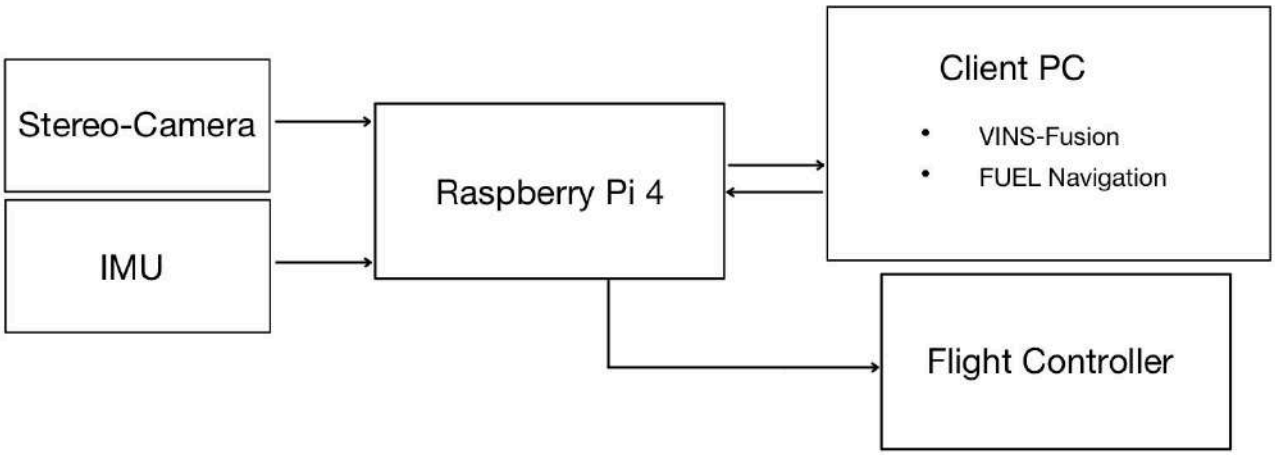


Figure 13: Software Data-flow Diagram

To effectively manage the autonomous navigation and data flow pipeline, the system architecture is structured into three core components:

1. Input - Comprising of Camera Module and Inertial Measurement Unit (IMU)
2. Compute - featuring VINS-Fusion and FUEL Navigation
3. Output - Connecting and Controlling the Flight Controller

This modular breakdown enables clear separation of sensing, processing, and control, ensuring scalability, maintainability, and robust system integration.

3.2.1.1 Input Phase

Camera Module

Enabled by the RealSense library, the D435i camera takes a dual mono8 image from its stereo IR sensors. These images are sent to the VINS-Fusion SLAM algorithm, as well as being fused into a single Z16 depth image from its internal graphics computational unit. The formulated depth image is then used by the FUEL navigation algorithm.

Inertial Measurement Unit (IMU)

Recorded using ahrs driver, synchronized with the camera input, ensuring that sensor fusion algorithms can align motion and vision properly. High-rate data allows systems like VINS-Fusion to maintain accurate and responsive localization. The IMU data, which provides the odometry–combined position and velocity data–was propagated into a pose data, as expected by the VINS-Fusion input, through a conversion algorithm.

3.2.1.2 Compute Phase

VINS (Visual-Inertial Navigation System)-Fusion

It is the central component of the drone's localization pipeline, responsible for providing accurate, real-time 6-DOF[1] pose estimation by combining stereo images with the IMU data. It implements a tightly-coupled, non-linear optimization-based approach to sensor fusion, which makes it significantly more robust and accurate.

The core of VINS-Fusion is its sliding window nonlinear optimization, which maintains a fixed-size set of recent states (camera poses, velocities, biases, and landmarks). Within this window, a bundle adjustment-like process optimizes all variables simultaneously by minimizing reprojection errors (from the visual features) and inertial preintegration errors (from the IMU).

In addition to real-time odometry, VINS-Fusion supports loop closure detection using a parallel thread that leverages the DBoW2[2] library (Bag-of-Words) matching to identify previously visited locations. When a loop is detected, a pose-graph optimization is triggered to correct the accumulated drift across the trajectory, significantly improving global consistency. This makes VINS-Fusion not just a local odometry system, but also capable of map relocalization, allowing the drone to recover from localization failure.

[1]: Degrees Of Freedom– tracks translational and rotational movement
[2]: Bag of Words– a feature tracking algorithm that tracks individual features to acquire a picture of the whole

ORB (Oriented FAST and Rotated BRIEF)-SLAM3

For future integration, ORB-SLAM3 support was added to the system. The source code can be accessed via the /feature/MonoNav branch of the project page on GitHub. This approach leverages the power of the modern ORB-SLAM3 library that can operate only on RGB-D[1] images.
Depth-Anything-V2 [5], a pre-loaded deep learning-based RGB depth prediction model, generates a depth map upon RGB image inputs through its custom ROS wrapper, and the data is consumed by the ORB-SLAM3 node to provide a pose for FUEL. Through this, the weight of the aircraft can be further reduced by replacing the stereo camera with a mono camera, ultimately achieving a longer flight time. Although the pipeline was verified on land, due to time constraints, the model could not undergo a flight test.

[1]: RGB-Depth– a combination of an RGB image with its depth mappings

FUEL (Fast UAV Exploration and Localization)

FUEL is a high-performance navigation framework designed to enable real-time autonomous exploration in unknown environments. It sits on top of the localization output from VINS-Fusion, leveraging the estimated 6-DOF pose of the drone to construct a local understanding of the environment and plan a safe, efficient path toward unexplored or goal-directed regions.

At its core, FUEL builds and maintains a local 3D occupancy map using frameworks such as Voxblox (for Volumetric mapping). As the drone moves and collects sensory data (depth images from

the Stereo Camera), the map is incrementally updated in real time. This local map is used both to identify obstacles and to detect frontiers - boundaries between known and unknown space.

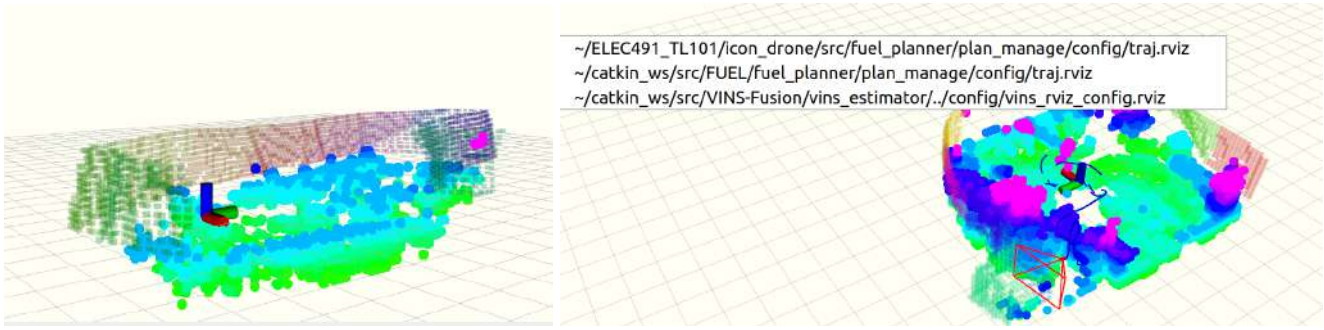


Figure 14: RViz Mapping of FUEL

FUEL then performs trajectory planning and ensures a direction that promises the most unexplored volume and a path that is collision-free using the 3D occupancy map, and continuously replans in real time to respond to dynamic environments or map updates.

3.2.1.3 Output Phase

Flight Controller (PX4) Connection and Control

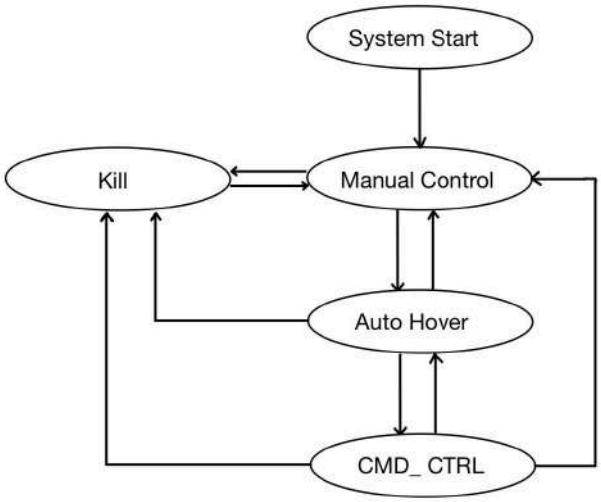


Figure 15: PX4 Controller Finite State Machine

The navigation goal posted by FUEL is interpreted as an immediate desired pose. The px4ctrl ROS node provides the parameters required by the PX4 controller for its calculation of appropriate thrust and orientation for the next movement via PWM control of each motor of the drone. Furthermore, the px4ctrl node uses a Finite State Machine and handles the remote signals received

from the controller or the client connection to switch between multiple states, including idle, manual control, auto-hover, auto-navigation, and more. Finally, the PX4 flight controller is connected through MAVROS[1] and interprets this and adjusts motor commands accordingly.

[1]: A pipeline that integrates the flight controller in the ROS network

3.2.2 Wireless Communication and ROS Node Connection

3.2.2.1 Wireless Communication

Figure 16: Remote WI-FI Connection

The wireless communication framework enabling distributed computation across the drone and ground station is implemented using a multi-machine ROS setup over a Wi-Fi network, leveraging ROS's native network transparency. In this configuration, the Raspberry Pi 4, mounted on the drone, acts as the ROS master, running the ROSCORE process and hosting the MASTER_URI. The ground-based workstation, which performs intensive computation of the navigation processing, is configured as a remote ROS slave, exporting its environment to point to the Pi's ROS master.

The Raspberry Pi onboard the drone interfaces with the stereo camera, capturing real-time images and IMU data. This data is streamed wirelessly to a remote computer, where it is processed by VINS-Fusion for state estimation and by FUEL for navigation planning. The resulting computational outputs, such as the estimated 6-DOF pose and the desired navigation goal, are then published over the shared ROS network. The Pi subscribes to these outputs, enabling it to close the control loop by feeding them into the flight controller node, which in turn commands the drone via PX4.

Keeping the wireless communication process with a latency was critical as the drone expects real-time processing for its navigation and obstacle avoidance. Thus, it is recommended to use an isolated LAN system. In testing, a 5-GHz broadband router (off the network) was used to assign network IPs to the two devices in communication. The validity of the real-time communication was checked by verifying that the data exchange happened at the required frequencies. More on this is discussed on Section 3.1 of the *Verification and Validation* Document.

3.2.2.2 ROS Node Connections and Topic Subscriptions

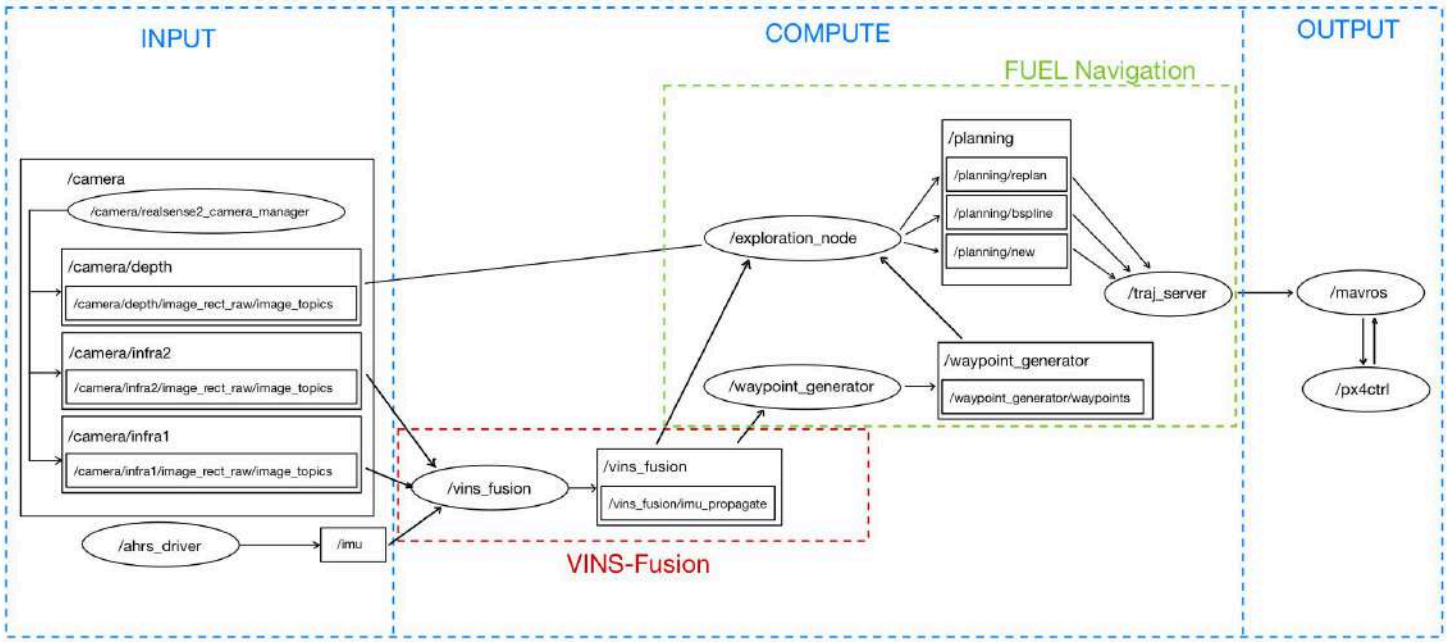


Figure 17: ROS Nodes (circle) and Topics (square) connection

Camera Module (Intel RealSense D435i)

Enabled by the RealSense Library, the Intel D435i Stereo Camera provides synchronized infrared streams (/camera/infra1, /camera/infra2) as well as a depth image (/camera/depth). These image streams are handled by the RealSense ROS driver nodes, such as:

1. /camera/realsense2_camera_manager
2. /camera/realsense2_camera

The infrared images are forwarded to the VINS-Fusion node (/vins_estimator) via topics:

1. /camera/infra1/image_rect_raw/image_topics
2. /camera/infra2/image_rect_raw/image_topics

It is recommended that the VINS-Fusion algorithm receives the input image streams at 30 fps or above. The Raspberry Pi-4 was unable to handle wireless raw image data transfers at this rate for high resolution (720p) images, and this was resolved through compression and decompression of the images. The RealSense SDK was used to produce a compressed stream of images, which, after being sent over wirelessly, were decompressed via ROS image transport and fed into VINS-Fusion as raw images.

Simultaneously, the depth data is used to construct a 3D local map for obstacle detection and frontier identification. The depth image is published to the FUEL navigation pipeline under:

1. /camera/depth/image_rect_raw/image_topics

Inertial Measurement Unit (IMU)

The IMU data is recorded using the ahrs_driver node, which publishes to the /imu topic

Visual-Inertial Navigation System (VINS-Fusion)

The /vins_estimator node is the central component of the localization stack. It ingests Stereo Camera Input (/camera/infra*) and IMU Input (/imu), then runs a sliding window optimization to estimate the drone's real-time 6-DOF pose (x, y, z, roll, pitch, yaw). Provides a precise pose estimation that is published via:

1. /vins_estimator/imu_propagate

The output of VINS-Fusion provides full odometry of position and velocity information. The result is then subscribed by FUEL's exploration manager, enabling accurate waypoint planning and map updates based on the drone's current position.

FUEL Navigation

The FUEL navigation system begins with the waypoint_generator node, which receives current pose estimates from VINS-Fusion via ROS topics. Using the depth images from the D435i camera (/camera/depth/image_rect_raw/image_topics), FUEL builds and maintains a local 3D occupancy map, identifying frontiers and obstacles.

The waypoint_generator publishes waypoints through /waypoint_generator/waypoints. These waypoints are consumed by the navigation_node, which performs real-time trajectory planning, obstacle avoidance, and dynamic replanning.

Flight Controller (PX4) Connection and Control

The final navigation goals and motion commands are published from the navigation stack to the PX4 flight controller via the px4ctrl node (via MAVROS). The output data flows into:

1. /mavros/setpoint_position/local for position control
2. /mavros/setpoint_velocity/cmd_vel for velocity control
3. /mavros/vision_pose/pose (optional) to feed VINS pose into PX4 estimator

The px4ctrl node is responsible for switching the drone between manual control, takeoff, hover, autonomous control, and landing states. The node expects appropriate state switch signals to be received, which, for the scope of the project, were configured to be transferred via an RC. During auto hover, the px4ctrl directs the PX4 flight controller to balance the aircraft in response to the odometry interpreted from the IMU signals. Then, the flight controller uses the pre-calibrated physical parameters (weight of the drone, required thrust ratio, PID) to output the necessary PWMs for each motor. The takeoff and the landing states take advantage of this hover ability to ascend or descend the drone from or to the ground. Finally, in autonomous control (cmd_ctrl) state, the flight controller rejects commands given from the RC and listens to the outputs of the FUEL navigation algorithm.

The PX4 firmware receives these setpoints and calculates the required motor outputs to execute the trajectory. Using onboard Proportional-Integral-Derivative (PID) and altitude controllers, the drone responds to navigation inputs in real time.

3.3 Autonomous Drone Control Simulation

This section describes the simulation environment developed to closely replicate the client's legacy codebase. It integrates and visualizes the key libraries and algorithms responsible for autonomous navigation. The main goal of the simulation is to test the system in different room layouts and environments. This allows us to verify that the autonomy stack performs as expected under varying spatial conditions. The simulation also serves as a benchmark for expected drone behavior. Real-world spaces can be modeled in Gazebo, where we can observe how the drone is expected to move and respond in each environment.

3.3.1 Simulation Design Decisions and Development Strategy

Before starting simulation development, a key design decision had to be made: whether to integrate a virtual simulation environment like Gazebo directly into the client's existing codebase or to develop a separate simulation based on the client's architecture. Both options had merit, but the trade-offs became clearer based on constraints such as workload distribution, access to resources, and our own personal limitations within the project timeline.

Integrating Gazebo into the client's codebase would have required significant upfront time to understand the existing code structure, write custom Gazebo plugins, and properly configure robot models and ROS-Gazebo interfaces. This path would have also required deep knowledge of simulation-specific tooling like URDFs, world files, and sensor emulation. Proficiency in these skills was beyond the scope of the available timeline.

In contrast, using existing open-source platforms and libraries would allow us to replicate the core functionality of the client's code without directly modifying it. They often have active communities and well-documented tutorials, which would enable faster iteration, easier debugging, and safer separation from the client's working system. It also aligned more closely with the core purpose of the simulation: to visualize the drone's behavior in controlled environments and estimate theoretical performance under idealized conditions.

Hardware availability also played a role. The client-provided laptop was the only system capable of handling real-time onboard computations for the actual drone. Attempting to run both the simulation and navigation development on the local host risked library mismatches and conflicts, which would risk breaking the code entirely. To avoid this, the simulation was developed separately on a personal laptop using an external 1TB SSD with a dual-boot Ubuntu setup. This setup was initially intended to support portability and allow for "plug-and-play" usage across machines. However, as will be discussed later, this decision introduced certain performance trade-offs. Particularly in simulation speed and disk I/O.

3.3.2 XTDrone as the Simulation Platform

With this understanding, the simulation development followed a clear roadmap. We began by setting up a baseline environment using the XTDrone simulation platform.

XTDrone is an open-source UAV simulation framework developed by Xiao et al. (2020) at Tsinghua University. It is built on Gazebo, PX4, and ROS, and supports a wide range of vehicle types, including multirotors, which will be our drone model. XTDrone is designed to bridge the gap between simulation and deployment. It allows developers to test high-level autonomy stacks in simulation and transfer them directly to real hardware with minimal changes. Its architecture is documented in the authors' papers, and the platform is actively maintained and extended by the open-source community.

One important challenge was that XTDrone was originally built for ROS Kinetic, while the client's legacy codebase operates on ROS Noetic. As a result, extra care had to be taken at each phase to ensure compatibility. Dependencies, launch files, and package versions were all checked and adjusted as needed. Although we were not directly simulating the client's code, our objective was to simulate the outputs of the same algorithms. As such, maintaining close alignment with the legacy ROS and PX4 stack was essential for meaningful comparison.

3.3.3 Summary of Important Algorithms & Libraries

Our client's legacy codebase relies primarily on three core components: VINS-Fusion for localization, the Intel RealSense libraries for depth sensing, and FUEL for autonomous path planning. Each of these plays a critical role in enabling onboard navigation.

VINS-Fusion (Visual-Inertial Navigation System)

- A SLAM (Simultaneous Localization and Mapping) algorithm
- Combines stereo camera images with IMU data.
- Provides real-time GPS-free localization by estimating the drone's position and motion.

Intel RealSense Libraries

- Handle stereo depth imaging using a pair of cameras.
- Estimate the distance to nearby objects, generating a 3D view of the environment.
- Critical for obstacle detection, wall identification, and mapping open space.

FUEL (Fast UAV Exploration Library)

- A high-level autonomous exploration planner.
- Uses localization and depth data to detect frontiers (unexplored regions).
- Generates smooth, dynamically feasible trajectories for safe and efficient exploration.

Although the final onboard system uses FUEL, the client's original development followed the Fast-Drone tutorial, which instead uses EGO-Planner for trajectory generation. EGO-Planner is a lightweight local planning algorithm designed for speed and responsiveness. It is described as an "ESDF-free gradient-based local planner." "ESDF" stands for Euclidean Signed Distance Field, a type of 3D map that tells the drone how far every point in space is from the nearest obstacle. This kind of map is computationally expensive to build and update. EGO-Planner avoids building this full map by computing obstacle avoidance gradients only along the specific path it's currently optimizing. This makes it much faster and more efficient for real-time replanning, especially on hardware with limited processing power.

EGO-Planner's efficiency likely made it appealing for early-stage development and onboard deployment. However, it has significant limitations. It operates with only a local view of the environment and does not maintain a global map or remember previously visited areas. As a result, it can struggle in complex layouts. For example, it might not plan a valid path through narrow doorways or across multiple rooms because it doesn't "know" what lies beyond its immediate surroundings

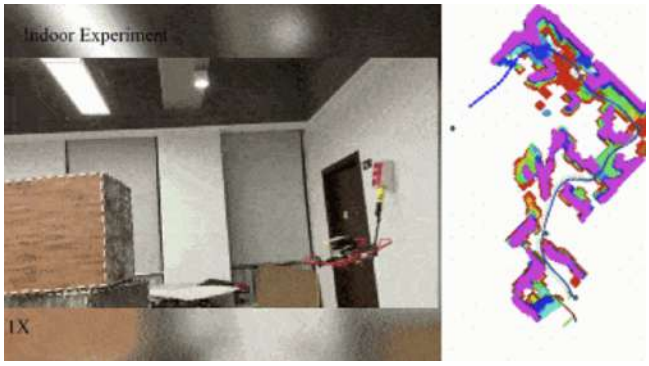


Figure 18: Image of EGO-Planner Reconstruction (taken from ZJU-FAST-Lab Github)

Likely due to these limitations, the client ultimately moved away from EGO-Planner and instead based their codebase on FUEL, which is designed for full autonomous exploration. While EGO and FUEL are distinct systems, they both build on top of the same underlying library: Fast-Planner. Fast-Planner is a software framework that provides two essential features:

1. **Kinodynamic search:** Searches for paths that obey the drone's motion constraints (such as max speed or acceleration)
2. **B-spline optimization:** a mathematical method for generating smooth curves. In this case, flight paths that the drone can follow safely and efficiently.

This shared foundation allows both EGO-Planner and FUEL to generate smooth, dynamically feasible paths with low latency, but FUEL extends the functionality with a more strategic, global planning layer.

Compared to EGO-Planner, FUEL offers several key advantages. It is designed for hierarchical exploration, meaning it plans at both a high level (which room or area to explore next) and a low level (how to fly there safely). FUEL builds and maintains a persistent map of the environment that includes both explored and unexplored regions. It actively selects new regions to explore based on what the drone has seen so far. This makes it especially well-suited for multi-room indoor exploration where long-term planning is essential.

3.3.4 Simulation Development Workflow

In its simplest setup, XTDrone supports manual drone control using keyboard input with no need for external hardware like a remote controller or receiver. Manual flight using a hardware-in-the-loop with a remote controller was already verified earlier. For our simulation, keyboard control was sufficient and avoided the risk of breaking existing code. XTDrone includes two Python scripts that allow the drone to be armed and flown directly from the terminal via keyboard control. This setup served as the simulation equivalent of manual flight on the real drone and provided the foundation for later autonomous testing.

Once manual flight was verified, each key autonomy library was integrated step by step. The first three components were supported by existing XTDrone tutorials, which made the process more approachable. However, successfully compiling and building the workspace still required multiple rounds of troubleshooting due to differences in library versions and minor syntax changes between ROS Kinetic and Noetic.

- VINS-Fusion for real-time localization using stereo visual-inertial odometry
- Intel RealSense depth sensing paired with RTAB-Map for 3D dense reconstruction, EGO-Planner, as supported by XTDrone's documentation, and compatible with its simulation setup
- FUEL, integrated using an open-source project by zhongtianda as the primary reference

While not overly complex, the final step of integrating FUEL was more involved than the previous components and required additional attention to detail. While the FUEL framework itself is well-documented and self-contained, integrating it with the XTDrone-based simulation required a clearer understanding of how its ROS nodes communicate and how its planning modules are triggered. Unlike earlier steps, there is no fully packaged simulation or tutorial to follow. Instead, a combination of project portfolios and partially complete GitHub repositories was referenced. Often, they lacked consistent structure, documentation, or were in a different language entirely.

3.3.5 Final Simulation Description

The simulation took place in a 3D indoor environment from XTDrone's built-in world models. The default Iris drone model from Gazebo was used with no changes to its shape or configuration. This model is pre-integrated with PX4 firmware, which is also used on the real drone. As a result, control commands sent through ROS and MAVROS were routed to PX4 in the same way as on actual hardware. This pipeline made it possible to verify that commands were properly processed and that simulated flight behavior was consistent with what would be expected from the actual drone.

To simulate depth perception, the drone's default stereo camera was replaced with the iris_realsense plugin. This plugin mimics the Intel RealSense sensor. It outputs a simulated RGB video stream and depth data using the same software libraries as in the physical system.

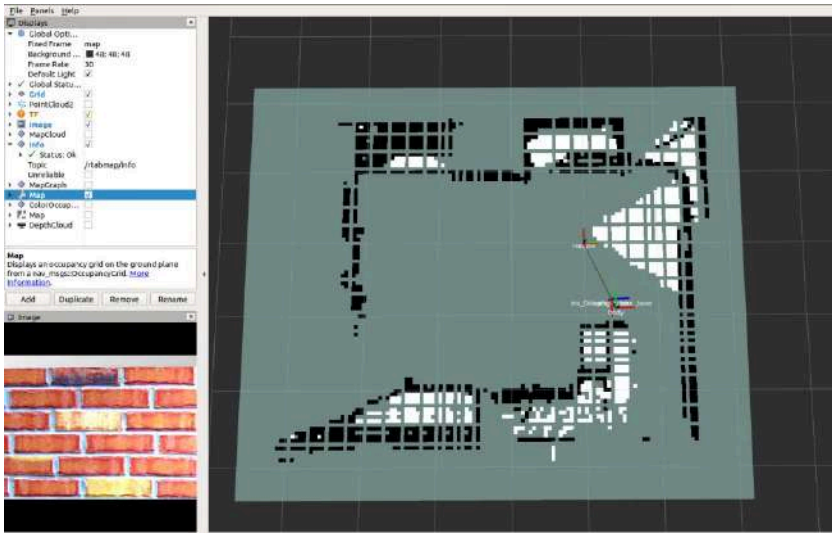


Figure 19: Correct Mapping Post Exploration (taken from XTDrone Manual)

With this setup in place, the simulation followed a structured execution sequence, designed to mirror how the real drone would be operated during autonomous flight:

1. **Gazebo World Initialization**
   The simulation launches using a Gazebo launch file. This initializes the 3D indoor environment and spawns the Iris drone model.
2. **MAVROS Bridge**
   The MAVROS node starts to bridge communication between ROS and the PX4 flight stack. It enables ROS-based software to control the virtual drone.
3. **Manual Takeoff via Keyboard Control**
   The keyboard_control.py script runs to manually arm the drone, lift off, and hover at a user-defined height.

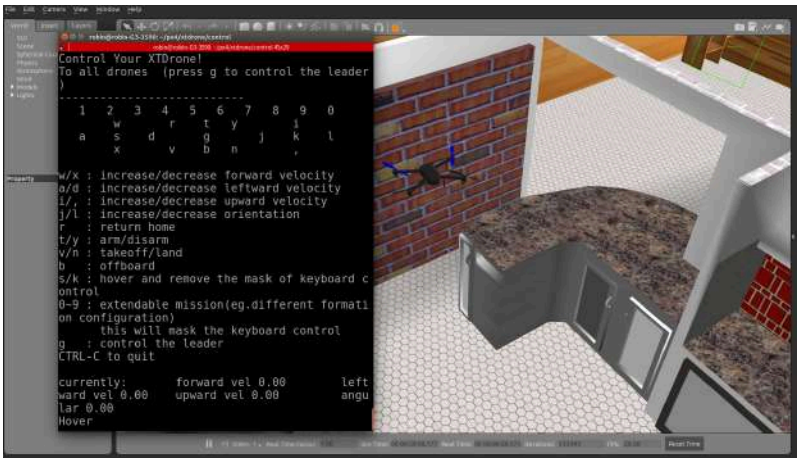


Figure 20: Keyboard Commands for Manual Drone Control on Gazebo (taken from XTDrone Manual)

4. **VINS-Fusion for Localization**
   The VINS-Fusion node launches and processes stereo images and IMU data from the simulated RealSense. It publishes position estimates to the /vins_estimator/pose topic. These pose estimates serve as the drone's current position for autonomous planning.

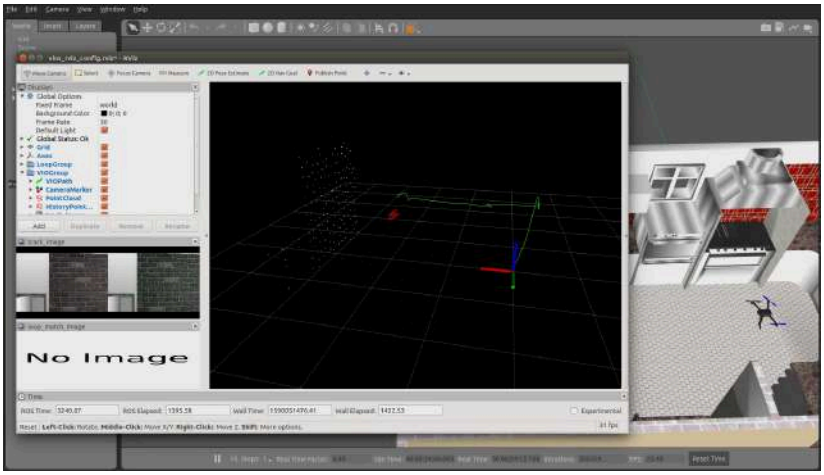


Figure 21: RViz Path Planning Visualization (taken from XTDrone Manual)

5. **RViz for Monitoring**
   RViz optionally launches to monitor simulation outputs in real time. It displays the drone's path, the room's point cloud reconstruction, and the depth camera feed. This helps verify that localization and obstacle detection function correctly

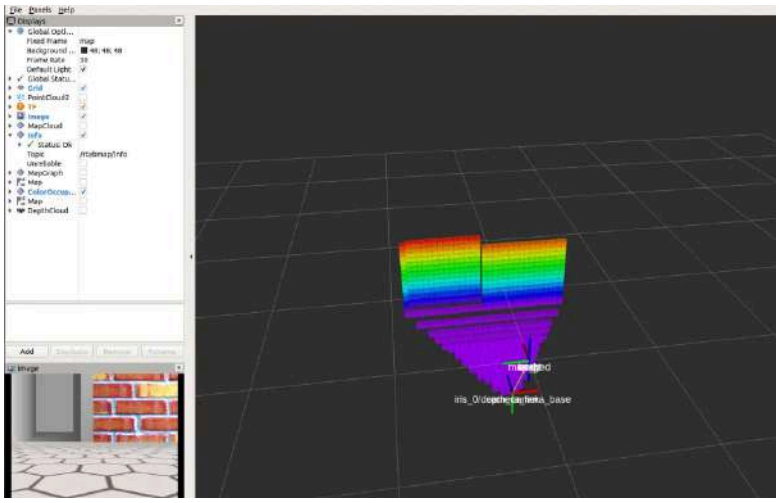


Figure 22: Depth Mapping Output (taken from XTDrone Manual)

6. **Autonomous Planning with EGO or FUEL**
   Once the drone stabilizes in hover, either EGO-Planner or FUEL launches and takes over command of the drone. It subscribes to localization data, computes a trajectory, and publishes control commands to ROS topics. MAVROS listens to these topics and routes the commands to PX4, which then executes them on the drone.

3.3.6 Performance of Autonomous Navigation Algorithms

This section evaluates the observed behavior and performance of the two tested navigation algorithms, EGO-Planner and FUEL, within the simulation environment. This is a summary of all the testing done, which can be found in the Verification and Validation Document.

EGO-Planner

EGO-Planner was tested first due to the availability of a tutorial. The setup involved copying the provided directory, making it executable, and launching the planner as a ROS node. However, the node was written for an older version of ROS. This mismatch caused several core packages to fail during the build process. Some components compiled, but others could not locate the required files during linking. Configuration files had to be modified to correctly export targets and link dependencies across the workspace. Multiple rounds of troubleshooting were required before the build succeeded.

Once operational, EGO-Planner took over after the drone was manually armed and hovering. Initial results showed promise. The drone was able to chart out a small portion of the room. However, localization was frequently lost. Obstacle detection was inconsistent. In several runs, the drone crashed or became stuck in corners.

When RViz was used to display the depth map, system lag became severe. In some trials, visualization had to be disabled entirely. This exposed the hardware limitations of the simulation setup, particularly with respect to CPU and disk I/O.

These issues are aligned with the known limitations of EGO-Planner. The planner does not maintain a global map and can miss narrow passages like doorways. However, the planner had been shown to work in the same simulation environment in the original tutorial. This indicated that while the system compiled, underlying errors or unresolved mismatches were likely still present at runtime.

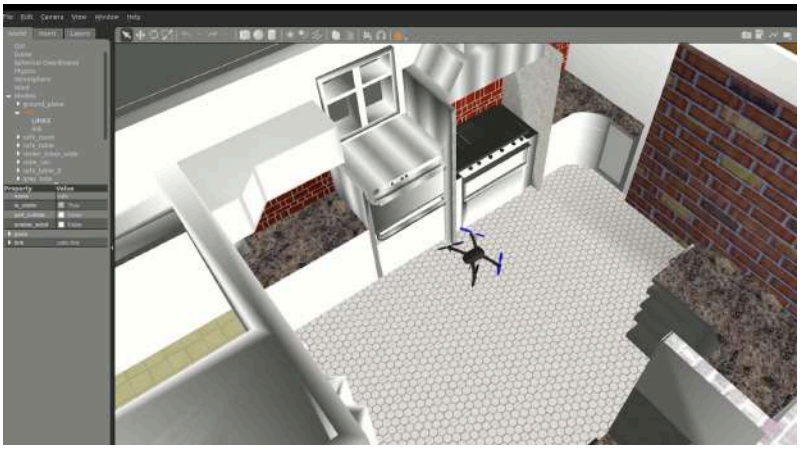


Figure 23: Drone in Tight Space that Ego Planner Mapped

FUEL

FUEL was tested after EGO-Planner. Confidence in its performance was slightly lower due to EGO's inconsistent behavior and growing hardware limitations. The setup involved using the previous EGO-Planner trajectory code along with a branched version of FUEL sourced from the open-source project by GitHub user zhongtianda. This version differed from the currently maintained FUEL repository. The rest of the XTDrone simulation environment was not provided, so the integration required several manual adjustments. These included verifying topic names, checking ROS node connections, ensuring dependency version compatibility, and modifying both the ROS launch files and CMake configurations to align with the XTDrone simulation stack.

When launched, FUEL occasionally showed signs of exploration behavior. In some instances, the drone appeared to initiate path planning and attempted to move toward unexplored areas. However, the majority of tests resulted in erratic flight. The drone often rose too high, flew outside the boundaries of the modeled world, or crashed into obstacles. In many runs, no clear path planning occurred at all.

The simulation lagged under FUEL's load, particularly when RViz was used for monitoring. These symptoms suggested that the system was approaching its computational limits. The simulation ran on an external SSD in a dual-boot configuration, which introduced I/O and performance bottlenecks. This design choice affected runtime behavior. However, the drone's erratic flight patterns were likely not caused by lag alone. A purely performance-based issue would be more consistent and predictable in failure. The irregularity of FUEL's behavior suggests deeper issues in configuration or integration.

3.3.7 Future Considerations

The next primary step is to investigate and resolve the integration issues with FUEL. While the framework launched successfully and initiated exploration in some cases, its behavior was largely erratic and inconsistent. Several targeted strategies could help isolate the cause:

- **Run the simulation on a more powerful machine** with Ubuntu installed natively on internal storage. The current setup with an external SSD over USB introduces I/O bottlenecks that could be contributing to dropped frames and synchronization issues.

- **Ensure full compatibility of dependencies** across all components. This includes verifying the OpenCV version, ROS messages, and any custom branches used in the FUEL integration. Inconsistent or stale installations may still be affecting runtime stability.

- **Establish a working baseline with EGO-Planner.** Establish a working baseline with EGO-Planner. Once EGO runs reliably, the ego_transfer script can be examined more closely. This script bridges the position output from MAVROS with the input expected by EGO or FUEL. It transforms the pose, adjusts the orientation and frame, and publishes the data at a steady rate to the correct topic. A stable EGO simulation would help verify if this link is working as intended. If FUEL itself is functioning correctly, then this transfer script may be the missing link in the XTDrone integration.

Originally, the final goal of the simulation was to generate a custom 3D room within Gazebo that matched the real-world testing environment. This would have enabled benchmarking of the autonomous algorithm in a controlled space. Metrics like time taken to fully explore the room, quality of the generated depth map, and completeness of trajectory coverage would provide valuable insight into the algorithm's performance under realistic constraints.

If the integration issues with FUEL can be resolved, revisiting this final test plan would offer a strong validation step for the autonomy stack.

4 Summary

The TL101 project focused on creating a smaller, more efficient version of ICON Lab's original drone. Key developments included hardware redesign, improved power management, and a shift to offboard processing using a Raspberry Pi and external PC. Manual flight was implemented and tested, with a custom frame, flight controller, and 4-in-1 ESC integrated into a compact 6-inch platform. Autonomous navigation was developed and using the RealSense D435i, VINS-Fusion, and FUEL, the autonomous navigation software was tested virtually but could not be successfully implemented due to unresolved integration issues. If timeline constraints allowed for it the next step towards integration would be PID tuning of the PX4 controller running on the Raspberry Pi 4 to verify autonomous hovering and navigation.

To support testing, a simulation environment was built using the XTDrone platform. This allowed the team to run the full autonomy stack in a virtual indoor environment and evaluate navigation algorithms without risking hardware. EGO-Planner and FUEL were both tested, but simulation performance was inconsistent and limited by hardware constraints and integration challenges. The end result is a semi-functional platform with clear documentation and a foundation for future development. Proposed upgrades, including alternate cameras and compute modules to further reduce size and improve performance, are outlined in Appendix C.

Appendix A. Contributions

Document Section	Major Content	Minor Content	Author	Review
[number & title]	[Initials]	[Initials]	[Initials]	[Initials]
NA. Abstract	RJ		RJ	
1.0 High-Level Overview				
1.1 Existing ICON Drone Architecture	NK		NK	PK, RJ
1.2 TL101 MVP Architecture	NK		NK	PK, RJ
1.3 MVP Drone Subsystems	NK		NK	PK, RJ
2.0 Manual Flight Subsystems				
2.1 Manual Flight Hardware				
➢ *2.1.1 Motors, Propellers, and Battery Selection*	RBC	ST	RBC	RJ, ST
➢ *2.1.2 Frame/Mounts*	ST	RBC	ST	RJ, RBC
➢ *2.1.3 4in1 Electronic Speed Controllers (ESC)*	ST	RBC	RBC	RJ ST
➢ *2.1.4 Flight Controller*	ST	RBC	ST	NK, RJ
➢ *2.1.5 Receiver & Transmitter*	ST	RBC	ST	NK, RJ
2.2 Manual Flight Software				
➢ *2.2.1 PX4 Firmware on Flight Controller*	ST	RBC	ST	NK, RJ, ST
➢ *2.2.2 Hardware-in-Loop Simulation*	NK		NK	RJ, RBC
3.0 Autonomous Flight Subsystems				
3.1 Autonomous Flight Hardware				
➢ *3.1.1 Onboard Raspberry Pi 4 and Remote Client Connection*	AS	RJ	AS	NK
➢ *3.1.2 Intel RealSense Camera*	NK	AS, RJ	NK	RJ
➢ *3.1.3 Inertial Measurement Unit*	AS, RJ		AS	RJ, NK
➢ *3.1.4 Raspberry Pi Compatible Mono Camera*	NK	AS	NK	RJ, AS
➢ *3.1.5 Power Module*	RBC, ST		RBC, PK	PK, RJ
3.2 Autonomous Flight Software				
➢ *3.2.1 Autonomous Navigation Pipeline*	AS, RJ		AS, RJ	RBC, NK
➢ *3.2.2 Wireless Communication and ROS Node Connection*	AS, RJ		AS, RJ	RBC
3.3 Autonomous Drone Control Simulation				
➢ *3.3.1 Simulation Design Decisions and Development Strategy*	NK		NK	RBC

➢ *3.3.2 XTDrone as the Simulation Platform*	NK		NK	RBC
➢ *3.3.3 Summary of Important Algorithms & Libraries*	NK		NK	RBC
➢ *3.3.4 Simulation Development Workflow*	NK		NK	RBC
➢ *3.3.5 Final Simulation Description*	NK		NK	RBC
➢ *3.3.6 Performance of Autonomous Navigation Algorithms*	NK		NK	RBC
➢ *3.3.7 Future Considerations*	NK		NK	RBC
4.0 Summary	RBC		RBC	
Appendix B				
Power Distribution Board	RBC	PK		PK, NK
Battery Management Board	RBC, PK		PK	RJ
Appendix C: Upgraded Drone Architecture Proposals				
4.1.3 Re-explore Shift to Raspberry Pi Zero	AS	RJ	AS	ST
4.2 Mono-Camera Only Alternative System	RJ	AS	RJ	NK
4.3 Stereo-Camera Only Alternative System	NK		NK	RJ

Appendix B. Power Distribution Board and BMS Board Design Alternative

Power Distribution Boards

While the drone's motors, flight controller, and electronic speed controllers can be powered directly by the battery, there are several devices required for autonomous flight, including the STM32 used for the BMS, the Raspberry Pi 4, and other sensors that require either 3.3V or 5V power supplies. To address this, two custom buck converter circuits based on the LM2678 switching regulator were designed. This IC comes in both a 3.3V and 5V fixed output voltage model, the LM2678-3.3 and LM2678-5.0.

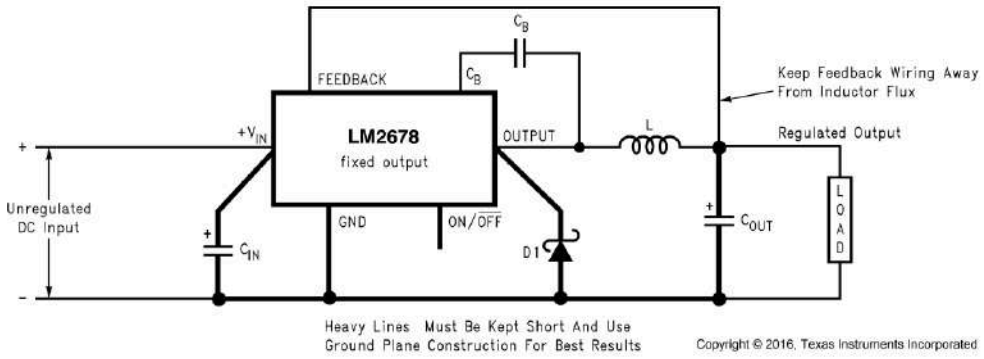
To specify the component size for the 5V and 3.3V supplies, the current draw of all loads was tallied, which can be found below. Based upon the maximum current draws, the 3.3V regulator was specified to provide a 1A output, and the 5V regulator was specified to provide a 3A output, both fitting well within the LM2678's maximum current output of 5A.

3.3V Device	Max Current Draw (mA)
BMS - STM32	10
BMS - Current Sensor	60
Thermal Sensor	200
Monocamera	400
Total	**670**

5.0V Device	Max Current Draw (mA)
Raspberry Pi 4	1250
Stereo Camera	700
Total	**1950**

3.3V and 5.0V Current Draw

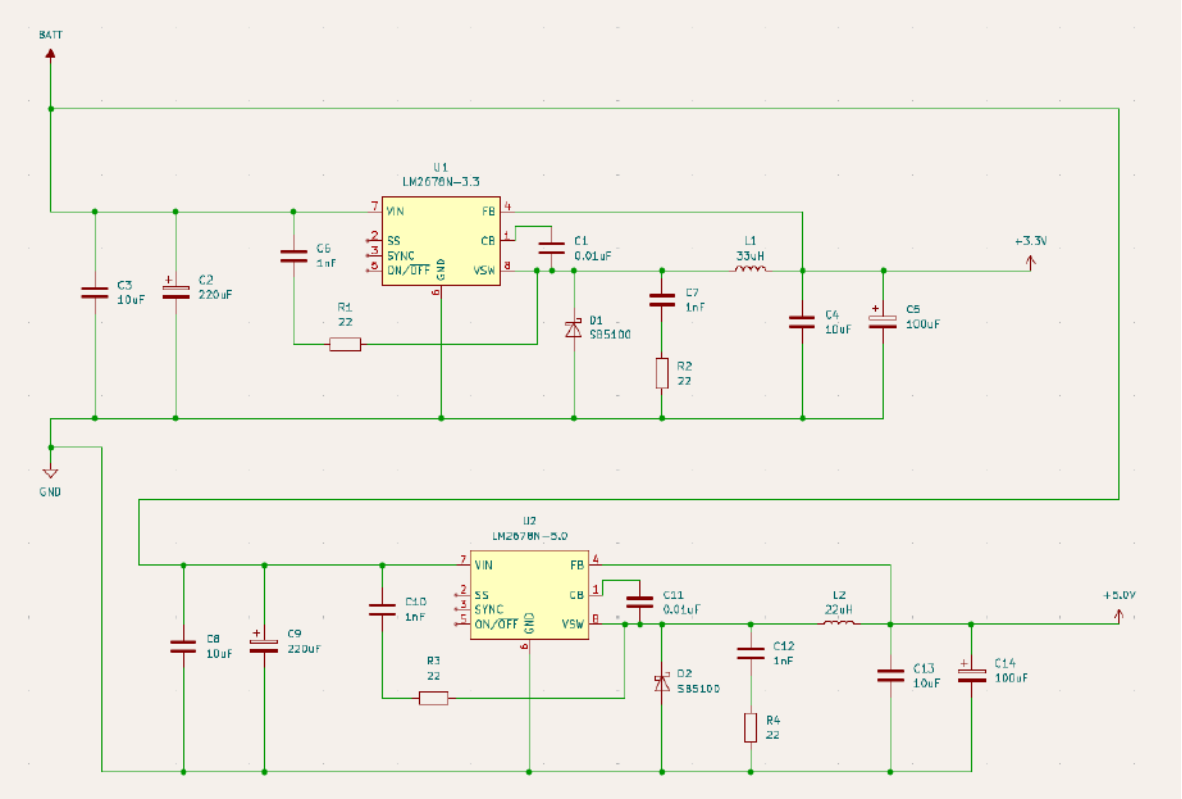
The LM2678 buck converter is manufactured by Texas Instruments, and they provided a basic circuit layout shown below as well as a detailed guide on component selection based on the maximum input voltage and output current.



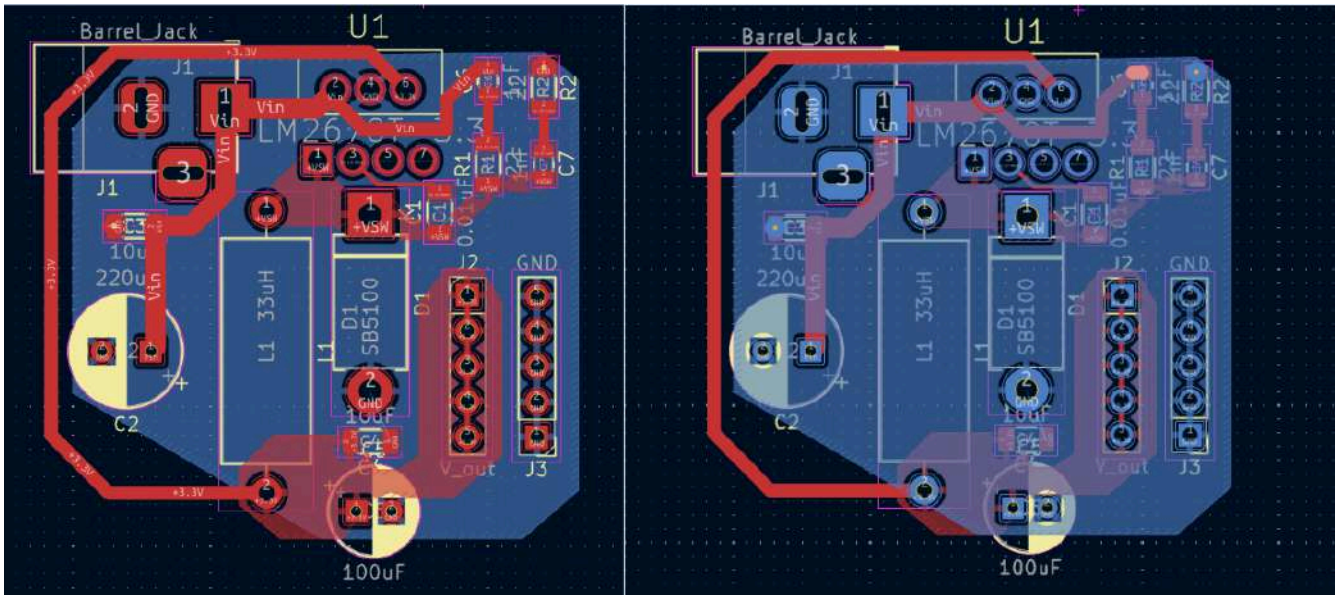
LM2678 Basic Circuit for Fixed Output Voltage

The basic circuit was constructed for both the 5V and 3.3V models, and regulation under light loads was successful across the full battery voltage range(12.0V to 16.8V). However, under heavier loads, severe spikes in the output voltage were observed due to high-frequency ringing of the switching node directly on the output pin of the LM2678. The ringing became more pronounced as the load increased and was traced to the parasitic inductance of the circuit. See the *Verification & Validation* document for example waveforms showing this ringing behaviour.

To address the issue of switch node ringing, two RC snubbers were used, one placed between Vin and the switching node and another between the switching node and ground. The snubbers act as low-pass filters, with the capacitors absorbing the high-frequency oscillations on the rising and falling edges. Ringing was observed on both the rising and falling edges of the switching node waveform, each requiring a dedicated snubber to provide an appropriate damping path. The switching node to ground snubber suppresses high-frequency oscillations on the rising edge, and the Vin to switching node snubber suppresses the oscillations on the falling edge. This dual-snubber approach significantly stabilized the switching waveform and output voltage under load, giving a constant output of 3.3V±5% at 1A and 5.0V±5% at 3A, respectively. See the complete circuit diagram and proposed PCB Layout below.



Power Distribution Board Circuit Diagram



Proposed PCB Layout for single buck converter board

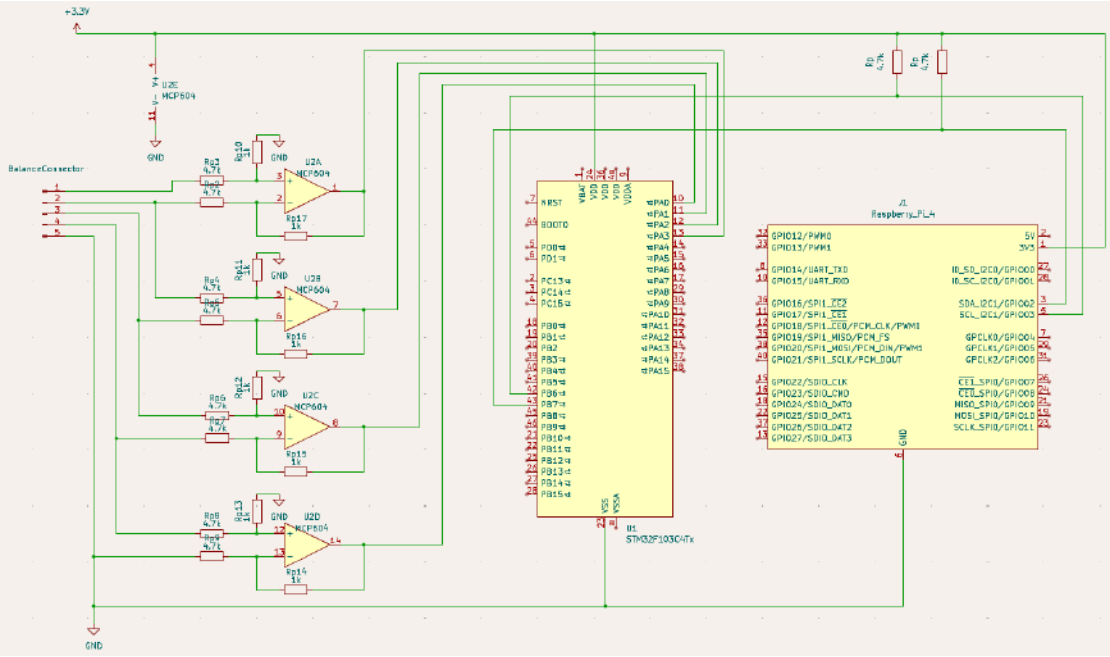
Battery Management Board

The Battery Management System (BMS) works to optimize a drone's power usage by monitoring individual battery cell voltage discharge and controlling operational parameters. Battery cells cannot drop below 3.0V; this can cause damage and lead to fire [1]. To mitigate this, the BMS signals the drone to halt autonomous flight when a cell's voltage reaches 3.6V. The 0.6V difference provides a safety margin and does not affect flight, as it accounts for 5% of the cell's capacity.

The current BMS firmware design uses an I2C communication protocol between an STM32F103 microcontroller and a Raspberry Pi 4. The STM32F103 was chosen over alternatives like the ESP32 due to its power consumption and customizability. The I2C protocol was selected for its versatility in addressing multiple devices and its implementation ease compared to alternatives.

The I2C is achieved by creating a 2-wire bus and integrating a ROS (Robot Operating System) node for data exchange. Communication between the STM32F103 and the Pi 4 transmits 20-byte messages containing sensor data. Each message includes 2 floats (4 bytes each, 8 bytes per message) representing voltages from four battery cells and a flag. The message format can be customized to accommodate the addition of new sensors.

The BMS reads data from individual battery cells through analog-to-digital (ADC) pins and sends a bit-flag once the reading drops below the desired value. To read the battery cells, a subtractor circuit was introduced between the ADC pin and the input voltage, as the battery design has an additive circuit that outputs the cumulative voltages across the cells. The op-amp MCP6024 was selected considering the circuit's operating voltage of 3.3V and the accessibility of four op-amp circuits within the IC, matching the number of cells to monitor. Given that the IC has all the required units, it was deemed the best option compared to others. Operating temperature is typical for similar ICs. The circuit diagram can be found in below.



BMS Board Circuit Diagram

To implement the circuit and finalize a prototype, a proto-board was used to solder the components and microcontrollers. Component placement was strategic to reduce the total surface area of the board, minimizing both the physical size and weight of the drone. As the integrated-circuit unit provides a parallel configuration of the op-amps, the circuit is soldered in parallel as well. This design balances performance, power efficiency, and scalability, with delineation of responsibilities: the STM32F103 handles sensor data acquisition and low-level control, while the Pi 4 processes and displays the data via the ROS node.

Appendix C Upgraded Drone Architecture Proposals

This section lays out our proposal for replacing certain parts of the drone to make it smaller These components help us make the drone smaller and more form factor efficient

Re-explore Shift to Raspberry Pi Zero

The current drone platform utilizes the Raspberry Pi 4, which meets the system requirements but results in a very tightly constrained hardware layout due to its size and heat footprint. To address this, the feasibility of using a Raspberry Pi Zero will be revisited. Although the Pi Zero was initially ruled out due to its limited processing power and poor performance in real-time video streaming, further investigation into the requirements of VINS-Fusion revealed that continuous video streaming is not essential. Instead, VINS-Fusion can operate effectively using discrete image frames for pose estimation and navigation. This insight reopens the possibility of leveraging the Pi Zero as a lightweight alternative. Given its smaller form factor and reduced power consumption, it will be quickly re-evaluated as a potential replacement for the Raspberry Pi 4 in the upcoming Milestone 4 phase.

Mono Cam Only Alternate System

To further reduce the overall weight of the final drone system, a streamlined approach was proposed: using a single image sensor to perform all necessary tasks onboard. Currently, two separate modules are required—one for depth estimation via a stereo camera and another for high-quality video capture used in post-flight 3D reconstruction. The goal is to consolidate these functions into a single camera system.

While an RGB camera alone is sufficient for video recording—especially with motion compensation—it does not provide the depth information required by most SLAM algorithms, including VINS-Fusion, which is currently in use for this project. However, this limitation can be addressed through deep learning-based depth inference.

In the proposed setup, a frame is captured by a single RGB camera and passed through a deep learning model to estimate its depth map. Depth-Anything-V2 was selected for this task due to its open-source availability, reliable performance, and fast inference capabilities. The resulting depth map is then converted into metric scale using the intrinsic parameters of the camera. Once both the RGB frame and its corresponding metric depth are obtained, they are fed into a SLAM algorithm that supports RGB-D input. Finally, the output of the SLAM is fed into the pathfinding algorithm, FUEL, unchanged.

ORB-SLAM3 was chosen as the target SLAM system because it is open source and integrates state-of-the-art tracking and mapping technologies. This pipeline enables the drone to perform both accurate localization and 3D scene reconstruction using only a single lightweight monocular camera.

Multiple cameras have been tested and calibrated to date to validate the reliable inference of distance metrics from a monocular lens, even under high-speed motion. This approach highlights the

potential to replace traditional, hardware-intensive solutions with a more efficient and cost-effective alternative, maximizing software capabilities to overcome physical limitations.

Stereo Cam Only Alternate System

As part of our ongoing efforts to optimize the drone's onboard hardware for autonomous navigation and data capture, the drone architecture is constantly being reevaluated. Presently, integral components for the autonomous drone pair the Intel RealSense D435i camera and Raspberry Pi 4. While this setup functionally satisfies both the depth mapping needs of the FUEL algorithm and the stereo vision requirements of VINS-Fusion, its physical size has proven problematic for mounting on the compact 6-inch frame. The Raspberry Pi 4 is capable of both powering and interfacing with the RealSense, but it carries excess peripherals, I/O ports, and size constraints that are not required in flight.

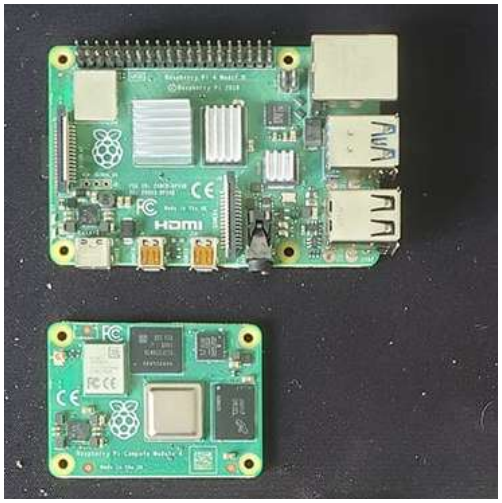
An earlier design considered using a single monocular camera paired with ORB-SLAM3, a well-documented SLAM algorithm capable of monocular visual-inertial odometry (See Section above). This approach offered advantages in weight and simplicity and was a valid alternative to the existing stereo pipeline. However, the proposed stereo-based system emerged due to two key factors:

- The existing architecture (VINS-Fusion and FUEL) was already built around stereo inputs and depth maps, which would require a full pipeline change to accommodate monocular SLAM.
- Given current timing constraints, the team opted to retain stereo compatibility to leverage prior work and avoid starting from scratch with a new SLAM integration.

(Left) Compute Pi 4 Module with Onboard External Antenna. (Center) WaveShare Nano Base Board for Peripherals. (Right) 1MP Global Shutter Stereo Camera.

Building on this recommendation, the dual 1MP global shutter stereo camera became the core of the proposed upgrade. This module serves a similar role to the earlier monocular camera idea — enabling simultaneous video recording and navigation — while meeting the client's requirement for global shutter imagery, which minimizes motion distortion and improves footage quality for 3D reconstruction.

This camera is also significantly more cost-effective than the RealSense D435i, while still providing synchronized stereo imagery. Importantly, it has much lower power requirements, which opens the door to using the Raspberry Pi Compute Module 4 (CM4) as the onboard computer. The CM4 offers the same processor as the full Pi 4 but removes unnecessary peripherals, includes better Wi-Fi capabilities, and has a dramatically smaller form factor. This would be a major advantage for integration on the 6-inch drone frame.



Comparison of Size of Raspberry Pi 4 (Top) and Compute Pi 4 (Bottom)

However, unlike the RealSense, this stereo camera lacks an onboard depth processor. The RealSense outputs depth maps in real time using dedicated hardware and an IR projector, which FUEL's mapping system depends on for occupancy grid generation and ESDF updates. Replacing this would require generating depth in software.

As part of this proposal, we are exploring the use of software-based stereo depth estimation, such as stereo_image_proc in ROS or OpenCV's stereo matching algorithms. These tools can compute disparity maps from the stereo feed and convert them into depth maps or point clouds, effectively emulating the RealSense's depth stream within the existing pipeline. This would allow FUEL's modules to continue operating without structural changes.

That said, this approach represents a significant shift in the software stack. It introduces new computational demands on the CM4 and would require careful tuning to ensure real-time performance. At this stage, the design remains a proposal and has not yet been finalized. It will only be pursued if sufficient progress is made on the current minimum viable product and more thorough consideration is given to the alternative mono-camera system. Further testing will determine whether this system should be formally recommended to the client for procurement.

5.0 Glossary

- **4-in-1 ESC**: An electronic speed controller unit that consolidates four motor drivers into a single PCB to reduce wiring complexity.
- **6-DOF**: Six degrees of freedom; describes motion in 3D space along x, y, z axes and rotation in pitch, roll, and yaw.
- **ADC (Analog-to-Digital Converter)**: Converts continuous voltage signals into digital values for microcontroller processing.
- **AHRS Driver:** Software that reads data from IMUs and provides orientation estimates (attitude heading reference system).
- **Airframe:** The structural framework of the drone that holds motors, payload, and electronics.
- **Altitude Hold:** A flight controller function that maintains the drone at a consistent height automatically.
- **Application Specific Integrated Circuit (ASIC):** A custom-built chip in the RealSense camera that performs real-time depth calculations.
- **Arming:** The process of enabling motor control; disarming is used to prevent accidental spinning.
- **Autonomous Flight:** Unmanned drone operation using algorithms without manual control input.
- **Battery Management System (BMS):** Monitors battery health and voltage; prevents deep discharge or overuse.
- **Betaflight:** A firmware platform often pre-installed on FPV drones; needs to be erased for PX4 usage.
- **Buck Converter:** An electrical component that steps down higher voltage to a lower regulated output.
- **DFU Mode:** Device Firmware Upgrade mode allows low-level firmware flashing on STM32 boards.
- **Depth Estimation:** Determining distance to objects from images, crucial for navigation and SLAM.
- **Depth Image:** A 2D array where each pixel contains a distance value from a depth camera.
- **Disparity Map:** Used in stereo vision to represent pixel differences, which are used to compute depth.
- **Dual-arm Bracket**: A mount used to vertically secure the Raspberry Pi or other boards inside the drone frame.
- **EMAX Formula Series:** A high-performance ESC model rated at 45A, used in the drone.
- **ESC (Electronic Speed Controller):** Takes PWM inputs and drives brushless motors using 3-phase AC signals.
- **FUEL:** A path planning algorithm that uses 3D maps for safe, efficient exploration.
- **Failsafe:** A software routine that triggers safe landing or hovering on signal or battery loss.
- **Field of View (FOV):** The angular range a camera can capture, affecting mapping quality.
- **Flight Controller:** Central processor that manages drone sensors, stabilization, and motor commands.
- **Flight Stack:** All software and firmware modules that run on the drone's flight controller.
- **Flyaway:** A failure mode where the drone loses control and escapes intended airspace.
- **Frontier Detection:** The SLAM-based identification of unexplored areas in the environment.

- **GPS Lock:** Sufficient satellite connection to provide reliable positional data.
- **Global Shutter:** A camera feature where all pixels are exposed simultaneously to avoid motion blur.
- **Ground Control Station (GCS):** Software like QGroundControl used to monitor and configure the drone.
- **Gyroscope:** Sensor in IMU measuring angular velocity along pitch, roll, and yaw axes.
- **Hover Throttle:** The throttle percentage at which the drone maintains level flight without climbing.
- **I2C:** A digital communication protocol used between microcontrollers and peripherals like the BMS.
- **IMU (Inertial Measurement Unit):** A device measuring linear acceleration and angular velocity for motion tracking.
- **Intel NUC:** A compact PC platform used for onboard SLAM and path planning computations.
- **KV Rating:** Indicates a motor's RPM per volt without load. Higher KV = faster but less torque.
- **Latency:** The delay between command issuance and observed effect; important in real-time systems.
- **LiPo (Lithium-Polymer) Battery**: High energy density power source used in drones, typically with 3–5 cells.
- **MAVROS:** A ROS node that connects PX4 with the ROS ecosystem over MAVLink.
- **Monocular Camera:** A single-lens camera used for vision; lacks depth unless processed.
- **Odometry:** Estimation of a robot's movement over time using sensor data.
- **Offboard Mode:** A control mode where the drone is commanded by an external computer via ROS.
- **PX4:** An open-source flight control firmware used for autonomous or manual drone control.
- **Payload:** Any component or sensor added to the drone that is not part of basic flight.
- **Pixhawk 6C**: An advanced flight controller hardware that runs PX4 or ArduPilot firmware.
- **Pose:** Position and orientation of the drone in 3D space, usually 6-DOF.
- **Power Distribution Board (PDB):** Distributes voltage from the battery to all other drone components.
- **QGroundControl:** GUI software to configure, calibrate, and monitor drones running PX4.
- **Quadcopter:** A four-motor drone configuration commonly used for aerial robotics.
- **R9DS:** A Radiolink receiver model that supports SBUS communication with flight controllers.
- **ROS (Robot Operating System):** A middleware framework used to organize and interconnect drone software modules.
- **ROS Node:** A single process in a ROS system that performs a specific function (e.g., navigation).
- **RPi (Raspberry Pi 4):** A compact Linux computer used onboard for computation and sensor interfacing.
- **Radiolink AT9S Pro:** A 12-channel RC transmitter used for manual drone control.
- **RealSense D435i:** Intel stereo depth camera with integrated IMU and depth-processing ASIC.
- **Receiver:** The drone-side unit that receives control signals from the transmitter.
- **SBUS:** A digital RC communication protocol allowing many channels over a single wire.
- **SLAM (Simultaneous Localization and Mapping):** An algorithmic framework that builds a map while estimating the robot's position.
- **STM32:** A microcontroller family from STMicroelectronics, commonly used in flight controllers and BMS boards.

- **Setpoint:** A desired value (e.g., position, velocity) sent to the flight controller for execution.
- **Snubber:** A circuit used to dampen high-frequency switching noise in power electronics.
- **Stereo Camera:** A dual-lens camera setup used to infer depth through image disparity.
- **Telemetry:** Real-time data transmitted from the drone to a ground station or onboard logger.
- **Throttle:** Control signal that adjusts the speed of the motors.
- **Trajectory Planning:** The generation of a path for the drone to follow safely and efficiently.
- **UART:** Universal Asynchronous Receiver-Transmitter; a protocol used for serial communication.
- **Ubuntu 20.04:** The Linux distribution used for both onboard and offboard systems.
- **VINS-Fusion:** A visual-inertial SLAM algorithm combining camera and IMU for pose estimation.
- **Vision Pose Topic:** A ROS topic providing position and orientation from vision sensors.
- **Voltage Regulator:** An electronic component that ensures consistent voltage to sensitive components.
- **Waypoint:** A target position in 3D space the drone is commanded to reach.
- **Wi-Fi 5 (802.11ac):** The wireless standard used for communication between drone and offboard host
- **Z-Map**: A type of depth map where each pixel value represents distance in mm.

6.0 References

[1] A. V. N.C., A. R. Yadav, D. Mehta, J. Belani, and R. Raj Chauhan, “A guide to novice for proper selection of the components of drone for specific applications,” *Materials Today: Proceedings*, vol. 65, pp. 3617–3622, 2022, doi: https://doi.org/10.1016/j.matpr.2022.06.187.

[2] B. C. Florea and D. D. Taralunga, “Blockchain IoT for Smart Electric Vehicles Battery Management,” *Sustainability*, vol. 12, no. 10, p. 3984, May 2020, doi: https://doi.org/10.3390/su12103984.

[3] STMicroelectronics, "STM32F103x8 STM32F103xB Medium-density performance line Arm®-based 32-bit MCU with 64 or 128 KB Flash, USB, CAN, 7 timers, 2 ADCs, 9 communication interfaces," *Datasheet*. Available: https://www.st.com/resource/en/datasheet/cd00161566.pdf

[4] Nanda Kumar BVN and A. Agrawal, “Framework to Drone Part Selection, Verification and Fabrication of UAV for Multiple Applications,” pp. 105–110, Dec. 2023, doi: https://doi.org/10.1109/mosicom59118.2023.10458791.

[5] L. Yang, B. Kang, Z. Huang, Z. Zhao, X. Xu, J. Feng, and H. Zhao, "Depth Anything V2," *arXiv preprint* arXiv:2406.09414, 2024. [Online]. Available: https://arxiv.org/abs/2406.09414.

[6] Texas Instruments, *LM2678 SIMPLE SWITCHER® 5A Step-Down Voltage Regulator*, Datasheet, Rev. H, Oct. 2016. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm2678.pdf

[7] X.T. Drone Project, “dense_reconstruction,” XTDrone Manual (in English), accessed: Aug. 5, 2025. Available: https://www.yuque.com/xtdrone/manual_en/dense_reconstruction; “XTDrone,” XTDrone GitHub Repository – README.en.md, accessed: Aug. 5, 2025. Available: https://github.com/robin-shaun/XTDrone/blob/master/README.en.md

[8] Z. Tianda, px4_gazebo_fuel, GitHub repository, accessed: Aug. 5, 2025. Available: https://github.com/zhongtianda/px4_gazebo_fuel

Validation and Verification Document

Client: Harry Zhang (Icon Lab)

Autonomous Drone Hardware Upgrade Project

TL 101: Arunark, Peter, Rylan, Nabiha, Ryan and Spencer

Date Last Updated: August 5, 2025

Table of Contents

1.0 Overview **3**
2.0 Manual Mode Testing **3**
2.1 Mechanical Tests 3
Component Mount Clearance Check 3
Frame Structural Integrity Test 3
Frame Size and Weight Constraint Verification 4
2.2 Electrical Tests 4
Power Module Current Sensing Validation 4
ESC Configuration and Motor Rotation Check 6
2.3 Software Configuration Tests 8
Flight Controller Sensor Output Verification 8
RC Input Channel Mapping 8
Low battery safety testing 9
2.4 Summary of Manual Mode Testing 9
3.0 Autonomous Mode Testing **11**
3.1 Wireless Communication Tests 11
Wireless Data Link Baseline Transfer Test 11
3.2 Codebase Configuration & Software Build Tests 12
3.3 ROS Nodes & Data Flow Tests 13
SLAM Pose Estimation Visualization 13
Depth Sensing 13
IMU Verification 14
ROS Node Graph Connectivity Verification 14
ROS Node I/O Behavior Verification 14
3.4 Autonomous Flight Testing 15
3.5 Video Footage Capture 16
Script-Based MJPEG Video Capture 16
Raspberry Pi Frame Capture via MJPEG Stream 17
Python Script for Video Recording with Full Autonomy Stack 18
3.6 Summary of Autonomous Mode Testing 18
4.0 Simulation Testing **20**
4.1 Manual Flight Control 20
Keyboard Arming and Flight Control 20
4.2 VINS-Fusion Visual Inertial Odometry 20
VINS Node Initialization and Odometry Publishing 20
Trajectory Coherence with Environment 21
4.3 RTAB-Map Depth Map 21
Point Cloud Generation and Topic Verification 21
Room Geometry Accuracy in Point Cloud 22
4.4 EGO-Planner 22
EGO-Planner Input Topic Verification 22
Autonomous Room Traversal with EGO-Planner 23
5.0 Project Requirement Testing **24**
Appendix A. Contributions **25**
5 Glossary **26**
6 References **29**

1.0 Overview

This Verification & Validation Plan outlines the process for evaluating whether the redesigned drone system meets its technical requirements and fulfills its intended use. Verification will ensure the system aligns with specifications through tests focused on size, weight, sensor integration, power performance, and communication. Validation will assess the drone's effectiveness in real-world indoor environments, based on practical testing and user feedback. Together, these steps will confirm that the final system is functionally reliable and suitable for ICON Lab's inspection applications.

2.0 Manual Mode Testing

This section details all the tests that were completed to ensure a functioning manual drone capable of being piloted with a remote controller and a human pilot. These tests were conducted only after the drone was fully built and configured, and were repeated iteratively at various stages to ensure a reliable and stable platform for eventually merging the autonomous software onto it.

2.1 Mechanical Tests

Test ID: MANU-VER-MECH-01

Component Mount Clearance Check

Ensure all mounted components maintain safe clearance from propellers and structural limits.

- Install all hardware components on the 6-inch frame
- Manually rotate propellers and observe clearances
- Arm the drone and observe no interference in the propeller spin

Expected Result: No mechanical interference or contact between propellers and components

Outcome

All components were securely installed on the 6-inch frame and did not interfere with the propeller sweep. Manual rotation of the propellers confirmed clearances under static conditions. After arming the drone, all four motors were spun up individually using QGroundControl, and no collisions or vibration-related movement of mounted components were observed.

Test ID: MANU-VER-MECH-02

Frame Structural Integrity Test

Assess the vibration tolerance of the drone frame under typical flight and handling conditions.

- Perform a shake and tap test
- Run motors at 50% on the test stand for 60 seconds
- Inspect for loose parts or resonance
- Run all flight testing and ensure long-term frame integrity

Expected Result: No visible frame deformation, component shift, or audible vibration issues

Outcome

During initial testing, vibration issues were observed during motor spin-up, which caused several electrical hardware mounting bolts to loosen. This indicated that the initial fastening method was insufficient for vibration tolerance. To address this, fastening washers were added to all electrical

hardware mounts, significantly improving retention. After the fix, the drone passed the same vibration tests without any bolts loosening or components shifting. No audible resonance or frame-related mechanical instability was observed. Additionally, after a large volume of cumulative flight testing, the frame showed no signs of bending, cracking, or mechanical degradation, confirming its long-term structural reliability.

Test ID: MANU-VER-MECH-03

Frame Size and Weight Constraint Verification

Objective: Verify that the frame size does not exceed the 8-inch diagonal constraint and achieves a reasonable weight-to-thrust ratio.

Test Procedure:

- Measure the frame diagonally from motor to motor after complete assembly
- Weigh the full assembled drone
- Calculate the thrust-to-weight ratio using the total weight and maximum thrust

Expected Result: Measured size ≤ 8 inches, and the drone has a weight-to-thrust ratio around 2.

Outcome

The fully assembled drone was measured diagonally from propeller tip to propeller tip. The total span was 7.25 inches, meeting the 8-inch maximum size constraint set by the client. The drone weighs approximately 660 grams, and total maximum thrust across all four motors was measured at 1308 grams, resulting in a thrust-to-weight ratio of 1.98. This confirms that the drone complies with both physical and performance-based constraints, ensuring suitability for controlled indoor flight.

2.2 Electrical Tests

Test ID: MANU-VAL-ELEC-01

Power Module Current Sensing Validation

Validate power module battery monitoring capabilities.

- Connect the power module and flight controller to an external power supply with adjustable current output.
- Apply a range of resistive loads and record the CUR pin voltage using an oscilloscope.
- Calculate volts per amp to determine the module's current sensing scale factor.
- Connect the CUR pin to the flight controller and review PX4 telemetry logs.
- Replace the power supply with a 4S LiPo battery and perform flight tests at hover throttle.
- Confirm real-time current monitoring matches expectations during active flight.

Expected Result: CUR pin voltage scales linearly with input current and matches expected telemetry values when monitored through PX4. During flight testing, live current readings fall within the expected operating range of 15–20 A during hover.

Outcome:

To calibrate the power module's current output, we conducted a full sweep of current levels from 1.4 A to 12 A using an external power supply and a set of known resistive loads. At each test point, we measured the CUR pin voltage and computed the corresponding amps per volt (A/V) value. This

produced a non-linear profile, especially at low currents, where the A/V ratio varied from approximately 23 to 19 A/V. However, beyond 8 A, the readings began to stabilize, averaging around 18.3 A/V, as shown in Figure X. This stable region aligns well with the drone's actual current draw during flight.

The purpose of this full characterization was to ensure that current sensing would remain accurate in the range that matters, namely, during flight at hover throttle and during normal load conditions. The drone operates primarily between 15–20 A, which falls entirely within the stable portion of the curve. Therefore, we used the average A/V value of 18.30 A/V as the scaling factor for interpreting PX4 telemetry data.

Once calibrated, the CUR pin was connected to the flight controller. We streamed current readings via PX4 and confirmed that telemetry logs matched the expected values generated by the power supply. After validating the signal chain, we switched to battery power and conducted live flight testing. During hover, the drone consistently drew 15–20 A, matching expectations and confirming that the power module provides reliable current sensing in real-world conditions.

Current(A)	Mean (mV)	A/V
1.4	61	22.95081967
2.04	96.13	21.22126287
3.03	150.8	20.0928382
3.705	187.4	19.77054429
3.95	203.1	19.44854751
4.54	235.8	19.25360475
5.95	315.9	18.83507439
6.2	329.6	18.81067961
8.008	434.8	18.41766329
8.9	488.5	18.21903787
9	494.2	18.21125051
9.4	514	18.28793774
10	543.8	18.38911364
11.3	615.8	18.35011367
12	658	18.23708207
	Average	18.30174269

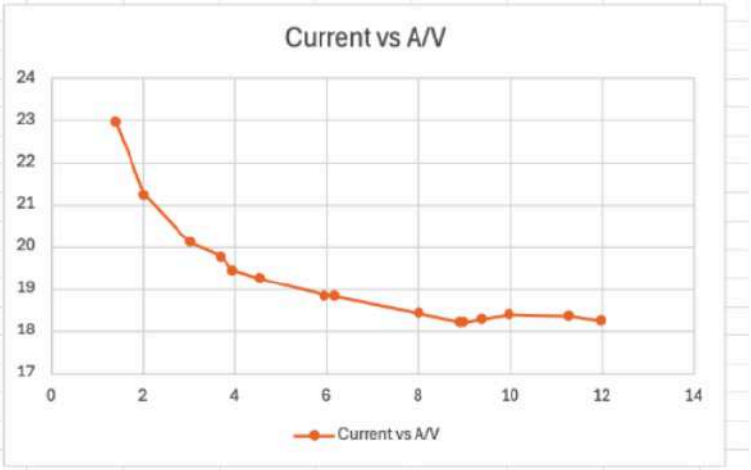


Figure X. Plot of current vs. A/V (amps per volt) measured from the CUR pin. The response is non-linear at low currents but stabilizes above 8 A. The shaded region corresponds to the drone's operating current during hover.

Test ID: MANU-VER-ELEC-02

Power Module 5.2V Output Validation

Validate the power module's 5.2V supply capabilities

- Connect the power module input to a lab bench power supply at 16.8V
- Measure the 5.2V output under no-load conditions
- Apply a range of resistive loads and gradually drop the resistance until a 1.67 Ohm load is connected using power resistors to verify 5V output at partial and full load conditions (3A)

Expected Result: The 5.1V Output of the power module should be stable within 5.1V ± 5% independent of the load applied to it.

Outcome:
The power module output was very stable at 5.1 V under no load conditions, as the load increased to the maximum load of 3 A, slight variations could be observed, see figure XX for the full load waveform. Under full load conditions, the minimum voltage is 5.000 V, the maximum is 5.280 V, and the mean is 5.165 V. This confirms that even under full load conditions, the output of the 5.1 V power supply is ±3.53% which is well within our specifications of ±5%

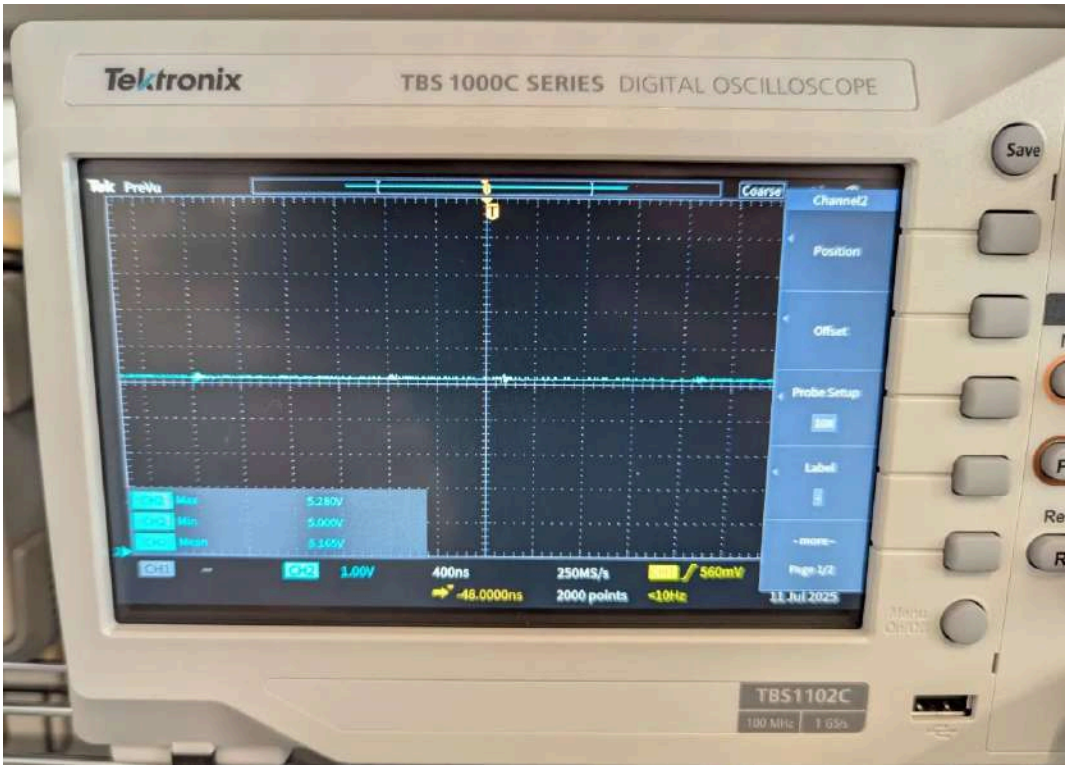


Figure XX. 5.1V Power Module Output Under Full Load Conditions

Test ID: MANU-VER-ELEC-03

ESC Configuration and Motor Rotation Check
Verify ESC firmware is functional and motors spin in the correct direction

- Use QGroundControl flight firmware configuration software to spin motors individually
- Confirm directions match the PX4 motor map

Expected Result: All four motors respond correctly and rotate per config (CW/CCW)

Outcome:
The motors were first spun individually using the actuator test function in QGroundControl to confirm ESC responsiveness and correct channel mapping. Initial spin directions were observed and then adjusted using the motor direction reversal feature available with DShot 600. This allowed motor rotation to be configured entirely in software without physically rewiring any phase leads. After the

update, all four motors rotated according to the PX4 standard layout (alternating CW and CCW), confirming proper ESC configuration and motor direction setup.

Test ID: MANU-VER-ELEC-04

RC loss failsafe test

Verify that when communication with the remote controller is lost, the drone can land safely

- Power on the drone and establish RC connection.
- Arm the drone and take off into a stable hover
- Turn off the transmitter to simulate an RC link loss event.
- Observe the drone's behaviour immediately following signal loss.

Expected Result: Upon RC link loss, the drone should automatically enter Land Mode, descend at a controlled rate, and disarm upon touching down.

Outcome:

Multiple tests were conducted to validate RC loss behaviour. Initial tests were performed at low altitude, just above the ground, to ensure safety in case of unexpected behaviour. Once a consistent, safe response was observed, the drone was tested at a full hover height of 1 meter. In all cases, powering off the transmitter resulted in a reliable transition to Land Mode. The drone performed a controlled vertical descent and disarmed upon touchdown. These results confirm that the failsafe system responds appropriately to RC signal loss and can safely recover without pilot intervention.

Test ID: MANU-VER-ELEC-05

Flight Time Test

Verify the drone's flight time from a fully charged battery voltage of 16.8V to a fully discharged battery voltage of 13.0V

- Fully charge the drone battery to 16.8V
- Arm the drone in manual mode and take off roughly 1 foot above the ground, and allow the drone to hover
- Continue to hover with the drone while monitoring the battery voltage until it reaches 13.0V, then land and disarm the drone
- Collect the flight log

Expected Results: Based on our flight time estimations with a 658g drone with a 2200mAh battery, we expect our total flight time to be roughly 8.5 minutes

Outcome

The battery voltage and current data were collected from the drone and plotted in Figure XX. The total flight time achieved was 7 minutes and 35 seconds. While this is lower than initially expected, it can be attributed to the actual measured capacity of the battery during operation. When the battery reached its fully discharged state at 13.0 V, it had delivered approximately 1950 mAh, despite being rated for 2200 mAh. This indicates that the battery provides only 88% of its advertised capacity. If flight time projections were adjusted to reflect the usable 1950 mAh instead of the nominal 2200 mAh, the observed flight duration aligns closely with expectations.

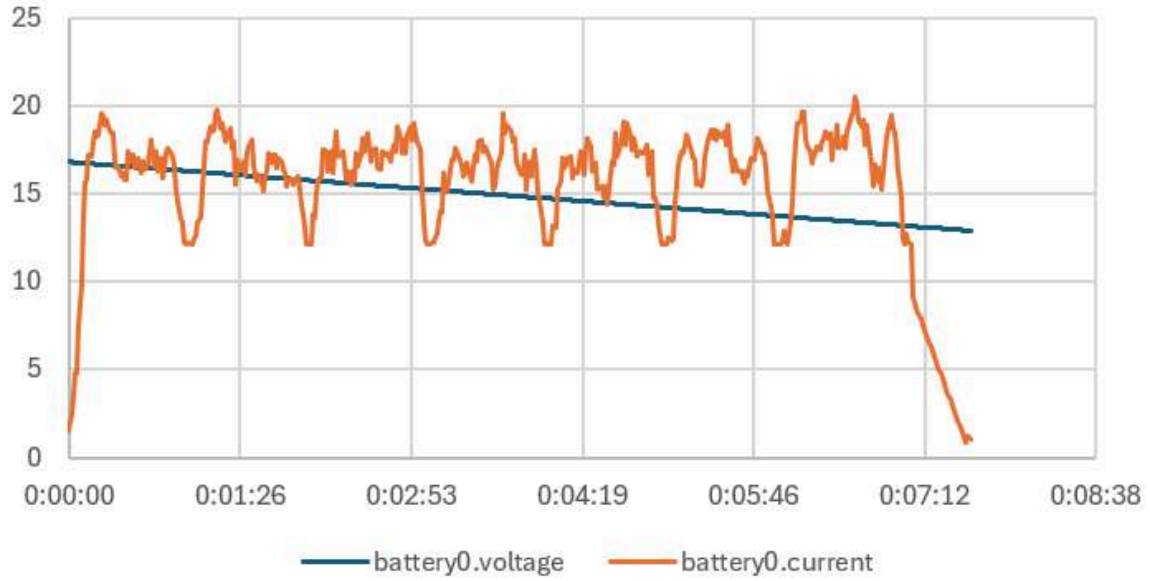


Figure XX. Battery voltage (V) and current levels (A) throughout the hover flight time test.

2.3 Software Configuration Tests

Test ID: MANU-VER-SW-01

Flight Controller Sensor Output Verification

Ensure IMU sensors initialize and provide valid data

- Connect the flight controller to QGroundControl via USB.
  Power on the drone and open the "Sensors" tab in QGroundControl.
- Move the drone through different orientations and observe accelerometer and gyroscope outputs.
- Verify consistent and smooth sensor response across all axes.

Expected Result: Smooth, responsive readings across axes

Outcome: Sensor output was verified by connecting the Kakute H7 Mini to QGroundControl and moving the drone in all three axes. The accelerometer and gyroscope plots responded smoothly and consistently in real time. No discontinuities, spiking, or dropped readings were observed. Sensor calibration was performed successfully, and the system was confirmed to deliver valid IMU data for flight control and autonomous estimation.

Test ID: MANU-VER-SW-02

RC Input Channel Mapping

Confirm RC transmitter sticks and switches are mapped correctly

- Bind the Radiolink R12DSM receiver to the AT9S Pro transmitter.
- Connect the receiver to the flight controller via the SBUS port.
- Power on the drone and open the "Radio" tab in QGroundControl.
- Move transmitter sticks and switches while observing channel response.

Expected Result: Throttle, yaw, pitch, roll on CH1–CH4; kill switch on CH11 and arm/disarm CH8.

Outcome: The transmitter and receiver were successfully paired, and SBUS communication with the Kakute H7 Mini was verified through QGroundControl. Stick inputs are mapped correctly to channels

1–4, with throttle, yaw, pitch, and roll displaying expected behaviour. A two-position kill switch was mapped to channel 11, and an arm/disarm switch was assigned to channel 8. Both were confirmed to function as intended in QGroundControl and during bench tests. All control inputs were responsive, and channel ranges were confirmed to operate within expected limits.

Test ID: MANU-VER-SW-03

Low battery safety testing

Verify that low battery conditions are detected and communicated to the operator in real time, prompting appropriate landing action before battery depletion.

- Power on the drone and connect it to the local host computer.
- Begin a flight using a fully charged 4S 2200 mAh LiPo battery.
- Monitor battery voltage throughout the flight via the PX4 telemetry stream displayed on the local host.
- Observe system behaviour as voltage reaches 14.0 V and again at 13.5 V.

Expected Result:

At 14.0 V, the operator prepares to land. At 13.5 V, the voltage is considered critically low, and the operator must land the drone immediately.

Outcome:

During test flights, battery voltage was continuously displayed in real time on the local host through PX4 telemetry. The messages published on /mavros/battery were tracked via a ROS terminal by the operator. As the battery voltage dropped to 14.0 V, the operator began preparing for landing. When the voltage reached 13.5 V, the operator immediately landed the drone to prevent unsafe operation. The operator relied on active monitoring of the telemetry display.

An automatic failsafe landing procedure was initially considered, but implementation required a GPS module, which was not available in the current configuration. Attempts to bypass this requirement were unsuccessful. As a result, manual monitoring was used. This method successfully ensured the drone landed safely before reaching dangerously low voltage levels, validating the effectiveness of the defined battery management protocol.

2.4 Summary of Manual Mode Testing

All manual mode tests were completed, ensuring the redesigned drone is safe, responsive, and mechanically stable for indoor flight under human control. Initial mechanical testing confirmed that all components were securely mounted with sufficient clearance from moving parts. Minor vibration-related fastening issues were observed early but were resolved by adding washers to all electrical hardware mounts. Long-term flight testing showed no structural degradation, confirming the frame's mechanical resilience.

The frame was verified to fall within the client’s size constraints at 7.25 inches from propeller tip to tip, and the drone achieved a thrust-to-weight ratio of 1.98, providing strong and stable lift for controlled indoor operation. The power module underwent detailed calibration to compensate for

non-linear response at low currents, and real-time current sensing was verified via PX4 logs during live flight. Hover current draw was measured at 15–20 A, confirming operation within the calibrated sensing range. Under typical hover and maneuvering conditions, the drone achieved a measured flight time of 7.5 minutes using the 2200 mAh 4S battery.

Electronic speed controllers were configured via QGroundControl using the DShot 600 protocol, allowing motor direction to be reversed in software without rewiring. All motors were verified to rotate correctly per PX4 configuration. The flight controller IMU sensors were tested and confirmed to produce stable and valid readings. Manual control was further validated by mapping RC channels to the flight stack, with throttle, yaw, pitch, and roll on channels 1–4, arm/disarm on channel 8, and a kill switch on channel 11.

These manual mode tests established a reliable flight platform that satisfies the client's physical and performance requirements and supports safe integration of autonomous features in later stages.

3.0 Autonomous Mode Testing

This section details all the testing conducted for autonomous flight, which was built upon a sufficiently tested manual drone. The process included subsystem-level tests, standalone software tests, individual algorithm validation, and tests of multiple subsystems working together. These incremental evaluations culminated in final integrated system testing to ensure the full autonomous stack functioned reliably in deployment conditions.

3.1 Wireless Communication Tests

Test ID: AUTO-VAL-NET-01

Wireless Data Link Baseline Transfer Test

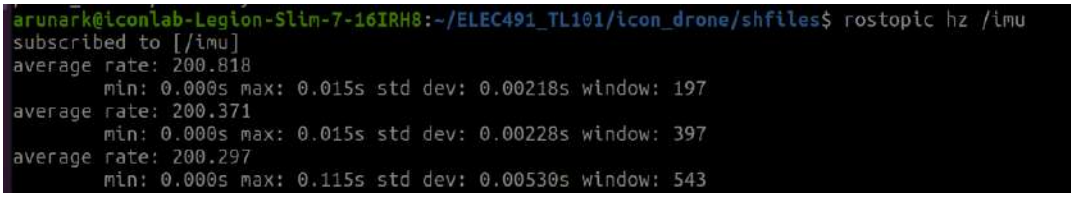
Validate that the wireless communication channel supports sustained, low-latency data transfer under basic network conditions

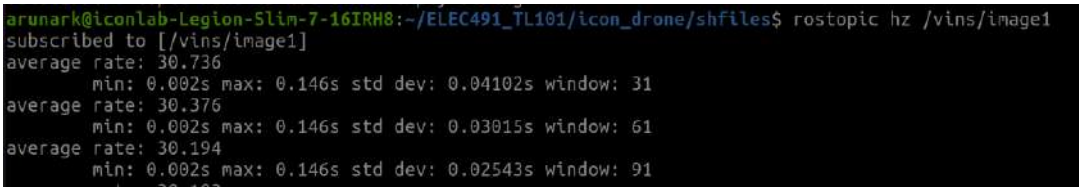
- Connect the Raspberry Pi and the host computer via Wi-Fi
- Stream video footage using HTTP/FTP protocols
- Monitor packet loss, throughput, and latency

Expected Result: Data transfers complete successfully with negligible packet loss and acceptable latency

Outcome

Data transfers were completed at the expected frequency and latency required by each node. The quality of the wireless network was controlled. To achieve optimal results, a 5-GHz router was used to simulate an isolated LAN.





As observed, the IMU data is received by the local PC at a frequency of 200 Hz, while images are received at 30 fps, both meeting the required and ideal specifications.

Test ID: AUTO-VAL-NET-02

Remote ROS Node Communication Test (Single Node)

Confirm the ability of Remote ROS to support topic-based communication over the wireless link in a simplified test configuration

- Configure Raspberry Pi as ROS master
- Launch a test publisher node on the remote host and subscribe from the Pi (or vice versa)
- Publish diagnostic messages (e.g., std_msgs/String)
- Monitor timing and topic availability through rostopic echo

Expected Result: Messages sent from the remote node are received and echoed correctly by the ROS master over Wi-Fi

Outcome

Messages were received without loss in real-time.

Test ID: AUTO-VAL-NET-03

End-to-End Remote Navigation Pipeline Validation

Validate complete bidirectional data flow during autonomous navigation using Remote ROS over Wi-Fi

- Start the ROS master on the Pi
- Publish stereo camera and IMU data to the remote PC
- Run VINS-Fusion and FUEL on PC
- Send planned path and velocity commands back to Pi
- Confirm navigation behavior visually using RViz and topic inspection

Expected Result: End-to-end loop completes with no loss of data or missed control updates

Outcome

All connections were secure, performing real-time operations, while all topics were published at expected times. RViz simulation verified the correct mapping of the environment.

3.2 Codebase Configuration & Software Build Tests

Test ID: AUTO-VER-SW-01

Client Codebase Build and Execution Verification

Confirm that the client-provided navigation code compiles and executes on the team's modified environment

- Clone the provided repositories
- Build the project with ROS and catkin
- Confirm build success and node launch

Expected Result: All packages build without error, and nodes run as expected

Outcome

All packages were built without error, and nodes exhibited expected behavior.

Test ID: AUTO-VER-SW-02

ARM Compatibility of RealSense and Lib Packages

Verify that Intel RealSense SDK and VINS-Fusion are rebuilt correctly for the Raspberry Pi's ARM architecture

- Manually rebuild librealsense on Pi
- Confirm build output
- Launch the camera and confirm the data stream

Expected Result: No segmentation faults; camera streams appear under rqt_image_view or rostopic echo

Outcome

Using the automated build script, the installation of all dependencies was ensured, and SDKs were installed correctly.

Test ID: AUTO-VER-SW-03

Automated Script-Based Deployment Verification

Confirm that custom setup scripts correctly install dependencies and launch the ROS environment

- Run the setup script on a fresh Pi and PC
- Observe the installation, build, and launch sequence
- Validate the absence of dependency or launch errors

Expected Result: All ROS workspaces are launched automatically and via one click

Outcome

The server and client scripts work together to automatically launch all required software, without waiting for further user action. The server script was set to launch automatically at the system boot up of the Raspberry Pi-4, and SSH was also enabled on the Pi for easy access to configuration files. Premature startup of the client script–that is, if the client is launched before the server is completely booted–fails to calibrate the IMU, which will fail any autonomous control that requires the IMU. In such cases, the output terminal indicates the calibration failure, and the user must restart the client initializer script before continuing.

3.3 ROS Nodes & Data Flow Tests

Test ID: AUTO-VAL-ROS-01

SLAM Pose Estimation Visualization

Validate that the SLAM module (e.g., VINS-Fusion, ORB-SLAM3) produces a stable and visually interpretable pose trajectory in a virtual environment

- Launch the full autonomy stack with a stereo camera feed and IMU data
- Use RViz to visualize the pose topic output from the SLAM node
- Observe the trajectory path and alignment with the expected motion within the map

Expected Result: Real-time trajectory in RViz is smooth, coherent, and consistent with simulated motion

Outcome

The SLAM algorithm reviewed in RViz displayed the correct mappings of the environment and accurately tracked the drone's real movements.

Test ID: AUTO-VAL-ROS-02

FUEL Trajectory Planning

Validate that the FUEL algorithm actively updates the next trajectories, based on the data that the drone collects during its flight

- Launch the full autonomy stack with a stereo camera feed and IMU data
- Use RViz to visualize the drone mapping
- Monitor the messages outputted by the FUEL Node and observe that its behavior matches the real flight

Expected Result: Real-time trajectory in RViz is smooth, coherent, and consistent with simulated motion. The decision logic to find the next path should be in line with real-time behavior

Outcome

The trajectory reviewed in RViz displayed the correct trajectory history of the drone. Results in RViz verified that the software is able to distinguish free space from obstacles and is able to plan the next path from the visual data that the drone receives. Appropriate flags were raised if the drone underwent unexpected control behavior (rapid-yaw, high velocity, etc.). It was also verified that in cases where all possible paths were blocked by obstacles, the drone fails to plan its next trajectory, hovers, and warns the user.

Test ID: AUTO-VER-ROS-01

Depth Sensing

Verify that the depth measured through the sensor is accurate

- Collect samples of the depth data at varying lighting and motion conditions
- Compare the outputs with the manually measured depth

Expected Result: The depth sensing outputs accurate data with no more than 1.5 m error.

Outcome

The depth sensors displayed accurate measures of the depth within the specified parameters. The RGB-D sensing technique through Depth-Anything-V2 initially produced inconsistent results with high exposure of light, which was corrected by post-processing the collected images.

Test ID: AUTO-VER-ROS-02

IMU Verification

Verify that the IMU provides reliable and consistent data

- Collect samples of the IMU data at varying motions

Expected Result: The IMU outputs realistic data that reflects the movements of the IMU, consistently over a duration of 5 minutes (flight time). The IMU data should not drift while stationary, and should not exhibit drastic jumps between data

Outcome

The IMU data recorded during flight outputs realistic data that reflects the movements of the drone. There is no visible spike in data or any noise interference.

Test ID: AUTO-VER-ROS-03

ROS Node Graph Connectivity Verification

Verify that all navigation-related ROS nodes are properly instantiated and interconnected within the ROS computation graph

- Launch the navigation stack in simulation
- Open the ROS graph visualization tool (rqt_graph)
- Check that all expected nodes appear and that publishers/subscribers are connected
- Confirm that topic names match across all interfaces

Expected Result: All nodes appear in the ROS graph with correct topic connections and no broken links

Outcome:

The ROS graph displayed all the nodes and topics, and also confirmed that all nodes were logically connected to each other.

Test ID: AUTO-VER-ROS-04

ROS Node I/O Behavior Verification
Confirm that each ROS node correctly responds to simulated sensor input with logically consistent and correctly formatted output messages

- Feed arbitrary but realistic sensor messages to each navigation-related node (e.g., stereo images, IMU, or pose data)
- Use rostopic echo or rqt tools to observe outputs
- Check formatting, data range, and temporal consistency

Expected Result: Each node publishes correctly structured messages to the appropriate output topic under stimulus

Outcome
Every node displayed the correct interconnections (publisher-subscriber). Expected data were exchanged at the required rates, without disrupting the overall ROS pipeline. Infrequently, validation of the exchanged data was rejected by the pipeline, but the high frequency of the communication made this occasional failure negligible.

Test ID: AUTO-VER-ROS-05

Data Flow Timing and Synchronization Test
Confirm timestamp and frame alignment across IMU, image, and pose messages

- Stream camera and IMU data from Pi
- Check synchronization using rqt_multiplot or custom visualization
- Validate TF tree using tf_view or RViz

Expected Result: Time deltas do not cause lag or latency issues, no TF lookup errors

Outcome
All data was received at the client computer, synchronized without issues. The relatively slower arrival of images was resolved by using a high-frequency IMU (for more thorough alignment with the frames) and compressed-format images.

3.4 Autonomous Flight Testing

Test ID: AUTO-VERVAL-FLIGHT-01

FSM Initialization and Arming Gate Test
Verify that PX4 FSM begins in SYSTEM_START and transitions only after arming
Validate that commands are correctly blocked until arming is complete

- Power on the drone, observe the FSM state (via console or /fsm_state)
- Attempt auto commands while disarmed → should be rejected
- Arm drone → FSM should transition to MANUAL_CTRL or AUTO_TAKEOFF

Expected Result: FSM behaves as expected; system rejects premature commands

Outcome
The FSM behaves as expected, transitions to the correct states upon appropriate signals received, and rejects premature commands

Test ID: AUTO-VERVAL-FLIGHT-02

Transition to MANUAL_CTRL After Arming and Takeoff

Verify that the FSM enters MANUAL_CTRL after arming and manual throttle input
Validate that the drone is controllable via manual RC input

- Arm drone
- Increase the throttle manually
- Confirm that FSM enters MANUAL_CTRL and the drone lifts off in response to stick inputs

Expected Result: FSM state changes correctly, and the drone responds to the pilot
Outcome
FSM behaves as expected, transition from MANUAL_CTRL occurs after arming and takeoff, the system behaves as expected, and the drone responds appropriately to user inputs from the RC controller

Test ID: AUTO-VERVAL-FLIGHT-03

Manual to AUTO_HOVER Transition Test
Verify the FSM transition from MANUAL_CTRL to AUTO_HOVER on input
Validate that the drone stabilizes in hover autonomously

- Fly manually, trigger hover via switch or command
- Confirm the FSM transition
- Observe stability in hover (no significant drift)

Expected Result: FSM state updates, drone autonomously hovers in place
Outcome
The FSM functions as expected, with state transitions occurring when RC input is used to switch the drone into AUTO_HOVER mode. However, the drone's auto-hover performance remains inconsistent. Through testing and the process of elimination, the issue was traced to PID tuning. While adjustments were made to the PID values to improve stability, due to a lack of time, the final PID values were not achieved.

Test ID: AUTO-VERVAL-FLIGHT-04

AUTO_HOVER to CMD_CTRL Transition Test
Verify FSM transitions only from AUTO_HOVER to CMD_CTRL (not directly from manual)
Validate that the drone accepts velocity or trajectory commands

- Enter AUTO_HOVER, then publish the control command
- Confirm FSM enters CMD_CTRL
- Attempt command from manual mode → should be blocked

Expected Result: FSM follows expected logic; drone responds to command only from the correct state
Outcome
The state transition happens correctly from AUTO_HOVER to CMD_CTRL. User commands were rejected other than switch control signals, and the drone received commands from the path searching algorithm. The drone successfully navigated within the room, demonstrating correct operation of the Software Navigation tools (VINS and FUEL). However, due to untuned PID settings, it was unable to correct errors and follow the planned path quickly and stably (i.e., Proportional control), resulting in gradual drift and loss of control.

3.5 Video Footage Capture

Test ID: SYS-VER-CAM-01

Script-Based MJPEG Video Capture

Verify that the monocam correctly streams and records MJPEG video to disk using the provided Bash script.

- Run the script:./record_video.sh
- When prompted, enter:
    - Resolution: 1280x720
    - Frame rate: 30 or higher
    - Press r to begin recording.
- Move the camera or place it in a dynamic scene for 5–10 seconds.
- Press q to stop recording.
- Confirm that an .mkv file was saved in ~/video with a timestamped filename.
- Play the file with a video player (e.g. VLC or mpv) and check for motion fidelity, resolution accuracy, and smooth playback.

Expected Result: The script correctly initializes the webcam, captures MJPEG video at the specified resolution and frame rate, and saves it as an .mkv file in the ~/video directory. The video should have no noticeable lag or motion blur under normal movement speeds.

Outcome History

Initial testing was performed on Ubuntu with a USB webcam. The script behaved as expected: the video recorded smoothly at 1280×720 resolution and 30 fps. The ability to adjust resolution and frame rate dynamically worked without issue. Video files were saved successfully in ~/video without overwriting previous files. The MJPEG input format avoided motion distortion typically caused by frame delay, and moving the camera at drone-like speeds resulted in sharp, stable recordings. The script is confirmed to support reliable streaming video capture over V4L2 on both desktop and Raspberry Pi platforms.

Test ID: SYS-VER-CAM-02

Raspberry Pi Frame Capture via MJPEG Stream

Verify that the Raspberry Pi running Ubuntu 20.04 can reliably capture and save raw MJPEG frames from the monocam to a timestamped folder, without overwriting prior data or overloading the system

- Run the script:./save_frames.sh
    - Quality and resolution automatically set
- Move the camera or place it in a dynamic scene for 5–10 seconds
- Press q to stop recording
- Inspect the created frame folder:
    - Ensure files are named frame_0001.jpg, frame_0002.jpg, etc.
- Confirm image quality is clear and sharp with no distortion
- Ensure previously saved folders in ~/frames/ are not deleted or overwritten.

Expected Result: All frames are saved without corruption, lag, or overwrite of prior sessions with sufficient picture quality.

Outcome History

This setup was tested on a Raspberry Pi running Ubuntu 20.04. The original method of recording video was removed because compression on the Pi was poor. The videos were choppy and looked low quality. If the Pi lost power while recording, the file would be corrupted and could not be recovered.

Instead, the new method saved individual MJPEG frames. Each frame was written directly to disk. This made the system more reliable during unexpected shutdowns. When only the monochrome UVC camera was connected and set to /dev/video0, the frames looked clear and the system was stable.

Problems started when the Intel RealSense was also connected. It sometimes took over /dev/video0, and the monocam became /dev/video6. Changing the script to use /dev/video6 worked at first. But running it alongside the rest of the autonomy software made the Pi slow down and eventually freeze.

The root issue was the camera's framerate. Forcing 30 fps overloaded the system and caused crashes. After updating the script to use a lesser framerate ~10fps the system ran smoothly. Frames were saved correctly and the image quality was good. As long as the Pi was not under heavy load, the script worked. However, with the RealSense working in parallel the script frequently crashed and another alternative needs to be explored.

Test ID: SYS-VER-CAM-03

Python Script for Video Recording with Full Autonomy Stack

Verify that the RGB camera records flight footage at the required specification of 720p resolution and 30 frames per second, suitable for 3D reconstruction.

- Activate the full server/client scripts, including the camera stack
- Run the updated Python-based recording script instead of the shell script.
- Record video for a few seconds during simulated or manual drone movement.
- Check the saved output file.
- Confirm the resolution and framerate of the footage
- Visually inspect the footage for consistency, clarity, and motion smoothness.

Expected Result: The RGB camera should successfully record footage at 1280x720 resolution and 30 fps. The video should be smooth and clearly show drone movement without motion distortion or dropped frames.

Outcome

Switching to a Python script allowed both the monochrome camera and the Intel RealSense to run together without crashing the system. The downsized RGB camera was software-accelerated and could only produce around 18–19 fps at 720p. The camera was set to fixed focus. Despite the lower framerate, the output was stable and visually clear.

3.6 Summary of Autonomous Mode Testing

The autonomous navigation system underwent comprehensive testing to validate its functionality and identify areas for improvement. The mapping capabilities were thoroughly assessed by manually carrying the drone and reviewing mapping outputs in RViz, as well as by piloting the

drone manually using a remote controller (RC) and analyzing the mapping results. Simulated environments, including RViz and Gazebo, were also utilized to verify the correctness of the software in controlled test scenarios.

Autonomous flight testing was conducted in part; however, the drone's inability to maintain balance and control over extended periods, caused by insufficient PID tuning, significantly limited the scope of these tests. During auto-hover trials, the drone remained stable for approximately 1-2 minutes before losing balance. In CMD_CTRL mode, it successfully avoided walls, but due to an untuned PID, specifically an inadequate proportional (*P*) value, it was unable to follow the VINS and FUEL-generated flight plan efficiently or correct errors quickly. This lack of timely correction ultimately led to a loss of stability and resulted in a crash, preventing the drone from passing the auto-hover state tests and hindering reliable data collection for advanced functionalities such as CMD_CTRL (the autonomous navigation state).

Despite these challenges, other software components, including initialization, scripting, ROS connectivity, and wireless communication, performed successfully in their respective tests. The results emphasize the need for further tuning and testing to achieve full autonomous navigation capabilities.

4.0 Simulation Testing

The simulation testing closely followed the staged progression outlined in the XTDrone tutorial. Each module was verified step by step, as later stages were dependent on the successful setup of earlier components. In several cases, critical build errors or compatibility issues required a full system reset. These instances involved completely wiping the workspace, reinstalling core packages, and redoing all prior tests to re-establish a working baseline.

The FUEL planner was integrated late in the process and was not thoroughly stress tested. While its functionality is briefly discussed in the design document, time constraints and repeated performance issues limited its in-depth evaluation.

4.1 Manual Flight Control

Test ID: SIM-VER-MANU-01

Keyboard Arming and Flight Control

Ensure that the simulated drone responds to arm and flight commands using XTDrone's keyboard control interface

- Launch roslaunch px4 mavros_posix_sitl.launch to start the simulation
- Start the keyboard interface scripts: multirotor_communication.py and multirotor_keyboard_control.py
- Press t to arm the drone and v to initiate takeoff
- Use keyboard inputs to verify manual maneuverability

Expected Result: The drone arms, takes off, and responds to keyboard commands with appropriate motor outputs and movement

Outcome History

Initial attempts to control the drone with the keyboard failed. The drone did not respond to any inputs. Diagnosis revealed that the multirotor_communication.py bridge script was not running. This script is required to forward keyboard commands to MAVROS

After launching the bridge node, the drone responded to the arm command but did not lift off. The issue was an incorrect PX4 parameter. The value of COM_ARM_SWISBTN was preventing full arming. The parameter was updated using QGroundControl.

Following the fix, the drone was able to arm and respond correctly to keyboard commands. It successfully lifted off and accepted directional input. ROS topic monitoring confirmed that state transitions were correct. This verified that communication between keyboard input, ROS, and PX4 was functioning properly.

4.2 VINS-Fusion Visual Inertial Odometry

Test ID: SIM-VER-VINS-01

VINS Node Initialization and Odometry Publishing

Confirm VINS-Fusion initializes properly and publishes visual-inertial odometry

- Launch VINS-Fusion nodes with indoor1.launch

- Confirm /vins_estimator/pose and /vins_estimator/path topics are publishing
- Observe TF and odometry tree in RViz

Expected Result: Continuous pose messages and a coherent TF tree from VINS

Outcome History

Initially, visualizing odometry in RViz was unsuccessful. No trajectory appeared, even though VINS-Fusion nodes were running. After troubleshooting, it was found that the original installation of VINS-Fusion was corrupted. To resolve this, the entire ROS environment, PX4 firmware, and packages were reinstalled from scratch.

After rebuilding the environment, the odometry output still failed to display. It was determined that the issue stemmed from a mismatch in OpenCV versions. The client's codebase was built around OpenCV 3.4, whereas the XTDrone tutorial and VINS-Fusion algorithm expected OpenCV 4.2. The code was recompiled using OpenCV 4.2.

Following this, the /vins_estimator/pose and /vins_estimator/path topics began publishing at approximately 20Hz. RViz successfully displayed the pose trajectory and path, confirming that the VINS-Fusion node was operating correctly.

Test ID: SIM-VAL-VINS-02

Trajectory Coherence with Environment

Validate that VINS-generated pose corresponds logically to drone's real-time location in simulation

- Launch indoor1.launch and fly manually for 60–90 seconds
- Overlay /vins_estimator/path in RViz

Expected Result: Path follows movement accurately without major drift or discontinuity

Outcome History

The visualized path closely matched the expected movement of the drone. Translations, rotations, and overall path curvature aligned with the drone's actual motion and the room layout. During hover, positional drift was minimal demonstrating stable odometry performance. Sharp turns, altitude changes, and cornering behavior were all represented in the path output without discontinuities. The result confirmed that VINS-Fusion maintained pose coherence throughout the test flight and accurately reflected the drone's navigation within the simulated space.

4.3 RTAB-Map Depth Map

Test ID: SIM-VER-DEPTH-01

Point Cloud Generation and Topic Verification

Verify depth camera integration and point cloud publication to RTAB-Map

- Replace iris_stereo with iris_realsense in launch files
- Launch RTAB-Map with rtabmap_vins.launch
- Monitor /rtabmap/cloud_map and /rtabmap/octomap_grid

Expected Result: Point cloud and occupancy grid should populate in RViz

Outcome History

First launch of rtabmap_vins.launch showed no point clouds in RViz. Although the /rtabmap/cloud_map and /rtabmap/octomap_grid topics were published, the visual output was empty. A

broken TF tree was then found. It was responsible for critical transforms from the world frame to the camera base were missing.

Static transform warnings confirmed the issue: RViz could not compute the transform from map or world to depth_camera_base. Static transform publishers were added to the launch file to link map → base_link and base_link → depth_camera_base.

After adding the transforms and relaunching RTAB-Map, the depth camera successfully generated a point cloud. RViz populated the map in real time, and the environment was accurately reconstructed with consistent updates from both cloud_map and octomap_grid.

Test ID: SIM-VAL-DEPTH-02

Room Geometry Accuracy in Point Cloud
Validate that depth map represents physical room structure

- Move drone through simulated indoor space
- Observe map buildup over time in RViz

Expected Result: Map structure should match visual room layout (walls, corners)
Outcome
The point cloud produced by RTAB-Map matched the visible room layout. Walls, open floor space, and major corners were outlined in RViz. After the TF tree was unified with proper static transforms, RTAB-Map remained stable during flight. Map updates appeared consistently as the drone moved. RViz displayed real-time updates for MapCloud, MapGraph, and Info, all synchronized correctly with the drone's motion. There is a slight concern about lag and overheating of the laptop however with this visualization.

4.4 EGO-Planner

Test ID:SIM-VER-EGO-01

EGO-Planner Input Topic Verification
Verify EGO receives necessary pose input and publishes trajectories

- Launch ego_planner_node
- Monitor /planning/trajectory output and input topics from VINS and MAVROS

Expected Result: EGO subscribes and publishes trajectories with correct orientation and frame
Outcome History
Build attempts for EGO-Planner failed due to missing dependencies and linker errors. Although pose_utils compiled successfully, downstream packages such as multi_map_server and odom_visualization could not find the shared library during linking. This caused the build process to halt with errors such as /usr/bin/ld: cannot find -lpose_utils.

To resolve this, pose_utils was restructured as a proper catkin library. Its CMakeLists.txt file was updated to declare and export headers and libraries using catkin_package. The packages that depended on it were updated to include pose_utils in find_package, catkin_package, and target_link_libraries.

All affected packages were cleaned and rebuilt using isolated catkin build commands in dependency order. After these fixes, the entire workspace compiled cleanly, and no packages were marked as "abandoned."

Once built, the ego_planner_node was launched successfully. ROS confirmed that it was subscribing to /mavros/local_position/pose and /vins_estimator/pose, and began publishing smooth curved trajectories to /planning/trajectory at around 10Hz. RViz confirmed that the output trajectories were visualized in the correct reference frame and orientation, completing the verification.

Test ID: SIM-VAL-EGO-02

Autonomous Room Traversal with EGO-Planner

Validate that EGO-Planner can generate local trajectories that allow the drone to traverse a simple indoor environment under offboard control.

- Launch indoor Gazebo simulation with default XTDrone room layout
- Arm drone and manually ascend to 1 meter altitude
- Switch drone to offboard mode
- Launch EGO-Planner node and observe trajectory takeover
- Monitor navigation performance via RViz and visual behavior

Expected Result: Drone responds to EGO-generated commands and navigates through the simulated room with smooth trajectories, maintaining localization and flight stability

Outcome History

EGO-Planner was tested across multiple trials. After a successful build, the planner was able to take control once the drone was armed and hovering at one meter. In early tests, it generated smooth, short trajectories. The drone was able to navigate through the small room where it was initialized. Visualization in RViz showed accurate path data at this stage.

As the drone encountered more complex parts of the environment, performance declined. It failed to recognize irregularly shaped objects. It also missed doorways and narrow passages. When this happened, the planner would stall or drift into corners. It stopped re-planning effectively. Despite this, manual control could always be restored. The drone could be safely recovered using manual input or the auto-land command.

Performance was highly sensitive to conditions. If the drone hovered above one meter, closer to or above the simulated wall height, EGO-Planner would break down. The drone began to move unpredictably, swerving in random directions. This suggested the planner was not robust to elevated viewpoints.

System performance also suffered. RViz became laggy when depth maps were rendered. Over time, the computer ran hot and resource usage spiked. This pointed to synchronization issues or hardware limits during visualization.

5.0 Project Requirement Testing

Please see the detailed list of requirements in the *Requirements Document* for requirement descriptions and testing procedures. The following table describes which tests are applicable to verify the various project requirements.

Requirements	Test ID For Verification
F_1 - Manual Controls and Manual Takeover	MANU-VER-ELEC-03, MANU-VER-SW-01, MANU-VER-SW-02, SIM-VER-MANU-01
F_2 - Communication Latency and Bandwidth	AUTO-VAL-NET-01, AUTO-VAL-NET-02, AUTO-VAL-NET-03, AUTO-VER-ROS-05
F_3 - Battery Safety	MANU-VER-ELEC-01, MANU-VER-ELEC-02, MANU-VER-ELEC-05, MANU-VER-SW-03
F_4 - Image Sensor Requirements	AUTO-VER-ROS-01, MISC-VERVAL-CAM-01, SYS-VER-VID-01
F_5 - Depth Data Acquisition	AUTO-VER-ROS-01, AUTO-VER-ROS-02, SIM-VER-DEPTH-01, SIM-VAL-DEPTH-02
F_6 - Manual/Autonomous Switching Procedure	MANU-VER-ELEC-04, AUTO-VERVAL-FLIGHT-01, AUTO-VERVAL-FLIGHT-02, AUTO-VERVAL-FLIGHT-03, AUTO-VERVAL-FLIGHT-04
F_7 - Path Planning & Perimeter Mapping	AUTO-VAL-ROS-01, AUTO-VAL-ROS-02, SIM-VAL-VINS-02, AUTO-VER-ROS-03, AUTO-VER-ROS-04, SIM-VER-EGO-01, SIM-VER-EGO-02
NF_1 - Operating System Compatibility	AUTO-VER-SW-01, AUTO-VER-SW-02, AUTO-VER-SW-03, SIM-VER-VINS-01
C_1 - Frame Size	MANU-VER-MECH-01, MANU-VER-MECH-02, MANU-VER-MECH-03
C_2 - Room Size Coverage	MANU-VER-ELEC-05

Appendix A. Contributions

Document Section	Major Content	Minor Content	Author	Review
[number & title]	[Initials]	[Initials]	[Initials]	[Initials]
1.0 Overview	NK		NK	RBC
2.0 Manual Mode Testing				NK
➢ *2.1 Mechanical Tests*	RBC, ST		RBC, ST	NK
➢ *2.2 Electrical Tests*	ST, RBC		ST, RBC	NK, RJ
➢ *2.3 Software Configuration Tests*	ST, RBC		ST, RBC	NK, RJ
➢ *2.4 Summary of Manual Mode Testing*	ST		ST	NK
2.0 Autonomous Mode Testing				NK
➢ *3.1 Wireless Communication Tests*	RJ, AS		AS	
➢ *3.2 Codebase Configuration & Software Build Tests*	RJ		RJ	
➢ *3.3 ROS Nodes & Data Flow Tests*	AS, RJ		AS, RJ	RBC
➢ *3.4 Autonomous Flight Testing*	AS, RJ	NK, RBC, ST	NK	RBC
➢ *3.5 Video Footage Capture*	NK, RJ	AS	NK	RBC
➢ *3.6 Summary of Autonomous Mode Testing*	AS, RJ		RJ	NK
4.0 Simulation Testing				
➢ *4.1 Manual Flight Control*	NK	RBC	NK	RBC
➢ *4.2 VINS-Fusion Visual Inertial Odometry*	NK	AS	NK	RJ
➢ *4.3 RTAB-Map Depth Map*	NK		NK	RJ
➢ *4.4 EGO-Planner*	NK		NK	RBC
5.0 Project Requirement Testing				
4.0 Project Requirement Testing	ALL	ALL	ALL	

5 Glossary

- **4-in-1 ESC**: An electronic speed controller unit that consolidates four motor drivers into a single PCB to reduce wiring complexity.
- **6-DOF**: Six degrees of freedom; describes motion in 3D space along x, y, z axes and rotation in pitch, roll, and yaw.
- **ADC (Analog-to-Digital Converter)**: Converts continuous voltage signals into digital values for microcontroller processing.
- **AHRS Driver:** Software that reads data from IMUs and provides orientation estimates (attitude heading reference system).
- **Airframe:** The structural framework of the drone that holds motors, payload, and electronics.
- **Altitude Hold:** A flight controller function that maintains the drone at a consistent height automatically.
- **Application Specific Integrated Circuit (ASIC):** A custom-built chip in the RealSense camera that performs real-time depth calculations.
- **Arming:** The process of enabling motor control; disarming is used to prevent accidental spinning.
- **Autonomous Flight:** Unmanned drone operation using algorithms without manual control input.
- **Battery Management System (BMS):** Monitors battery health and voltage; prevents deep discharge or overuse.
- **Betaflight:** A firmware platform often pre-installed on FPV drones; needs to be erased for PX4 usage.
- **Buck Converter:** An electrical component that steps down higher voltage to a lower regulated output.
- **DFU Mode:** Device Firmware Upgrade mode allows low-level firmware flashing on STM32 boards.
- **Depth Estimation:** Determining distance to objects from images, crucial for navigation and SLAM.
- **Depth Image:** A 2D array where each pixel contains a distance value from a depth camera.
- **Disparity Map:** Used in stereo vision to represent pixel differences, which are used to compute depth.
- **Dual-arm Bracket**: A mount used to vertically secure the Raspberry Pi or other boards inside the drone frame.
- **EMAX Formula Series:** A high-performance ESC model rated at 45A, used in the drone.
- **ESC (Electronic Speed Controller):** Takes PWM inputs and drives brushless motors using 3-phase AC signals.
- **FUEL:** A path planning algorithm that uses 3D maps for safe, efficient exploration.
- **Failsafe:** A software routine that triggers safe landing or hovering on signal or battery loss.
- **Field of View (FOV):** The angular range a camera can capture, affecting mapping quality.
- **Flight Controller:** Central processor that manages drone sensors, stabilization, and motor commands.
- **Flight Stack:** All software and firmware modules that run on the drone's flight controller.
- **Flyaway:** A failure mode where the drone loses control and escapes intended airspace.
- **Frontier Detection:** The SLAM-based identification of unexplored areas in the environment.

- **GPS Lock:** Sufficient satellite connection to provide reliable positional data.
- **Global Shutter:** A camera feature where all pixels are exposed simultaneously to avoid motion blur.
- **Ground Control Station (GCS):** Software like QGroundControl used to monitor and configure the drone.
- **Gyroscope:** Sensor in IMU measuring angular velocity along pitch, roll, and yaw axes.
- **Hover Throttle:** The throttle percentage at which the drone maintains level flight without climbing.
- **I2C:** A digital communication protocol used between microcontrollers and peripherals like the BMS.
- **IMU (Inertial Measurement Unit):** A device measuring linear acceleration and angular velocity for motion tracking.
- **Intel NUC:** A compact PC platform used for onboard SLAM and path planning computations.
- **KV Rating:** Indicates a motor's RPM per volt without load. Higher KV = faster but less torque.
- **Latency:** The delay between command issuance and observed effect; important in real-time systems.
- **LiPo (Lithium-Polymer) Battery**: High energy density power source used in drones, typically with 3–5 cells.
- **MAVROS:** A ROS node that connects PX4 with the ROS ecosystem over MAVLink.
- **Monocular Camera:** A single-lens camera used for vision; lacks depth unless processed.
- **Odometry:** Estimation of a robot's movement over time using sensor data.
- **Offboard Mode:** A control mode where the drone is commanded by an external computer via ROS.
- **PX4:** An open-source flight control firmware used for autonomous or manual drone control.
- **Payload:** Any component or sensor added to the drone that is not part of basic flight.
- **Pixhawk 6C**: An advanced flight controller hardware that runs PX4 or ArduPilot firmware.
- **Pose:** Position and orientation of the drone in 3D space, usually 6-DOF.
- **Power Distribution Board (PDB):** Distributes voltage from the battery to all other drone components.
- **QGroundControl:** GUI software to configure, calibrate, and monitor drones running PX4.
- **Quadcopter:** A four-motor drone configuration commonly used for aerial robotics.
- **R9DS:** A Radiolink receiver model that supports SBUS communication with flight controllers.
- **ROS (Robot Operating System):** A middleware framework used to organize and interconnect drone software modules.
- **ROS Node:** A single process in a ROS system that performs a specific function (e.g., navigation).
- **RPi (Raspberry Pi 4):** A compact Linux computer used onboard for computation and sensor interfacing.
- **Radiolink AT9S Pro:** A 12-channel RC transmitter used for manual drone control.
- **RealSense D435i:** Intel stereo depth camera with integrated IMU and depth-processing ASIC.
- **Receiver:** The drone-side unit that receives control signals from the transmitter.
- **SBUS:** A digital RC communication protocol allowing many channels over a single wire.
- **SLAM (Simultaneous Localization and Mapping):** An algorithmic framework that builds a map while estimating the robot's position.
- **STM32:** A microcontroller family from STMicroelectronics, commonly used in flight controllers and BMS boards.

- **Setpoint:** A desired value (e.g., position, velocity) sent to the flight controller for execution.
- **Snubber:** A circuit used to dampen high-frequency switching noise in power electronics.
- **Stereo Camera:** A dual-lens camera setup used to infer depth through image disparity.
- **Telemetry:** Real-time data transmitted from the drone to a ground station or onboard logger.
- **Throttle:** Control signal that adjusts the speed of the motors.
- **Trajectory Planning:** The generation of a path for the drone to follow safely and efficiently.
- **UART:** Universal Asynchronous Receiver-Transmitter; a protocol used for serial communication.
- **Ubuntu 20.04:** The Linux distribution used for both onboard and offboard systems.
- **VINS-Fusion:** A visual-inertial SLAM algorithm combining camera and IMU for pose estimation.
- **Vision Pose Topic:** A ROS topic providing position and orientation from vision sensors.
- **Voltage Regulator:** An electronic component that ensures consistent voltage to sensitive components.
- **Waypoint:** A target position in 3D space the drone is commanded to reach.
- **Wi-Fi 5 (802.11ac):** The wireless standard used for communication between drone and offboard host
- **Z-Map**: A type of depth map where each pixel value represents distance in mm.
- **PID**: Proportional-Integral-Derivative

6 References

[1] raeditio, "ELEC491_TL101," *GitHub Repository*, 2024. [Online]. Available: https://github.com/raeditio/ELEC491_TL101. [Accessed: 30-Mar-2025].

[2] X.T. Drone Project, “dense_reconstruction,” XTDrone Manual (in English), accessed: Aug. 5, 2025. Available: https://www.yuque.com/xtdrone/manual_en/dense_reconstruction; “XTDrone,” XTDrone GitHub Repository – README.en.md, accessed: Aug. 5, 2025. Available: https://github.com/robin-shaun/XTDrone/blob/master/README.en.md

[3] Z. Tianda, px4_gazebo_fuel, GitHub repository, accessed: Aug. 5, 2025. Available: https://github.com/zhongtianda/px4_gazebo_fuel