Final Design Document

LION: Autonomous Drone

Group 10a

Team Members:

Rayyan Jamil

Abraham Ng

Nathaniel D'Alfonso

Connor Hallman

Sponsors:

Madalyn Braganca

Christopher Griffith

Serco

IDesign Coordinators:

Matthew Gerber

Richard Leinecker

COP4934, Summer 2025

7/28/2025

Table of Contents

1. Executive Summary

The LION Autonomous Drone project represents an advanced robotics initiative designed to fundamentally bridge the gap between high-level human intent and complex robotic action. The primary goal is to develop an intelligent aerial system that can understand and execute complex commands given in natural language. This objective is powered by a multi-faceted AI architecture that includes a Visual Language Model (VLM) for real-time perception, a Large Language Model (LLM) for reasoning, and an Agentic AI Framework, like SmolAgents, to orchestrate these intelligent components.

At the heart of our system lies a GCS-centric AI architecture, a strategic design that offloads intensive computational tasks from the drone to a powerful Ground Control Station (GCS). This approach allows the drone to remain a lightweight, agile, and power-efficient sensor platform while the GCS leverages state-of-the-art AI models that would be impractical to run onboard. This strategic division of labor allows us to maximize AI performance by leveraging GCS computing power and dedicate our primary efforts to innovative software development.

The physical implementation consists of a Holybro X500 V2 quadcopter equipped with a ZED 2i stereo camera and a Raspberry Pi 5 companion computer. The Raspberry Pi 5's enhanced processing power is critical for handling the simultaneous, high-throughput streaming of RGB video and raw depth data to the GCS, which is powered by an NVIDIA Jetson compute platform. Within the GCS, the agentic AI processes the live sensor data, formulates an actionable plan, and translates it into precise flight instructions. These commands are then transmitted to the drone's PX4 flight controller via the robust MAVLink protocol, enabling fully autonomous execution.

The project's Minimum Viable Product will demonstrate this entire workflow in a defined indoor environment. Key objectives include real-time 3D point cloud generation, VLM-powered object identification and localization, and the successful execution of natural language commands such as "fly five feet up and orbit the red chair." Ultimately, the LION project aims to deliver a robust and extensible platform for applied research in human-robot interaction, showcasing a practical methodology for deploying complex AI on autonomous systems in real-world scenarios.

2. Introduction

2.1 Project Significance

An agentic drone system is a significant achievement and milestone in the world of autonomous robots. Although we do have limited autonomous systems such as Roombas, self-driving cars, warehouse robots, and other highly specialized systems that perform very niche activities, these systems lack true agency; they do not have the broader decision-making capability and goal-driven initiative that characterize true agentic systems. Agentic autonomy is more than merely executing preprogrammed tasks; it requires the ability to perceive dynamic environments, generate and evaluate plans, adapt strategies in real time, use the right tools for the right tasks, and pursue objectives without constant human oversight.

Our system will be a truly agentic drone system, with two clear consequences of significance. Firstly, by laying the groundwork for an agentic drone system, we will help pave the way for future drone systems to operate with even more complex behavior. We anticipate that subsequent innovations may include multiple drone systems, role-based behavior (surveillance, operator, reinforcements), and further autonomy with minimal to no oversight. Secondly, our framework will likely be extensible to other types of vehicles, such as land vehicles, aquatic surface vehicles, space rovers, planes, and more. These benefits follow from our commitment to using open-source technologies and making our codebases open-source.

Professionally, our agentic drone framework could potentially find application in a diverse array of other industries. Drones are becoming more commonplace for photography and videography at weddings, real estate, and other events. They are utilized for industrial inspections, location mapping, livestock mustering, crop monitoring, and medical deliveries. However, what these applications all have in common is that they require significant attention from a human operator, leaving room for improvement in efficiency, scalability, and safety. Instead of requiring someone to constantly guide every moment, an agentic drone can understand mission objectives, adapt to changing conditions, and make intelligent decisions in real time. Our hope is that our agentic framework will provide a foundation for these industries to revolutionize to the next level.

2.2 Project Motivation

In this section, each member will discuss their own individual motivations regarding our project.

2.2.1 Rayyan Jamil

For me, the LION Autonomous Drone System project represents a multidisciplinary challenge because it is positioned at the intersection of my interests in robotics, information technology, and software engineering. I am primarily motivated by the opportunity to address the complex technical roadblocks that come with creating intelligent cooperation between an autonomous aerial platform and its human operator. The core challenge of designing an AI system that can adapt, learn, and make robust decisions in dynamic, real-world scenarios is exactly the kind of significant engineering problem I find most engaging.

A key draw of this project is its engagement with deployable AI hardware. The integration of compact computing platforms, such as the NVIDIA Jetson for GCS processing and the Raspberry Pi 5 for onboard data handling, provides a link between abstract AI concepts and their practical application in an airborne system.

Furthermore, this project is distinguished by its strict reliance on live, real-world data. In contrast to many AI development workflows that utilize simulated or pre-recorded datasets, this project's commitment to operating exclusively on live sensor streams presents a more authentic and demanding challenge. The system must process and react to the unpredictable and often imperfect data that characterizes real-world operation, forcing a more robust design.

Finally, the project's focus on an AI-heavy implementation is a significant motivating factor. The emphasis on utilizing Large Language Models (LLMs), agentic workflows, and Visual Language Models (VLMs) offers an invaluable opportunity to implement these advanced concepts in a system that functions autonomously in the physical world. It is one thing to develop AI models in a controlled setting, but it is another to deploy them in dynamic environments where their decisions have immediate, tangible consequences. This project provides a unique chance to bridge the critical gap between cutting-edge AI research and its impactful, real-world deployment, yielding practical solutions to complex problems.

2.2.2 Abraham Ng

My motivation for selecting this project is fourfold. Foremost, I aim to deepen my current understanding of agentic AI, alongside the large language models and machine learning that underpin such agents. My prior background working with AI has been limited to calling an AI model via API from the frontend of a cat breed detector project and using AI chatbots. Through this project, I hope to improve my ability to interface with large language models (through prompting them and writing relevant APIs on both frontend and backend) and to better understand their technical foundations and applications (through machine learning and LangChain/LangGraph).

Second, I am interested in developing stronger skills with Python, as it has become ubiquitous in programming, particularly for AI and associated libraries and frameworks. My prior exposure to Python is highly limited, with most of my programming has been in C, Java, and Javascript thus far. Given Python's relevance and prevalence, I believe it is critical for me to gain more experience and familiarity with the language.

Third, I would like to continue practicing full-stack development via Next.js, as my most recent work prior to this project required me to fully develop both a Vite frontend for a full stack project and virtually the entire catalogue of required APIs to interface with the backend via Express.js. I anticipate that being able to work on the React dashboard for this project will both utilize my previous background with React on the frontend and reinforce my backend integration skills in writing API calls.

Lastly, I am hopeful to gain experience working directly with a defense contractor in an Agile environment. Learning the standard practices involving scrum and sprints is important for working with a team in the programming world, and exposure to these standard practices has been helpful in increasing my productivity as well as my understanding of the typical workflow and environment for software developers. Additionally, the local job market has a high concentration of defense contractors and related companies, and I anticipate that this project will be instrumental in helping me land my first job after graduating with my bachelor's degree in computer science.

2.2.3 Nathan D'Alfonso

My motivation for joining the LION Autonomous Drone project stems from a long-standing interest in AI, autonomous systems, and robotics. When I first learned about the project opportunity sponsored by Northrop Grumman (now Serco), I immediately saw it as a chance to integrate my academic interests with real-world engineering challenges. The project's focus on developing a drone system that interprets and executes human commands through AI aligns closely with my interests in multi-agent systems, machine learning, and embedded hardware design.

Unlike previous software-based academic projects I have worked on, this effort bridges both hardware and software, an area I have been eager to explore further. Designing a working drone, integrating sensors, and building a fully functional human-AI interface is exactly the type of system-level engineering that I hope to pursue professionally.

I believe that the human-in-the-loop interface should be intuitive and transparent, providing clear feedback about what the drone is doing and why. Building a reliable, explainable AI system is not just a technical goal but a human-centered one, especially in scenarios where autonomous behavior must be trusted.

Additionally, I appreciate the creative freedom that Serco has given us in shaping the system's design. This flexibility allows us to explore innovative approaches and iterate on ideas, rather than following a rigid client spec. Working closely with a multidisciplinary team of peers, each bringing different strengths to the table, makes this a uniquely collaborative and fulfilling experience.

Overall, this project is giving me the opportunity to apply and extend my skills in a highly practical setting. From integrating AI into real-time decision-making, to researching advanced perception techniques, to solving coordination challenges across hardware and software components, I am excited by the breadth and depth of what we are building. I see this experience as a critical step in my journey toward solving complex, interdisciplinary problems and contributing to the future of intelligent robotic systems.

2.2.4 Connor Hallman

My motivation for dedicating myself to the LION project stems from a long-held desire to work with physical systems. While I have focused primarily on web development throughout my computer career, the idea of writing code that translates into tangible, real-world action is something I have always wanted to do. For one, it's cool! Stuff is happening! But also, working with hardware presents a unique and exciting set of challenges that I have previously been intimidated by, and this project offers the perfect opportunity to overcome that.

I am also excited by the agentic portion of this project. Being able to architect a system that reasons and acts autonomously is a priceless opportunity with the current state of the world. AI is at the forefront of computer science and almost every aspect of life right now. I am excited to contribute to a field with profound implications for researchers and everyday people.

This project has challenges that will push me out of my comfort zone and let me grow faster than anything else I could do. I plan to use it as a vehicle for expanding my capabilities, forcing me to grapple with unfamiliar concepts in robotics, aerial systems, and agentic AI. It has also been on my to-do list to learn Python for quite some time now, but I have never had the opportunity to tackle it. More importantly, this experience will fundamentally expand my problem-solving toolkit, creating a new mental framework that will allow me to relate future technical concepts to the problems, solutions, and strategies I learn throughout this project.

Beyond the academic and technical benefits, I am excited to apply my existing skills. I believe a polished and intuitive user experience is critical to the project's success, and I am eager to leverage my web development experience to build a truly great React dashboard for the Ground Control Station. The skills developed here spanning autonomous systems, AI integration, and robotics are directly applicable to a wide range of cutting-edge industries, which opens doors to exciting job opportunities in drone systems, agentic AI development, and with defense contractors, aligning perfectly with my long-term professional goals.

Last but not least, I am thankful to Serco and our sponsors for the creative freedom they have given us, and to UCF and my group mates for the opportunity to work collaboratively on this project toward more possibilities, learning opportunities, and real-world applicability.

2.3 Broader Societal Impacts

The LION Autonomous Drone project has the potential to create positive change across several areas of society, from accessibility and safety to education and national preparedness. By designing a drone system that responds to natural language and supports human-in-the-loop decision-making, we are lowering the barrier for people of all technical backgrounds to use autonomous systems effectively. This is especially important for users with disabilities, who may not be able to operate traditional drone systems with manual controls. A voice-command or text-based interface opens new possibilities for accessible robotics and greater inclusion in the use of smart technologies.

Our system also helps reduce environmental impact by supporting non-invasive aerial scans. Traditional survey methods often require vehicles or people to physically move through an area, which can disturb fragile ecosystems. With our drone's lightweight sensors and autonomous navigation, the system can quickly scan large areas without causing disruption. This is useful not only for disaster response but also for environmental monitoring tasks such as tracking erosion, vegetation loss, or pollution.

In education and STEM, our project encourages learning by combining real-world hardware with cutting-edge software tools. The use of accessible technologies like React, Python, ROS, and open-source AI frameworks can help students and researchers explore similar systems in their own work. Our documentation and design process also serve as a learning resource for others hoping to develop intelligent, mission-based robotics.

Finally, this project supports national security efforts by enabling faster, safer, and more reliable situational awareness. During emergencies or unexpected events, autonomous drone systems like ours can provide live updates and map dangerous areas without risking human lives. By combining live video, thermal imaging, and real-time decision-making, our system could support first responders, public safety agencies, and defense contractors in gathering the information they need quickly and efficiently. While our current focus is on research and development, the broader goal is to contribute to safer, smarter, and more adaptable technologies that help protect both people and infrastructure.

2.4 Legal, Ethical, and Privacy Issues

2.4.1 FAA Regulations

Our primary legal concern is compliance with the regulations of the Federal Aviation Administration. All outdoor drone use falls under FAA jurisdiction, including private spaces such as the limited clearing outside the Serco campus building where we currently plan to operate our drone (originally, we had considered operating on the grounds of the University of Central Florida but decided that the myriad of administrative formalities and procedures would prohibitively set back our project timeline). Specifically, any unmanned aircraft that weighs more than 0.55 pounds needs to be registered accordingly, which our drone clearly will exceed. In addition, our drone operator(s) will need to pass the requisite FAA tests to be appropriately certified. To fulfill these legal obligations, we will ensure that all registrations and certifications necessary will be completed before conducting any of our live drone tests.

2.4.2 Agentic Autonomy and Human Oversight

A key ethical consideration in the deployment of our system is balancing machine decision-making with human judgment. Fully autonomous drones can potentially misinterpret commands, leading to unpredictable behavior in complex environments. Recognizing this risk, our system includes a structured human-in-the-loop design, where a human operator can approve the plan of action created by one of our AI agents before it is carried out. In addition, the human operator will be able to engage in a manual override at any point in time to take full control of the drone. Lastly, our current project scope will limit the environment to a narrow enclosure with limited obstacles and uniform terrain, avoiding overly complicated scenarios while we work on developing our agentic AI models. These measures will reduce the likelihood of unpredictable behavior and ensure that any critical decisions will remain under human oversight and discretion. Our project's approach thus mitigates concerns about the trustworthiness of autonomous behavior and demonstrates a commitment to responsible agentic AI usage.

2.4.3 Personally Identifiable Information

Nearly all drone-related projects run the risk of capturing personally identifiable information, including facial features, license plates, and any other personal items in the environment. Our plan regarding our testing site will reduce this risk by implementing a variety of measures. Foremost, we will be operating the drone in a small clearing adjacent to our sponsor's corporate building rather than a public field, reducing the likelihood of encountering unanticipated foot traffic. Furthermore, we will clearly mark the zone of our drone operations to deter other building occupants from entering the area. Finally, we will ensure all captured data will be used strictly for point cloud generation within the boundaries of our testing site, and we plan to ensure that all identifiable information captured by camera will be either removed or rendered unintelligible. Together, these precautions will strengthen the privacy standards of our project.

2.4.4 Data Security

Given the potentially sensitive nature of the spatial and visual data that our drone system will collect as well as the human and agentic AI interactions that our React dashboard will facilitate, data security is a critical area of concern. Without the proper safeguards, our project's data may be intercepted or exploited, whether through our local wireless communication protocols or through global communication to outside large language models. To address these issues, we are adopting a privacy-centric architecture for our project. First, we will be utilizing locally hosted large language models through Ollama, avoiding the use of online models that may collect user commands or contextual data during our drone missions. Second, all wireless communication between the drone and the ground control station will be encrypted using industry-standard protocols. Third, we will restrict access to our system to authenticated users via a secure login system, with all private and sensitive data strictly hidden behind User Based Access Control (UBAC) and Role Based Access Control (RCAC). These technical safeguards will ensure that all data from the drone, from the human operator, and from the large language models will remain secure and compliant with best practices regarding data security.

3. Project Characterization

3.1 Objectives and Goals

The main goal of the LION Autonomous Drone project is to build a system that can take natural language commands from a human and turn them into real-world drone actions. We are aiming to make drones easier to use by combining hardware, AI models, and a smart control center into one unified architecture. Instead of relying on manual controls, scripted instructions, or complicated programming interfaces, users will be able to communicate with the drone using everyday language, such as speaking or typing a sentence. From there, the system uses a combination of agentic AI frameworks, onboard sensors, and real-time data processing to interpret the request, break it into actionable steps, and execute a full mission cycle autonomously.

One of our main objectives is to demonstrate a complete mission flow from start to finish with minimal human intervention. This includes everything from parsing the natural language command, to autonomous flight planning, environmental sensing, data collection, and return or follow-up actions. For example, a user might issue a high-level instruction like "scan the field for damage and report anything unusual." In response, the drone would deploy, analyze thermal or RGB imagery, and highlight relevant findings back to the user. Achieving this flow validates that our architecture can coordinate between the language model, companion computer, flight controller, and user interface seamlessly. We also aim to make this system modular so that new sensors, input types, or AI capabilities – such as object detection or anomaly tracking – can be added later without requiring complete rewrites of the core system.

Another key objective is to design the interface to be simple, transparent, and intuitive. We believe usability is critical for widespread adoption of autonomous systems. By building a clear dashboard with real-time video, sensor telemetry, and status messages, we aim to demystify the internal workings of the drone and ensure that even non-technical users can understand what the system is doing and why. This transparency supports trust and safety, especially in sensitive or time-critical scenarios like emergency response. The interface is also built to support future scalability, with features like multi-drone status panels or AI reasoning summaries planned for later phases.

We are also focused on ensuring that the system is safe, flexible, and scalable for future use cases. This includes implementing fail-safe mechanisms, real-time monitoring tools, and well-defined handoff points between autonomous logic and human oversight. Our architecture is intentionally built with open-source tools and clear modular boundaries so that it can serve as a foundation for future research or practical deployments. In the long term, we envision LION being adapted for applications in search and rescue, disaster relief, infrastructure inspection, and environmental scanning. By publishing documentation and maintaining reproducibility, we hope to provide a platform that others in the academic or public safety communities can build upon, adapt, and evolve beyond the scope of this project.

3.2 Specifications and Requirements

To meet our goals, we have identified a number of technical and user-based requirements that our system needs to fulfill. These requirements are grouped into three main categories: functional capabilities, hardware/software performance, and usability for the end user. Together, they define the baseline of what our autonomous drone system must support to be considered successful and scalable beyond our minimum viable product. These requirements also serve as the foundation for future modular expansion and enable informed trade-offs between stretch goals and critical deliverables.

From a functional standpoint, the system must allow a human user to input a natural language command and receive a successful response from the drone system. The command must be interpreted by an agentic AI framework capable of parsing high-level goals, breaking them down into executable steps, and transmitting appropriate instructions to the drone in real time. This framework should support both autonomous planning and responsive task adaptation, allowing the system to react fluidly to unexpected sensor input or environmental changes. The agent must be able to maintain internal memory, manage task queues, and prioritize mission-critical objectives. In addition, the system must be able to collect sensor data – such as live video, depth information, or point clouds – and send it back to the Ground Control Station (GCS). The GCS must then process and display this data in a clean, intuitive format that gives the operator full situational awareness, empowering the user to evaluate progress and intervene when needed.

From a hardware perspective, the drone must support high-quality sensors such as the ZED 2i stereo camera, which provides both RGB and depth data and is suitable for indoor and outdoor mapping. A reliable flight controller is essential for stable flight and accurate movement execution, while wireless communication modules ensure a continuous link between the drone and the GCS, even in signal-challenged environments. The drone must be lightweight yet powerful enough to support necessary payloads and onboard computation when needed. Mechanical design must also allow for modularity; sensors, mounts, or compute modules should be easily replaceable or upgradable without requiring system redesign. Meanwhile, the GCS should be able to run a local instance of a Large Language Model (LLM), likely using the NVIDIA Jetson platform or similar embedded AI hardware. This allows the system to remain functional in remote or bandwidth-constrained environments without relying on cloud infrastructure. In addition, all communication between the drone and the GCS must be secure, low-latency, and capable of handling both high-bandwidth video streams and time-sensitive flight commands simultaneously. Redundancy and fail-safe mechanisms should also be considered in both hardware and network layers to improve mission robustness.

From a user experience standpoint, the interface must be designed for ease of use, especially in high-pressure environments such as search and rescue or disaster response. The dashboard must include four core components:

1. A live video feed from the drone to provide immediate visual awareness
2. A 3D digital twin of the scanned environment that updates in near real time for spatial context
3. A chat-style input where users can issue natural language commands and receive confirmations
4. A system log that displays real-time telemetry, status messages, and system alerts for debugging and situational clarity

These features ensure that the user is always aware of what the drone is doing and has the ability to make decisions, intervene, or switch to manual control if needed. The interface must be responsive, informative, and reliable under varying environmental conditions, whether operating in daylight, low-light, or outdoor field deployments. To reduce friction, the interface should also support guided input or command suggestions to help non-technical users interact with the drone effectively.

Beyond immediate usability, interoperability with external systems is another design goal we considered early in the requirement definition phase. Whether through exporting scan data in standard formats, integrating with public geospatial APIs, or accepting mission parameters from third-party planning tools, the system must remain open and extensible. This ensures that future applications, such as integration with emergency response platforms or environmental monitoring dashboards, can be realized without rebuilding the core system. Establishing standardized input/output formats and API endpoints also promotes collaboration, reuse, and long-term sustainability.

To better understand how our system could operate in real-world contexts, we developed several user stories. These stories describe hypothetical interactions with the system and help validate the completeness and flexibility of our design. While not all of these scenarios will be achieved during the research phase, they illustrate what a future-ready implementation should be able to support:

- As a search and rescue operator, I want to tell the drone to find people in a disaster area so I can quickly locate survivors without entering unsafe zones.
- As an engineer, I want to generate a 3D map of a building site so I can inspect construction progress remotely.
- As a researcher, I want to compare drone scans from two different times so I can analyze changes in the environment, such as erosion or structural shifts.
- As a technician, I want to override the drone at any time so I can maintain control during field testing or emergencies.

These user stories are more than hypotheticals – they inform how we prioritize feature development, design data workflows, and architect agent behavior. By grounding our system design in real user needs and practical challenges, we are ensuring that the final product is not only functional but also meaningful and impactful. These requirements help guide both our short-term development and long-term planning, supporting a system that is useful, adaptable, and ready for real-world deployment across multiple domains.

3.3 Concept of Operations

Our system follows a centralized architecture, where the Ground Control Station (GCS) handles all major reasoning and decision-making. The drone acts as the physical tool, while the GCS serves as the brain-enabling lightweight onboard hardware and advanced processing on the ground. When a user submits a natural language command through the dashboard, it is parsed by a Large Language Model (LLM) running on a Jetson companion computer. The LLM interprets the intent and breaks the request into high-level tasks the drone can execute.

These tasks are passed to an agent built using LangChain and LangGraph frameworks, which manage memory, sequencing, and reasoning. The agent translates each step into MAVLink commands compatible with flight stacks like PX4 and ArduPilot. It also monitors feedback – such as pose estimates, battery levels, and object detections – and adapts the plan in real time as needed.

While executing, the drone streams live video and sensor data to the GCS, where it is processed for object detection, depth mapping, and telemetry. These streams are rendered on a React-based dashboard, which presents synchronized 2D and 3D views to keep the user informed about the drone's position, environment, and decisions. This interface allows the human operator to understand not just where the drone is, but why it is taking specific actions, thereby reinforcing transparency and trust in the system's behavior.

To ensure safety and user control, a manual override is available through either an RC transmitter or dashboard control. This halts any ongoing or queued commands, and control can later return to the agent to resume operations. A live mission log and telemetry feed support replay and diagnosis, giving users full insight into system behavior.

For example, a command like “Scan the area and orbit the red car” triggers the drone to launch and begin a patterned scan. The visual module detects red objects, filters them to find a car, and computes an orbital path around the target. The drone orbits while capturing RGB and depth data, which updates the dashboard’s digital twin. If multiple red cars are found, the agent may request clarification to ensure correct targeting.

This architecture enables autonomous drone operation in unstructured environments without constant supervision. Meanwhile, the operator stays in the loop via a transparent interface that displays mission flow and allows real-time intervention. This hybrid model balances automation with human judgment, which is crucial for safety and mission flexibility. It also ensures the system is scalable and adaptable as new features are added.

Looking ahead, the same GCS could coordinate multiple drones, each handling distinct tasks or regions. Offloading intelligence to the ground reduces drone complexity and cost, while enabling shared memory and multi-agent planning. Centralizing coordination also makes it easier to integrate new drone types, payloads, or behaviors without firmware changes, supporting a modular and maintainable system architecture.

To support this kind of multi-agent coordination, we designed the GCS to store persistent memory across drone sessions, enabling shared mission context between flights. For instance, if one drone scans an area and another is deployed later, it can retrieve the prior map and extend the mission without repeating work. In more advanced cases, drones could negotiate task assignments or relay information through the GCS to avoid duplication or conflict.

By investing in modular software and standardized protocols from the start, we have created a foundation that is flexible enough to support both experimental deployments and long-term field applications. Whether used by a single operator in a disaster zone or scaled up for coordinated missions across wide areas, the system remains easy to understand, safe to use, and ready to evolve alongside future needs.

4. Design Documentation

4.1 Research and Investigative Process

4.1.1 Project Proposals

Before we settled on our final candidate for how our minimum viable product would be expressed, our team collected a variety of individual proposals, documented below.

4.1.1.1 Space Invaders

In this proposal, we would have two drones operating in a scenario inspired by the video game Space Invaders, with one drone adopting the role of the shooter and the other drone adopting the role of the evader. The first drone would be able to initiate virtual "lasers", which could be nearly instantaneous or slower-moving projectiles, or would be able to "place" smaller automated attackers, while the second drone would attempt to predict and dodge the incoming virtual attacks. Each drone would have its own agentic AI powering autonomous movement based on the virtual field of attacks and the locations of both drones. Potential stretch goals would include restricted visual sensors or required movement at periodic intervals.

4.1.1.2 Package Delivery System

In this proposal, we would have two drones simulate the delivery of packages. The first drone would perform higher-altitude larger-scale geographical scans to plan general road or path routing, and the second drone would perform lower-altitude smaller-scale local scans to "navigate" the route in real time while taking into account environmental hazards. Stretch goals would include the drones trading roles akin to a relay race, or the ground control station feeding virtual obstacles or route restrictions to either drone.

4.1.1.3 Local Weather and Environmental Monitoring System

This proposal concept involved equipping the drone with a suite of specialized sensors to autonomously fly routes and collect hyper-local atmospheric data, such as temperature, humidity, pressure, and air quality indexes. While the idea had practical applications in localized forecasting and pollution monitoring, it was ultimately not pursued due to the significant hardware overhead required. The process of sourcing, integrating, powering, and accurately calibrating numerous environmental sensors would have shifted the project's focus away from our core interest in developing agentic AI and autonomous control systems, and more towards a project centered on instrumentation and sensor engineering.

4.1.1.4 Infrastructure and Asset Inspection

This idea described the process of a drone autonomously scanning infrastructure and providing a health report for any changes that may need to be made. This idea could work on a variety of types of architecture: bridges, homes, skyscrapers, roads, or docks. The AI would need to be fine tuned for the specific type of architecture we chose, likely homes. Then. we would decide what we want the drone to look for. A few examples could be broken shingles, chipped paint, moldy/discolored walls, water leak. With enough training data to detect these issues and a smart enough agent, the drone could conceivably scan one home or even an entire neighborhood by itself and report issues in real time to a command station, which could be relayed to an HOA or repair company.

4.1.1.5 Emergency Response

This idea centered on an autonomous emergency response system featuring a VLM Agent to visually detect hazards, a Pilot Agent for navigation, and a "911 Agent" to autonomously contact emergency services. While a preliminary analysis revealed significant technical and ethical complexities with automating emergency communications, this investigation was crucial in validating the core architecture of using a visual perception agent and a programmatic pilot agent. This led to the strategic decision to first perfect a robust, general-purpose agentic framework that could be proven in a controlled environment before being adapted for specialized tasks like emergency response in the future.

4.1.1.6 Traffic Monitoring and Navigation System

In this proposal, two drones coordinate to manage foot traffic at crowded public events like festivals or outdoor markets. One drone flies overhead to monitor crowd density and detect bottlenecks using computer vision, while the second drone responds by projecting LED signs or playing audio prompts to guide attendees. The system relies on live heatmaps, AI-based congestion detection, and LLM-generated alerts such as, “Zone B nearing unsafe density. Suggest rerouting.” Stretch features include support for voice commands, QR code guidance, and adaptive patrol scheduling for different event modes.

4.1.1.7 Geospatial Point Cloud Collection

This idea focused on using a drone to autonomously scan and convert an area into a 3D digital twin using photogrammetry or SLAM-based methods. The drone would follow a structured flight plan and transmit image and depth data to a ground control station, where it would be processed into a point cloud for visualization. Preprocessing techniques such as downsampling and noise filtering could improve clarity and performance. This proposal laid the groundwork for our real-time SLAM-based mapping pipeline and informed several design decisions about our system’s perception and rendering stack.

4.1.1.8 Rationale for Final Proposal Selection

In narrowing down our proposal candidates, we evaluated them in light of three criteria:

- The agentic drone system should be capable of taking in commands.
- The agentic drone system should be able to generate plans.
- The agentic drone system should be able to issue tool calls.

After discussion among our project members and our sponsors, we settled on the geospatial point cloud proposal as the most interesting while meeting the criteria above along with requirements set out by our sponsors for our minimum viable product. In addition, we anticipated that this proposal would lay the groundwork for implementing other ideas that we had as well as for expanding with more goals under subsequent research groups in the future.

4.1.2 Drone Research

After deciding on a Ground Control Station (GCS)-centric architecture, the next critical step was selecting a physical platform that fit this model. The ideal platform would be a reliable and programmable body for our GCS brain, not a closed, all-in-one system. For this reason, the Holybro X500 V2 PX4 Development Kit was identified as the ideal platform.

A primary advantage of the X500 V2 is its Almost-Ready-to-Fly nature. The kit arrives with the motors and ESCs pre-installed, significantly reducing the most tedious and error-prone aspects of a drone build. No soldering or welding is required to build the drone. This was a critical strategic decision that saved dozens of hours of mechanical/electrical assembly and significantly reduced the chance of an assembly issue. This allowed the team to immediately focus on the project's true innovation: the software, AI integration, and control systems.

The 500mm wheelbase places the drone in a size class that is large enough to be stable and carry a meaningful payload, yet small enough to be easily transportable and tested safely. The frame's carbon fiber plates provide a large, sturdy mounting surface for our core electronics, including the Pixhawk flight controller, power systems, and the Raspberry Pi 5. Crucially, the integrated payload rails provide a secure, standardized mounting point for the ZED 2i camera, ensuring it has a clear, forward-facing view and is isolated from vibrations.

Furthermore, the selection of this kit was fundamentally driven by its open-source core. The platform is built around the Pixhawk 6C flight controller running the PX4 Autopilot firmware. This open-source stack provides access to the drone's control systems via the MAVLink protocol. This well-documented, standardized communication protocol is the essential bridge that allows our GCS-based agent to send high-level programmatic commands to the drone. This stands in stark contrast to the locked-down, proprietary SDKs of many commercial drones, which would have made our GCS-centric control architecture difficult to implement.

Beyond the core platform selection, the team conducted a broader evaluation of drones and sensor payloads to understand our system's flexibility, integration potential, and constraints. Key factors were compared including: onboard compute compatibility, flight time, expandability, SDK accessibility, sensor support, and ROS/MAVSDK integration. Commercial options such as

DJI and Skydio were initially attractive for their polished design and reliable flight performance, but they introduced significant barriers for our agentic AI integration due to locked-down APIs and limited SDK control.

Alternatively, modular research kits like PiDrone 2 and Raspberry Pi builds offered full control and open-source support. These platforms allowed us to freely modify firmware, connect directly with our onboard computer, and leverage open messaging protocols. However, they also required more engineering overhead: tuning flight controllers, calibrating IMUs, and integrating sensor data pipelines would have delayed progress. The Holybro X500 provided a practical middle ground between ease of use and full-stack flexibility.

An analysis of sensor options for real-time spatial data acquisition was conducted as well. We selected the ZED 2i stereo camera due to its depth sensing, ROS support, and ability to stream both RGB and point cloud data. While LiDAR was evaluated as a future enhancement for more accurate spatial modeling, it was deferred as a stretch goal due to its high cost, power consumption, and added payload weight. This prioritization ensured that the core system remained lightweight and within the payload budget of the X500 airframe.

Another essential part of the research focused on communication protocols and SDK interoperability to support the GCS-centric architecture. MAVLink emerged as the ideal protocol for command and control due to its widespread support in the PX4 ecosystem, and MAVROS bridges were selected to simplify integration. The research also explored more advanced features, such as the potential for context-aware task prioritization within the drone's autonomous workflow. This concept, where the drone could alter its behavior based on real-time scene understanding, remains a stretch goal but highlights a path for future system intelligence.

The modular design of the X500 platform also supports future sensor integration and experimentation. As the project evolves, the team anticipates testing additional payloads such as thermal cameras or acoustic sensors. This foundational research and platform comparison process resulted in a modular, agent-ready drone architecture that is compatible with ROS, supports AI integration, and maintains flexibility for future upgrades and emerging technologies.

4.1.3 Agentic and General AI Research

One of the critical goals of our project is to leverage an agentic AI framework embedded within our ground control station in order to interpret natural language and translate them into actionable drone behaviors. This interpretation and translation will be carried out through a collaborative network of autonomous software agents, each with specific roles. Within this network, these agents will interact dynamically by sharing data and tools via an interoperable software architecture, with the goal of permitting operators to issue simple, intuitive commands and having the agents respond and execute them in real time.

To accomplish this, we looked into the following software components:

4.1.3.1 Model Context Protocol

The Model Context Protocol (MCP) was recently introduced by Anthropic to help standardize how tools, data, and other environmental contexts are communicated to large language models with the goal of developing AI agents. It has subsequently seen widespread adoption across the AI ecosystem, as it enables consistent interfacing and interoperability without having to create custom connectors for every combination of agent and tool.

We considered using the MCP as the backbone of inter-agent communication in our system, allowing each agent to access a shared context, including:

- **Drone telemetry**, including location and current velocity,
- **Visual and sensor input**, such as the RealSense camera stream,
- **Environmental maps**, particularly the digital twin and point cloud data, and
- **Toolkits and algorithms** that may be commonly called by agents.

One potential application of the MCP to our agentic system would be to have our agentic AI be able to act as an MCP client, permitting it to make arbitrary tool calls to an MCP server to access additional data. However, at the current time, we have designated MCP integration into our agentic AI as a stretch goal.

4.1.3.2 Ollama and Localizing Large Language Models

In alignment with our commitments to data localization and security, we plan to run all large language models locally on our NVIDIA Jetson hardware. Ollama is an ideal solution for local LLMs, as the platform is optimized for the deployment of lightweight language models. By using Ollama, our system will be able to achieve:

- **Offline operation**, without the need for internet connectivity,
- **Data privacy**, as all data and human prompting remains local,
- **Model flexibility**, since Ollama allows for the facile swapping of LLMs,
- **Low-latency inference**, operating with real-time responsiveness even on limited hardware,
- **Lower costs**, by avoiding having to run API calls for proprietary LLMs, and
- **Full control over system prompts**, as we do not have to deal with the effects of system prompts that may interfere with our goals

In researching localized models, we observed that there were potential drawbacks compared to calling APIs for a typical industry-leading large languae model, including slower tokenization and generation, weaker output with lower parameters, and a lack of system prompts. Nevertheless, the importance of data privacy and offline operationality led us to focus on localized models for the time being. In the case that we encounter significant speed bottlenecks or insufficiently relevant or detailed model output, we may consider switching to a nonlocal model.

In our investigative stage, we examined a variety of large language models in our investigative stage, but our ultimate choice of LLM will eventually function to parse human instructions into structured goals and then interpret them into corresponding code for drone movement. Depending on how our system develops, we could deploy multiple instances of a single LLM or different instances of different LLMs for multistep interpretation of human commands. For visual language model tasks such as interpreting image or video data, we looked into multimodal variants of the aforementioned models.

Further discussion on the individual models can be found in sections 4.1.3.6 and 4.1.3.7.

4.1.3.3 LangChain and LangGraph

LangChain and LangGraph will allow us to establish a controlled flow involving multiple agents, and both are compatible with the MCP server. Below is one potential pipeline for our agentic system (see *Figure 1.0* for a visual flow chart):

1. A human-in-the-loop sends a command via the React dashboard.
2. An LLM agent will receive the natural language command and interpret it.
3. The VLMAgent can query or process the incoming video feed if necessary.
4. Another LLM agent will synthesize the general flight plan.
5. The PilotAgent produces the corresponding MAVSDK-Python instructions.

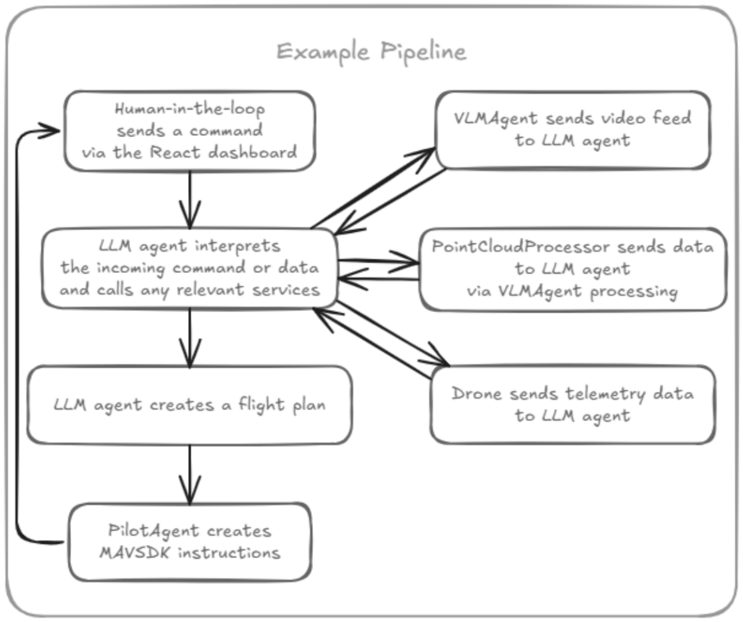

Example Pipeline
Human-in-the-loop sends a command via the React dashboard
VLMAgent sends video feed to LLM agent
LLM agent interprets the incoming command or data and calls any relevant services
PointCloudProcessor sends data to LLM agent via VLMAgent processing
LLM agent creates a flight plan
Drone sends telemetry data to LLM agent
PilotAgent creates MAVSDK instructions

Figure 1.0 – Example Pipeline *of a potential workflow for LangChain or LangGraph*

LangChain and LangGraph are able to facilitate any conditional branching our project might require, such as if we need the PointCloudProcessor to forward its point cloud data to the command interpreter agent if the VLMAgent does not immediately identify or find an object. Furthermore, LangChain and LangGraph can allow an AI agent to cycle through a series of tasks or repeat a specific task, such as if we would need the DroneAgent to continue proceeding with autonomous navigation, or if we would require the VLMAgent to continuously send image data until an object is discovered, depending on the tools that we create and the tuning of various language models.

4.1.3.4 Docker and Containerization

For portability across multiple team members working on shared aspects of our agentic AI system, we looked into utilizing Docker and containerization. This approach would offer several advantages:

- **Modular development**: Components can be created and tested independently.
- **Dependency isolation**: Each container can have its own libraries, drivers, software versions, and language versions.
- **Reproducibility**: Deployments are consistent across different environments.
- **Version control**: Reversions to more stable builds are facile.
- **Edge deployment**: There is minimal system reconfiguration required while using Docker.

4.1.3.5 SmolAgents Agentic Framework

The final piece of the software puzzle was the agentic framework, the software that orchestrates the LLM and its tools. After evaluating several options, we selected SmolAgents due to its design philosophy and suitability for our project.

SmolAgents is a minimalist, lightweight Python framework designed specifically for creating tool-using agents that follow the ReAct (Reason+Act) paradigm. While more extensive frameworks like LangChain offer a vast number of features, their complexity can introduce significant overhead and a steeper learning curve.

The rationale for selecting SmolAgents was twofold:

1. Simplicity and Clarity: The framework has very little boilerplate code, making it incredibly straightforward to define our custom tools (like the PilotAgent and VisionAgent) and to trace the agent’s logic during execution. This is invaluable for rapid prototyping and debugging in a research context.
2. Lightweight Performance: Its minimal overhead ensures that the framework itself consumes very few resources, which is critical when running the entire AI stack on the embedded NVIDIA Jetson platform.

This choice allows us to implement the powerful ReAct loop efficiently, enabling our agent to reason, act, and observe in an iterative cycle without being bogged down by unnecessary abstractions. A diagram of our agent's workflow is shown below (*Figure 1.1*).

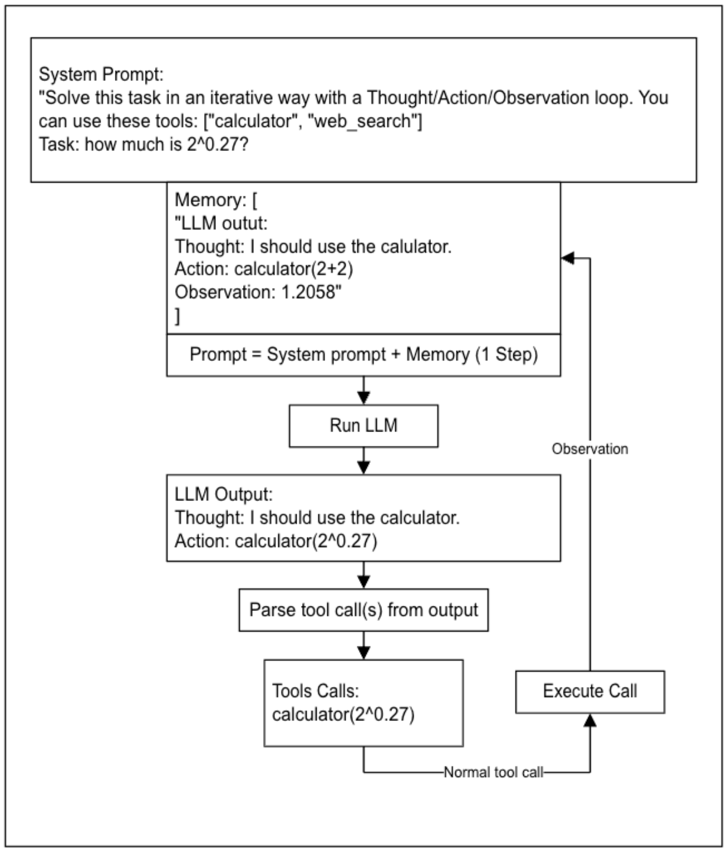

System Prompt:
"Solve this task in an iterative way with a Thought/Action/Observation loop. You can use these tools: ["calculator", "web_search"]
Task: how much is 2^0.27?
Memory: [
"LLM outut:
Thought: I should use the calulator.
Action: calculator(2+2)
Observation: 1.2058"
]
Prompt = System prompt + Memory (1 Step)
Run LLM
Observation
LLM Output:
Thought: I should use the calculator.
Action: calculator(2^0.27)
Parse tool call(s) from output
Tools Calls:
calculator(2^0.27)
Execute Call
Normal tool call

Figure 1.1 – SmolAgents Diagram: *General Workflow Concept*

4.1.3.6 Vision Language Model Research

For the system to perceive its environment, our research extended to Visual Language Models (VLMs). A VLM combines a computer vision "encoder" with a Large Language Model to interpret images based on text prompts. The vision encoder processes a video frame, converting it into a numerical representation that the LLM can understand. This allows the agent to effectively "ask questions" about what the drone sees.

Our investigation included models like LLaVA (Large Language and Vision Assistant), which are well-documented multimodal models compatible with our local deployment strategy. In our system, the VLM serves as a specialized tool. Instead of just generating conversational text, it is prompted to return structured, machine-readable data. For example, given the prompt "Find the red chair," the VLM would be instructed to return a JSON object containing the object's label and its bounding box coordinates within the image. This structured output is essential for passing precise information to other agents, like the PilotAgent, for action.

4.1.3.7 Language Model Research

Our research into Large Language Models (LLMs) focused on identifying models suitable for running locally on our NVIDIA Jetson GCS via the Ollama platform. The goal was to find a balance between reasoning capability, performance, and resource consumption.

We investigated several prominent open-source models:

- Mistral Models (e.g., Mistral 7B): These are highly capable general-purpose models known for their excellent performance-to-size ratio. They are well-suited for understanding user intent and general-purpose planning.
- CodeLlama Models (e.g., CodeLlama 13B): These models are specifically fine-tuned by Meta for code generation. This makes them an ideal choice for our primary CodeAgent paradigm, where the LLM's main task is to write a correct Python script to execute the mission.

A key differentiator between models is their parameter count (e.g., 7B for 7 billion parameters). This number reflects the model's size and complexity; a higher parameter count generally means more knowledge and nuanced reasoning but also requires significantly more memory (VRAM) and processing power.

To deploy these models effectively on edge hardware like the Jetson, we researched quantization. This is an optimization technique that reduces the precision of the model's numerical weights (for instance, converting from 16-bit floating-point numbers to 4-bit integers). The primary benefit is a drastic reduction in the model's file size and memory footprint, allowing larger models to run on resource-constrained devices with faster inference speeds. The trade-off is a minor, often negligible, potential loss in response quality, which is acceptable for our use case.

4.1.4 Photogrammetry and Visualization Research

4.1.4.1 Evaluating Mapping Approaches

To enable real-time 3D environment reconstruction, we initially explored both traditional photogrammetry and SLAM (Simultaneous Localization and Mapping) as candidate technologies. Traditional photogrammetry relies on analyzing multiple still images taken from varying angles to reconstruct highly detailed 3D models. It is widely used in surveying, architecture, and archaeology due to its accuracy and ability to generate photorealistic surface reconstructions. Tools like COLMAP and Meshroom have made this method more accessible, but they still rely on the full dataset being available before model generation begins.

However, despite its precision, this method posed a critical limitation for our use case: it is inherently batch-processed. All images must be captured before the reconstruction can begin, and the computational load can be intensive, sometimes taking hours for large datasets. This made it incompatible with our primary requirement of real-time responsiveness for disaster mapping or autonomous inspection missions.

SLAM, in contrast, operates incrementally. It builds a map and estimates the drone's position in real time using video streams or depth data as input. This continuous reconstruction aligns well with our need to provide live spatial feedback to the user through our React dashboard. SLAM frameworks such as RTAB-Map and ORB-SLAM2 were considered for their open-source compatibility and real-time capabilities. While SLAM may not produce photorealistic outputs and can be less geometrically precise under noisy conditions, it enables the type of dynamic updates that our MVP depends on. Based on this comparative analysis, we prioritized a SLAM-based pipeline for real-time feedback and chose to treat photogrammetry as a secondary tool for post-mission accuracy refinement.

4.1.4.2 Evaluating RViz and Exploring Lightweight Alternatives

During early simulations, I explored using RViz – a visualization tool built into the Robot Operating System (ROS) – to display point cloud and camera data in real time. RViz offered useful built-in support for laser scans, depth maps, and occupancy grids, and was capable of providing a live view of SLAM-generated maps or LiDAR feeds. However, the ROS-based rendering introduced significant integration challenges.

Specifically, streaming RViz output into a web dashboard required intermediate services to convert ROS messages into web-compatible formats, such as JSON or MJPEG. This middleware increased latency and system complexity, leading to delays in the video and depth stream. Additionally, RViz's heavy dependencies and lack of native web support made it difficult to integrate within our Next.js/React frontend. This bottleneck conflicted with our need for lightweight, real-time updates that support fast user interactions.

As a result, I pivoted to a simpler approach using OpenCV and Flask to stream the drone's video feed in MJPEG format. This allowed us to embed a lightweight camera preview directly into the dashboard with minimal delay. It also opened the door to building custom overlays and annotation tools using browser-native technologies. In future iterations, we plan to upgrade this architecture with gRPC or WebSockets for improved scalability and data flexibility. Additionally, we began investigating how we might later layer real-time telemetry and object bounding boxes on top of the feed to support visual tracking and feedback loops with the VLM system.

4.1.4.3 Automated Photogrammetry and Point-Cloud Processing

To reduce manual effort during scan reconstruction, I planned a fully automated photogrammetry pipeline using headless COLMAP. This system monitors a specified folder for incoming images from the drone. Once a new dataset is detected, the pipeline automatically initiates the SfM and MVS processes and exports a 3D point cloud and textured mesh. This was achieved through a combination of Python scripts, GPU configuration settings, and pre-compiled binaries of COLMAP and OpenMVS.

This automation allowed us to simulate a "one-click" workflow for generating spatial data. During testing, I also explored the integration of OpenMVS and CloudCompare to refine point cloud outputs. CloudCompare allowed us to apply preprocessing steps such as noise filtering (e.g., radius-based outlier removal), voxel-based downsampling for performance optimization, and normalization for consistent alignment across missions. Open3D, another promising tool, was evaluated for scripting-based cloud filtering and mesh generation directly within Python.

Together, these steps formed a robust scan-to-model pipeline that could support both MVP testing and future expansion. The ability to automate dense reconstruction makes this approach suitable not only for rapid prototyping, but also for hands-free field deployment. The design anticipates integration with the React dashboard for on-demand model viewing or download, with later versions possibly incorporating cloud storage or WebGL rendering.

4.1.4.4 Rendering in Unity Using Pcx

To visualize 3D models, I evaluated Unity as our rendering engine due to its versatility, file format support, and WebGL compatibility. I paired Unity with the Pcx plugin, which is specifically designed to render point clouds efficiently using GPU shaders and custom materials.

Pcx supported direct import of .ply files, which fit naturally into our COLMAP-to-Unity pipeline. During tests, it was able to render millions of points with minimal frame drops, making it well-suited for future scan visualization on both desktop and browser-based platforms. I experimented with dynamic coloring techniques using Pcx shaders to highlight elevation changes, object classification, and scan density. This helped us create a more informative visual experience, especially for field operators who may rely on fast interpretation of spatial data in disaster scenarios.

The use of Unity and Pcx enabled us to build a visualization layer that is not only technically powerful but also aesthetically clear for end users. I documented this toolchain as the default renderer for future 3D display work, ensuring consistency and extensibility in our mapping subsystem. These tests were also useful in understanding performance trade-offs for larger, denser scans, helping shape our visualization targets for later phases.

4.1.4.5 Preparing for Real-Time and WebGL Rendering

While our MVP goals emphasized offline and semi-real-time scan feedback, I also prepared the system for future real-time 3D rendering in the browser. The exported builds successfully rendered .ply files using Pcx and were viewable in a modern web browser without requiring local installations.

WebGL, while powerful, introduces limitations such as reduced shader compatibility, memory constraints, and slow load times for high-density models. To address these, I studied real-time optimization strategies, including Level of Detail (LOD) management, occlusion culling for non-visible geometry, and shader-based simplification that reduces visible point density without degrading overall fidelity.

In addition to performance tuning, I considered how WebGL export workflows might support interactive features like user-placed annotations or object labeling. These optimizations will allow our system to scale gracefully to larger environments or multi-drone scenarios in the future. More importantly, they position our dashboard for future upgrades where users can interact directly with live spatial maps in-browser, without needing high-end workstations or special software.

4.1.5 Multi-Drone Coordination and AI Research

4.1.5.1 Planning for Scalable Multi-Agent Systems

From the outset of the project, I recognized that our system architecture needed to eventually support multiple drones working in tandem. While the MVP focused on a single autonomous unit, I conducted early research into how multi-drone systems could be coordinated in scalable, modular ways. My approach was to identify practical coordination methods that could evolve from human-assigned task delegation to fully autonomous agentic behavior in future versions.

I started by exploring basic manual coordination strategies. One of the simplest and most effective involved dividing the mission area into grid sectors, with each drone assigned a non-overlapping region. This method reduced complexity, prevented redundant scanning, and made later map merging more predictable. It also aligned well with stretch goals around post-processing, allowing us to cleanly separate data by spatial origin.

4.1.5.2 Task Distribution and ROS-Based Communication

To understand how drones could coordinate more autonomously, I researched how the Robot Operating System (ROS) could be used for inter-drone messaging. I focused on strategies that assigned each drone its own set of ROS topics to prevent message conflicts and GPS drift issues. This setup allowed each drone to publish its position, mission status, and sensor data independently, while still reporting to a centralized controller or dashboard.

I studied several relevant ROS message types and looked into building lightweight publisher/subscriber nodes that would allow real-time coordination. For example, one drone could broadcast a “sector complete” signal, prompting another to shift zones or update its path plan. These mechanisms laid the groundwork for swarm-like coordination, even within a semi-autonomous or partially human-supervised system.

4.1.5.3 AI-Driven Coordination Frameworks

Beyond low-level messaging, I researched higher-order coordination using agentic AI frameworks. I focused on LangChain and LangGraph, two Python-based libraries built to support multi-agent task flow management. These tools allow each drone or system module to behave as an independent “agent” capable of planning, observing, and reporting actions.

In practice, this means we could design a system where Drone A scans Sector 1 and detects incomplete coverage. It can then send a request to Drone B, via LangGraph’s messaging logic, to assist with that sector. Alternatively, the central Ground Control Station (GCS) could run a LangChain-based planner that dynamically assigns tasks based on status reports from each drone.

This agentic structure supports a wide range of future behaviors: autonomous scan handoff, coverage optimization, obstacle avoidance, and even collaborative object detection missions. Though too ambitious for full implementation in the current semester, it shaped our architectural vision and validated our decision to build around a centralized logic layer.

4.1.5.4 Swarm Behavior and Control Models

To go a step further, I explored common models for multi-agent control used in robotics literature. These included:

- Boids Algorithms – Simulating flocking behavior with basic attraction, repulsion, and alignment rules that govern how individual agents behave in a group. This model, while originally intended for computer graphics, has become foundational in swarm robotics for enabling decentralized coordination without central control. It offered a simple yet powerful way to conceptualize emergent group behavior that could be adapted for autonomous drone formations.
- Potential Fields – Generating gradient-based maps that represent forces in the environment, such as attractive forces pulling drones toward mission goals or repulsive forces pushing them away from obstacles. This approach enables continuous motion planning in cluttered environments and can be used to guide drones along efficient, collision-free paths. It also supports dynamic updates as the environment changes, which is useful for real-time scanning and exploration missions.
- Behavior Trees – Structuring complex decisions into prioritized, modular branches that can be evaluated in real time. Behavior trees are widely used in game AI and robotics for managing high-level task logic while preserving flexibility. This model offers a scalable and readable way to manage conditional actions, allowing drones to react autonomously to context such as new operator commands, sensor triggers, or mission state changes.

These models provided insight into how a multi-drone swarm could dynamically distribute itself, avoid collisions, and adapt to changing terrain or mission parameters. While we did not implement any of these methods directly in our MVP, they strongly informed my thinking around how to modularize coordination logic for future extensibility. For example, they inspired ideas for designing reusable action nodes in our AI planning stack or setting up distributed decision-making frameworks that could work alongside a centralized GCS agent. Exploring these models early in development helped shape the architectural flexibility of our system and laid the foundation for implementing more sophisticated multi-agent behaviors in future phases.

4.1.5.5 Planning Around Human-AI Collaboration

Throughout my research, I emphasized the need for human-in-the-loop controls within the coordination system. This included features like task override, visual mission summaries, and operator interventions. We did not want to build a black-box AI swarm; we wanted a transparent system where users could understand why a drone made a decision, and step in if needed.

By designing coordination protocols that report back to a centralized GCS, and by integrating natural language summarization via LLMs, we created space for collaborative autonomy. This design also enables eventual voice commands such as "Send drone to Sector C," which can be routed through an AI planner and translated into real drone behavior.

4.1.6 Ground Control Station Research

The most foundational design decision in the development of the LION Autonomous Drone was the selection of its core computing architecture. This choice dictates how and where data is processed, directly influencing the system's capabilities, physical design, and development workflow. Two primary architectural models were evaluated: a fully onboard processing architecture, where all AI computation occurs on the drone itself, and a GCS-centric architecture, which strategically splits processing tasks between the drone and a powerful ground station. After a thorough analysis of the project's objectives and constraints, the GCS-centric model was selected as the optimal approach.

The GCS-centric architecture operates on a principle of specialized roles. In this model, the drone is treated as a highly capable mobile sensor platform and actuator. Its primary responsibilities are limited to the reliable acquisition of high-fidelity sensor data and the precise execution of flight commands. Onboard processing, handled by the Raspberry Pi 5, is focused exclusively on data management tasks such as capturing the RGB and depth streams from the ZED 2i camera, performing necessary compression, and streaming this data over a high-bandwidth wireless link to the Ground Control Station. The drone does not perform any high-level reasoning or AI model inference onboard.

Conversely, the Ground Control Station functions as the centralized intelligence hub of the entire system. All computationally intensive tasks are offloaded to this ground-based computer, which is an NVIDIA Jetson platform chosen for its powerful GPU acceleration. The GCS is responsible for running the core AI stack, which includes the Large Language Model for command interpretation and planning, the Visual Language Model for scene analysis, the generation of 3D point clouds from the received depth data, and the execution of the main agentic framework that orchestrates the entire operation. This division of labor allows the project to leverage processing power that would be far too great to place on the drone, enabling the use of more sophisticated and capable AI models. The two halves of the system are connected by two distinct communication channels: a standard Wi-Fi network for the high-bandwidth video and depth data streams, and a dedicated, low-latency SiK telemetry radio link for the robust transmission of essential command-and-control MAVLink messages.

Onboard AI Architecture

- Computational Power: Limited by size, weight, and power (SWaP) constraints of an onboard computer. Requires smaller, quantized AI models with reduced performance.
- Payload Weight & Flight Dynamics: Significantly increased. The weight of an AI-capable compute module, its cooling system, and dedicated power source directly reduces flight time and agility.
- Thermal Management: Highly challenging. Dissipating heat from a constantly running GPU in an enclosed, vibration-prone airframe is difficult and can lead to thermal throttling, reducing performance.
- Power Consumption: High onboard consumption. An AI processor would draw significant power from the main flight battery, drastically shortening mission duration. Requires larger, heavier batteries.
- Development Agility: Slow and cumbersome. Debugging and software updates require physical interaction with the drone. Managing onboard dependencies and libraries is complex.

GCS-Centric AI Architecture

- Computational Power: High. Leverages powerful GPUs (e.g., NVIDIA Jetson or desktop-class) on the ground. Enables use of larger, more accurate, and more capable AI models.
- Payload Weight & Flight Dynamics: Minimized. The drone carries only essential components (camera, Pi 5), keeping it lightweight. This maximizes flight endurance and maneuverability.
- Thermal Management: Simplified. The ground station can employ large, active cooling solutions (fans, heatsinks) ensuring stable, sustained performance without risk of overheating.
- Power Consumption: Minimal onboard consumption. The drone's power budget is primarily dedicated to propulsion. The GCS uses a stable, external power source, independent of the drone's battery.
- Development Agility: Fast and efficient. AI software can be developed, tested, and debugged on a standard workstation. Facilitates rapid iteration and integration with simulation tools.

A deeper analysis of these criteria reveals the advantages of the GCS-centric model for this project. The computational requirements of modern Large Language and Visual Language Models are immense. An onboard architecture would necessitate the use of heavily optimized and quantized models, which inherently trade accuracy and capability for a smaller footprint. The GCS-centric approach imposes no such compromise, granting the system access to the full power of the NVIDIA Jetson. This allows for higher-resolution analysis, more nuanced language understanding, and more complex reasoning, all critical to achieving the project's goals. It also enables rapid model experimentation, since developers are not constrained by onboard compute limits. Models can be swapped, retrained, or fine-tuned on the ground with minimal disruption.

The physical implications on the drone itself are equally significant. Adding a compute module capable of AI inference, along with its necessary cooling and power hardware, would add hundreds of grams to the airframe. This increase in payload weight demands more thrust from the motors, which increases power draw and reduces flight time. The GCS-centric design breaks this cycle by keeping the drone's configuration light and efficient, dedicating the onboard power budget to propulsion and maximizing endurance. It also opens up design freedom to explore more compact or energy-efficient airframes in the future.

Thermal management presents another obstacle for onboard architectures. Sustained AI workloads generate substantial heat, and failure to dissipate it leads to thermal throttling. Designing effective cooling for a compact, vibrating drone frame is a non-trivial challenge. The GCS, operating in a stable ground environment, can use simple cooling solutions, allowing AI models to run at peak performance indefinitely.

Finally, an onboard-centric model would dramatically slow development. Every software update would require connecting to the drone and managing complex onboard dependencies. The GCS-centric model decouples software development from drone hardware, allowing the team to build and test agents on a standard computer using simulated data. This enables parallel development and a far more efficient debugging process, essential for completing a project of this complexity within a single academic year. It also allows team members to work asynchronously without needing physical access to the drone at all times.

In conclusion, while a fully self-contained, onboard AI system may represent an ideal future, the GCS-centric architecture provides the most practical and powerful path forward. It maximizes AI capability, reduces airframe complexity, and creates an agile development environment making it fundamental to the LION project’s feasibility and success.

4.2 Design Diagrams

4.2.1 Block Diagram

The block diagram below (*Figure 2.0*) illustrates a high-level view of the architecture behind our project. We decided to separate the physical flight hardware from advanced computation, allowing us to keep the drone lightweight while making other parts of the system more modular. The diagram at the bottom of this page highlights interactions between three major components:

- **Drone** – The drone is a Holybro X500 V2 equipped with a Pixhawk 6C flight controller, a Raspberry Pi 5, a ZED 2i camera, and a Holybro M10 GPS module.
- **Jetson Orin Nano** – Part of the ground control station, the Jetson will handle all computationally-intensive software, including visual language models, large language models, and SLAM algorithms.
- **React Dashboard Laptop** – Part of the ground control station, the laptop houses the four-component React dashboard, which needs to connect to both the drone and the Jetson components

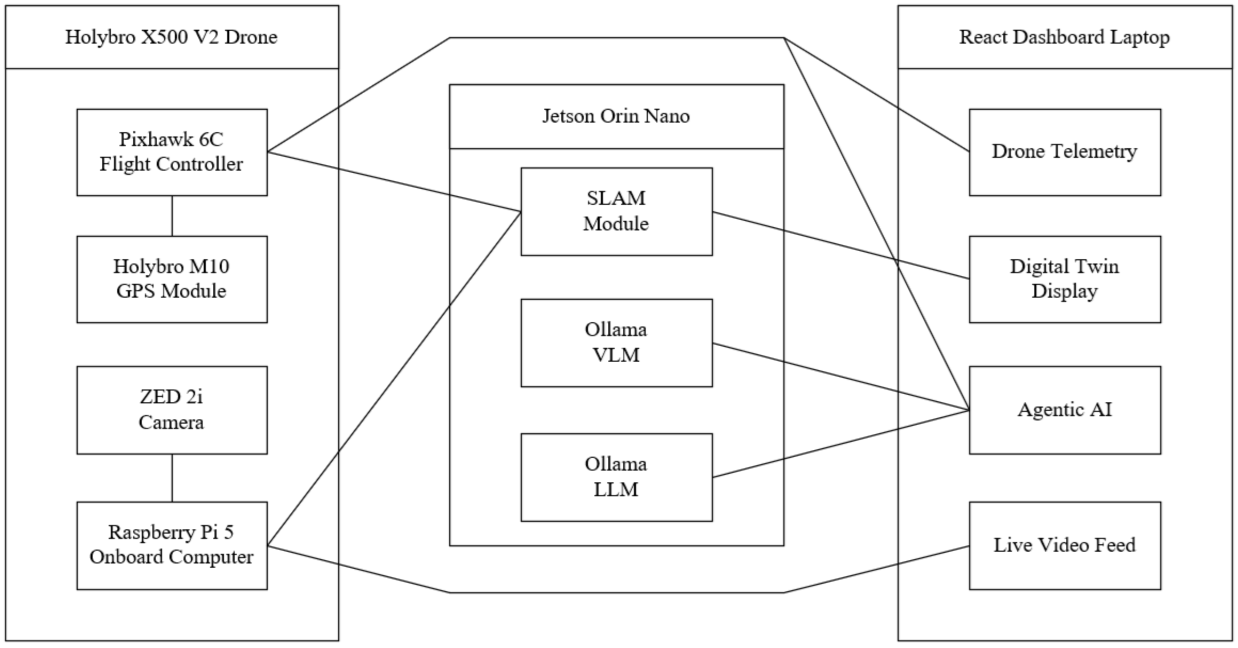


***Figure 2.0 – Block Diagram** showing interactivity between the drone, Jetson, and dashboard*

4.2.2 Use-Case Diagram

Below (*Figure 2.1*) is a simplified version of the use case diagram. An exhaustive version would be redundant with all the tool calls. The core thing to keep in mind is who all the actors are and how they interact. There is only one user actor, which will be a drone operator. They will be able to start their mission, use the UI to start missions, issue commands, and end missions. Executing plans will invoke ROS code and our core libraries on the ground control station. Executing plans will also store telemetry in our postgres database or actor number two. The final actor, the s3 bucket will only be interacted with when uploading the SLAM data to persistent storage, or when storing pictures/videos.

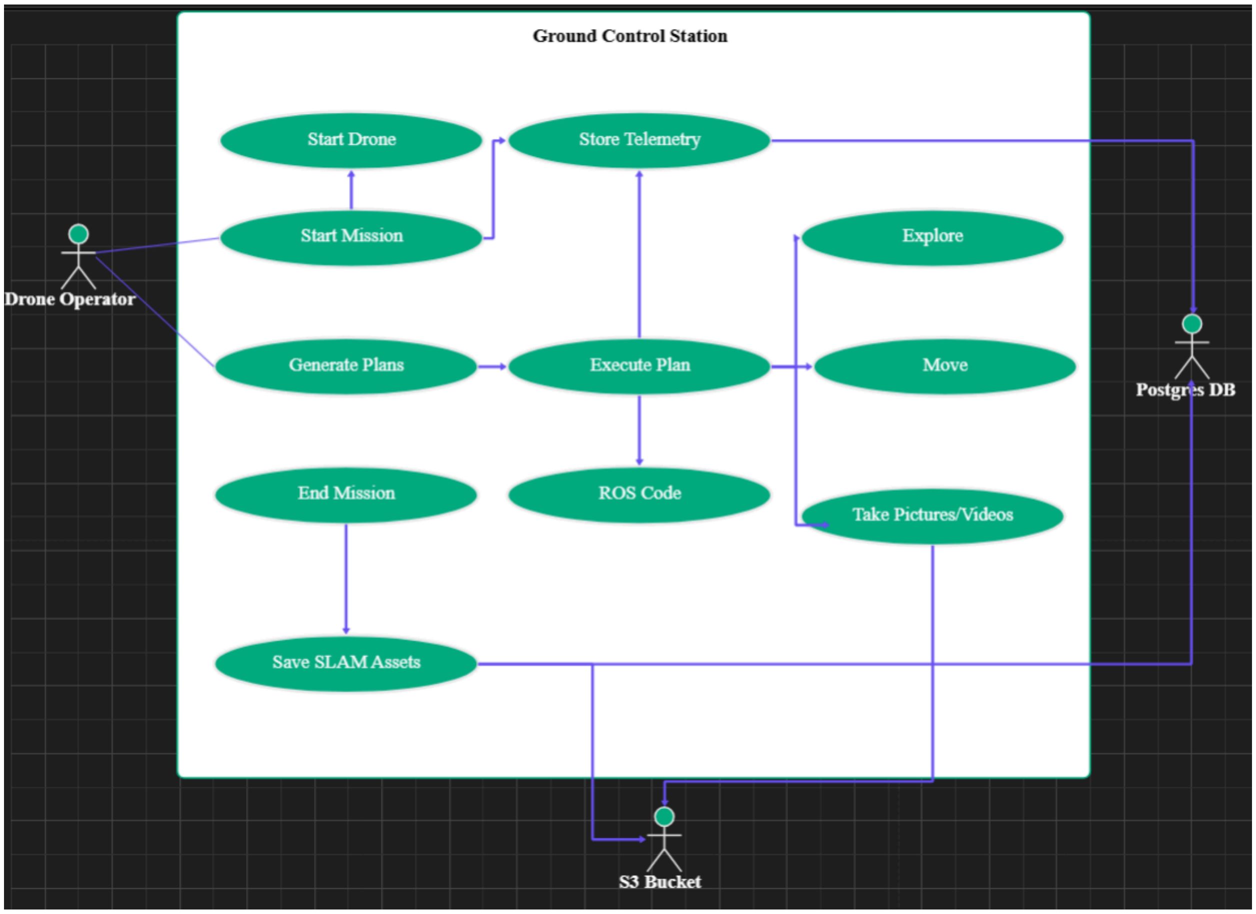


Figure 2.1 – Use-Case Diagram

4.2.3 Activity Diagram

This diagram (*Figure 2.2*) illustrates the system's primary loop. A user's natural language command is received by the Ground Control Station, where an agentic framework uses a Large Language Model to interpret the mission. In an iterative loop, the LLM decides whether to query the Visual Language Model for perceptual data or to direct the PilotAgent to perform a flight maneuver. When a flight action is commanded, the PilotAgent translates the goal into MAVSDK instructions, which are sent via the MAVLink protocol to the drone's flight controller for execution.

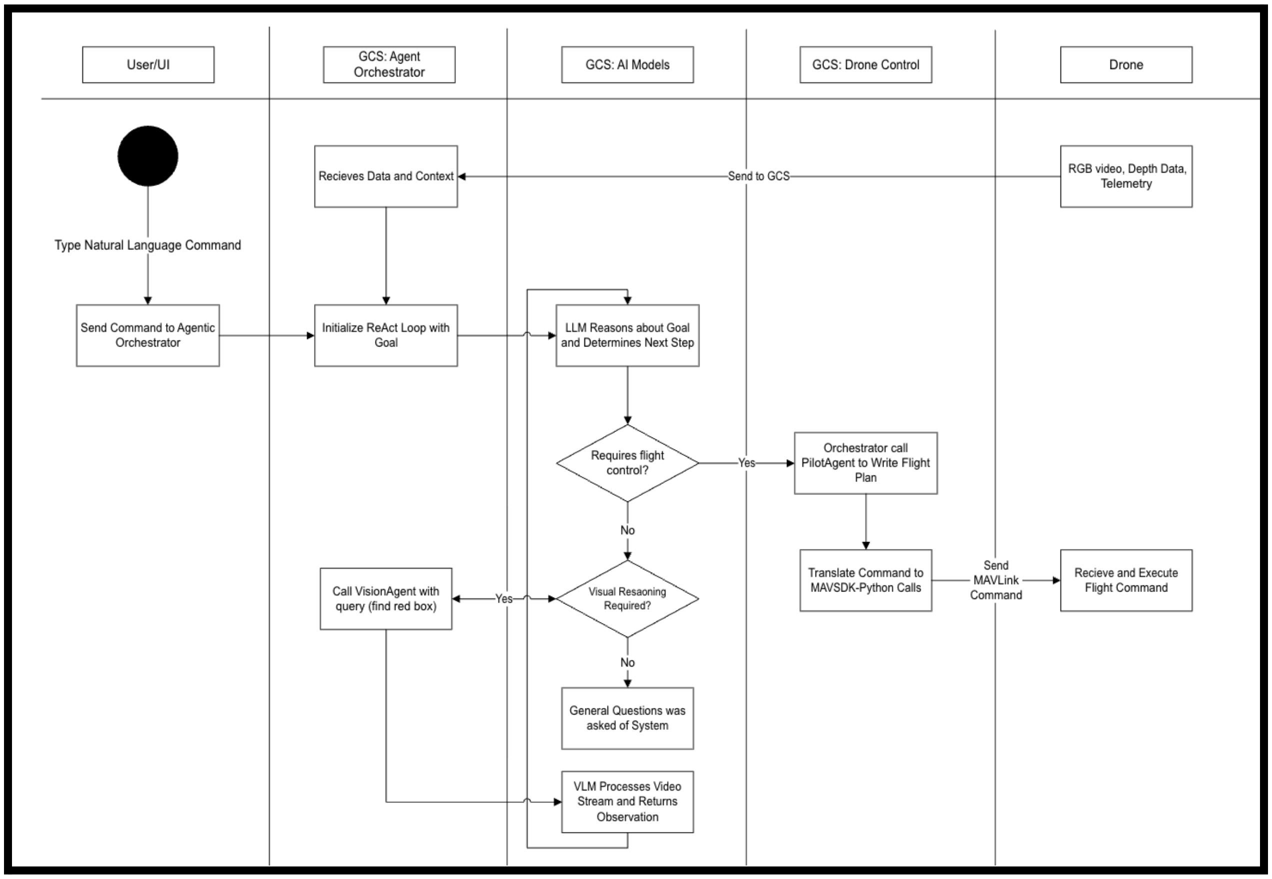

User/UI
GCS: Agent Orchestrator
GCS: AI Models
GCS: Drone Control
Drone
Recieves Data and Context
Send to GCS
RGB video, Depth Data, Telemetry
Type Natural Language Command
Send Command to Agentic Orchestrator
Initialize ReAct Loop with Goal
LLM Reasons about Goal and Determines Next Step
Requires flight control?
Yes
Orchestrator call PilotAgent to Write Flight Plan
No
Translate Command to MAVSDK-Python Calls
Send MAVLink Command
Recieve and Execute Flight Command
Call VisionAgent with query (find red box)
Yes
Visual Resaoning Required?
No
General Questions was asked of System
VLM Processes Video Stream and Returns Observation

Figure 2.2 – Activity Diagram *with individual lanes*

4.2.4 Entity-Relationship Diagram

Below is our entity-relationship diagram (*Figure 2.3*). From a database perspective, everything will start with the user. Users will have a One-To-Many (OTM) relationship with missions, each representing a session with a drone. A mission will have a name (like “Scan football field”) and include start/end timestamps. It will also have a One-To-One (OTO) relationship with slam_assets. We will take the output of SLAM from RTAB-Map and upload it to S3 so that anyone, anywhere, can download the assets and view the digital twin. Through a mission, users will talk to the agent via chatlogs, stored as mission_commands. Each command will generate agent_plans, composed of 0 or more steps. During the mission, messages will be added to the telemetry section of the React dashboard and stored as telemetry_logs. If the message comes from the drone (source), it will include drone_telemetry such as position and status. All in all, when a mission is complete, we should have more than enough logs and information for task tracking and debugging. Not shown, but if and when we give the drone tools for pictures/videos, we’ll create an OTM mission_capture table with an S3 link.

The authentication schema has been simplified for brevity. There will be a few more tables to store sessions, verifications, and accounts.

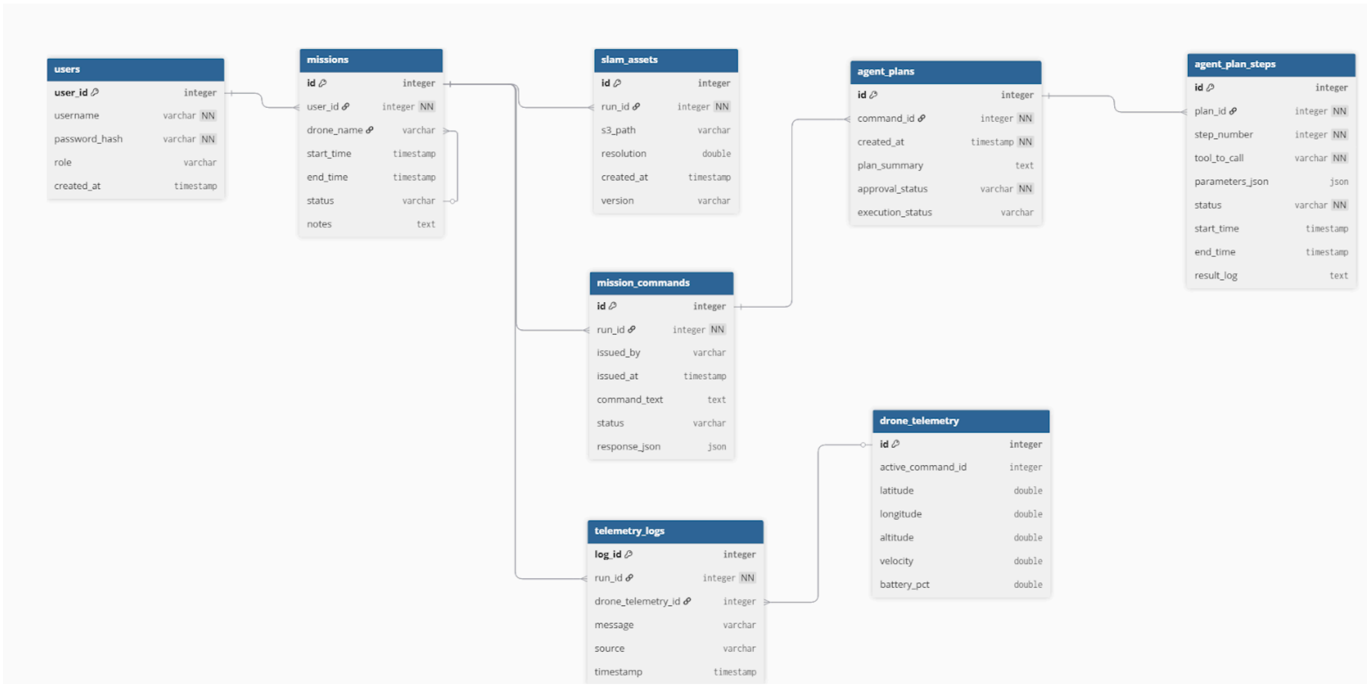


Figure 2.3 – Entity-Relationship Diagram

4.3 Hardware Details

The on-drone hardware consists of all the components required for flight, perception, and communication that are physically mounted on the airframe. Each component was selected to create a lightweight, efficient, and robust platform capable of serving as the mobile half of our GCS-centric architecture.

4.3.1 Drone

4.3.1.1 Holybro X500 V2 Frame Kit

The Holybro X500 V2 Frame Kit serves as the structural backbone of our drone. Its purpose is to provide a rigid and durable chassis that houses all the other onboard components, including the propulsion system, flight controller, companion computer, and sensor payload. Built from high-quality carbon fiber, this frame is designed in a 500mm wheelbase quadcopter ’X’ configuration, a size class that offers an excellent balance between stability for smooth flight and agility for responsive maneuvering. Key features include integrated payload mounting rails and a design that emphasizes both durability and ease of assembly. Its open-layout architecture also facilitates future component upgrades and simplifies access during testing or troubleshooting, which is particularly valuable in a research and prototyping environment.

The selection of the X500 V2 as an Almost-Ready-to-Fly (ARF) kit was a critical strategic decision. The kit comes with the motors and Electronic Speed Controllers (ESCs) pre-installed on the arms, which connect to the main body via convenient plug-in connectors. This significantly reduces the mechanical build time and potential for assembly error, allowing our team to focus its efforts on software development and systems integration rather than on basic fabrication and soldering. The carbon fiber top and bottom plates provide ample surface area for mounting the Pixhawk 6C flight controller, a power distribution board, and our Raspberry Pi 5 companion computer. Furthermore, the included payload rails are essential for securely mounting our primary sensor, the ZED 2i Stereo Camera. This combination of pre-built convenience, modularity, and robust construction made the X500 V2 the ideal, cost-effective foundation for our development platform. Its compatibility with open-source components and industry-standard mounting points ensures that our system can grow in complexity without requiring an entirely new airframe.

4.3.1.2 Propulsion System: Motors & ESCs

The propulsion system is the muscle of the drone, responsible for generating the thrust required for lift and maneuverability. It consists of four brushless motors paired with four Electronic Speed Controllers (ESCs). The motors (Holybro 2216-920KV) are electromechanical devices that spin the propellers at high speeds. The ESCs (Holybro BLHeli-S 20A) act as the interface between the flight controller and the motors, taking the low-power flight commands from the Pixhawk and translating them into precise, high-power signals that control the speed of each motor independently. The ESC works like a translator, converting control signals from the flight controller into the high-power electricity the motors need. Without the ESC, the motors couldn't understand or use the flight controller's instructions.

The primary reason for using these specific components is that they are the manufacturer-matched propulsion system included with the X500 V2 ARF kit. This is a significant advantage, as it eliminates the complex and error-prone process of selecting and pairing individual components. This point was emphasized by professor Rakhshan when asked for advice about purchasing drone parts; do not cheap out, small manufacturer differences can ruin a system. The system is factory-balanced: the 920KV motors are optimized to work with 10-inch propellers and the 20A ESCs when powered by a 4S LiPo battery. This ensures reliable performance and efficient power usage without requiring extensive tuning or guesswork. Each ESC is connected to a main output port on the Pixhawk 6C flight controller to receive its control signal and draws power directly from the drone's power distribution board.

4.3.1.3 Flight Control Unit: Holybro Pixhawk 6C

The Holybro Pixhawk 6C is the central flight control unit for the drone. Think of it as the cerebellum for flight; it's not doing high-level "thinking," but it's responsible for all the complex, real-time calculations that keep the drone stable in the air. Its primary purpose is to run the PX4 Autopilot firmware, which processes data from its onboard sensors (like accelerometers, gyroscopes, and a barometer) to execute flight stabilization and navigation. For our project, its most critical function is receiving high-level commands, like "fly to these coordinates," from our GCS via the MAVLink protocol and translating them into the precise, low-level motor adjustments needed to perform the action.

The Pixhawk 6C was chosen because it's a powerful and reliable industry standard for drone development. It features a high-performance STM32H753 processor and multiple redundant, temperature-controlled IMUs, ensuring robust and trustworthy flight performance. Its native compatibility with the PX4 firmware and MAVLink protocol is the cornerstone of our entire control architecture, enabling our software to programmatically command the drone's movements. The flight controller serves as the main hub on the drone, connecting directly to the propulsion system's ESCs, the GPS module, the RC receiver for manual override, and most importantly, to our Raspberry Pi 5 companion computer via a serial port, creating the bridge for our agentic commands to control the drone.

4.3.1.4 Navigation and Communication Systems: GPS & Telemetry

The Holybro M10 GPS module is the drone's primary sensor for determining its absolute position and heading. Its purpose is twofold: the GPS receiver acquires signals from satellites to calculate the drone's precise latitude and longitude, while the integrated magnetometer (compass) provides heading information relative to magnetic north. This data is absolutely essential for all autonomous navigation functions, including flying to specific GPS waypoints, holding a steady position against wind, and executing critical safety features like Return-to-Home. It also plays a key role in enabling coordinated behavior with other drones by providing a shared geospatial reference frame.

This module was chosen for its high-performance u-blox M10 chipset, known for its rapid satellite acquisition and accuracy. It connects directly to the Pixhawk 6C's dedicated GPS port. Crucially, the module is mounted on an elevated mast, a deliberate choice to position the sensitive magnetometer away from the magnetic interference generated by the drone's power distribution system and motors. This ensures a clean, reliable heading reference, which is vital for accurate navigation. Without a dependable GPS and compass, any form of autonomous outdoor flight would be impossible.

4.3.1.5 Communication Systems: Telemetry Radio & RC Receiver

The Holybro SiK Telemetry Radio set establishes the primary command and control (C2) link for our system. This is not a Wi-Fi connection; it's a dedicated, long-range, and low-bandwidth radio link operating on the 915 MHz frequency. Its specific purpose is to reliably transport MAVLink protocol messages between the GCS and the drone's Pixhawk 6C. This link is how our PilotAgent sends flight commands to the drone and how the drone sends back critical flight data like battery voltage, GPS status, and altitude. By separating this essential C2 link from the high-bandwidth video stream (which uses Wi-Fi), we ensure that flight control is never compromised by a potential video feed interruption.

The RC Receiver is a dedicated safety component. Its sole function is to receive signals from a standard, handheld RC transmitter operated by a human pilot. It connects directly to the RC input port on the Pixhawk 6C, which is configured to give immediate and absolute priority to manual RC commands over any command received from the GCS. Not only will it give priority to RC input, but it will also pause the current execution plan until resumed via the react dashboard. This provides an instantaneous, failsafe manual override. If the autonomous software ever behaves unexpectedly, the pilot can immediately take control, making the RC receiver a non-negotiable part of our safety and testing protocol.

4.3.1.6 Onboard Companion Computer: Raspberry Pi 5

The Raspberry Pi 5 serves as the onboard data hub and the bridge between our high-bandwidth sensor and the GCS. While the Pixhawk handles the immediate tasks of flying, the Pi 5, a powerful single-board computer running a full Linux OS, manages the mission-critical data payload. Its primary purpose is to connect to the ZED 2i Stereo Camera, capture its high-resolution video and depth data streams, and efficiently stream them over Wi-Fi to the GCS for processing by our AI models.

We selected the 8GB RAM model of the Raspberry Pi 5 specifically for this demanding role. Its powerful 64-bit processor and ample memory are essential to handle the immense data throughput from the stereo camera without creating a bottleneck or dropping frames. The Pi 5 connects to the ZED 2i camera via a USB 3.0 port to ensure maximum data transfer speed and communicates with the Pixhawk 6C over a serial connection to exchange telemetry and command data. Its built-in Wi-Fi provides the connection to the GCS network. The flexibility of its Linux environment and its compact, lightweight form factor make it the perfect choice for an onboard companion computer.

4.3.1.7 Primary Perception Sensor: ZED 2i Stereo Camera

The ZED 2i is the primary perception sensor for the LION project, responsible for capturing the rich visual and spatial data that our AI system needs to understand its environment. This advanced stereoscopic camera uses two lenses to mimic human binocular vision, allowing it to compute a real-time depth map alongside a standard color video stream. It provides the RGB video feed that our VLM agent analyzes for object recognition and the depth data that our GCS uses to construct detailed 3D point clouds for mapping and navigation.

The ZED 2i was specifically chosen over other options for several key reasons. First, its global shutter is critical for capturing crisp, distortion-free images from a moving and vibrating platform like a drone. Second, it features a built-in IMU, enabling it to produce a high-quality Visual-Inertial Odometry (VIO) stream. This gives us a very accurate, high-frequency estimate of the drone's motion that is perfectly synchronized with the visual data, which is a massive advantage for performing accurate SLAM. Finally, its robust IP66-rated enclosure, extensive SDK, and excellent ROS (Robot Operating System) integration make it a perfect fit for a serious robotics development project. The camera connects directly to a USB 3.0 port on the Raspberry Pi 5, providing both power and data through a single cable.

4.3.1.8 Onboard Power System: LiPo Battery & UBEC

The main flight battery is an OVONIC 4S 4000mAh LiPo pack. As the primary power source for the entire aerial vehicle, its purpose is to provide the high-current electricity needed to drive the four main motors while also powering all onboard electronics. We selected a 4S (14.8V) battery to match the voltage requirements of our propulsion system, and the 4000mAh capacity offers a good balance between flight endurance and weight. This pack connects directly to the drone's power distribution board, supplying the raw energy for every component.

A small but critical component in the power system is the dedicated 5V UBEC (Universal Battery Elimination Circuit), also known as a buck converter. The main battery’s 14.8V output would instantly damage our sensitive electronics. The UBEC’s job is to tap into that main battery voltage and efficiently "step it down" to a stable, clean 5V output. This regulated 5V supply is used exclusively to power the Raspberry Pi 5 and the attached ZED 2i camera. Using a dedicated UBEC capable of supplying at least 5 Amps is crucial for reliability, as it prevents the Pi 5 from rebooting due to voltage sags when under heavy processing load.

4.3.2 Ground Control Station

The Ground Control Station is the intelligence hub of the LION project. While the drone acts as the mobile eyes and ears, the GCS contains the powerful computing hardware necessary to run the advanced AI models and orchestrate the entire mission. It serves as the central point for command interpretation, task planning, and data processing – allowing the drone to remain lightweight and focused solely on physical execution. By offloading perception and reasoning to the GCS, we unlock more sophisticated capabilities than what could be achieved with onboard hardware alone. This setup also enables real-time interaction with the user through a dashboard, where mission progress, live sensor streams, and agent behavior are continuously monitored and updated.

4.3.2.1 AI Compute Platform: NVIDIA Jetson

The NVIDIA Jetson Orin Nano (8GB) serves as the dedicated AI brain of the operation. This compact, powerful, single-board computer is specifically designed for accelerating AI workloads at the edge. Its primary purpose within the GCS is to receive the high-bandwidth video and depth streams from the drone and perform all computationally expensive tasks. This includes running the Large Language Model for reasoning, the Visual Language Model for visual understanding, generating the 3D point cloud, and executing the main agentic framework. The key to the Jetson platform is its powerful integrated NVIDIA Ampere architecture GPU, which utilizes hundreds of CUDA cores to massively parallelize the calculations required for neural network inference. This allows the system to run large, sophisticated AI models in real-time, a feat that would be impossible on a standard CPU.

The platform's software environment is built upon the NVIDIA JetPack SDK, a comprehensive package that includes the Linux operating system (Linux for Tegra), the CUDA toolkit, and accelerated libraries such as cuDNN. This pre-configured stack is a significant advantage, as it abstracts away the complex process of manually configuring a system for GPU-accelerated computing and provides all the necessary drivers and libraries for AI development. The Jetson was chosen over a standard desktop or laptop because it offers exceptional AI performance in a small, power-efficient package, making the GCS portable for potential field use.

To achieve the lowest possible latency for AI model execution, the project leverages NVIDIA TensorRT, a high-performance deep learning inference optimizer. TensorRT takes the trained VLM and LLM and performs several optimizations to prepare them for the specific Orin Nano GPU. These optimizations include graph optimizations, where network layers are fused together to reduce overhead, and precision calibration, which can convert model weights to use faster, lower-precision formats like INT8 with minimal impact on accuracy. By using TensorRT, the inference speed of the neural networks can be significantly increased, which is a critical requirement for a system that needs to react to its environment in real-time.

For managing the complex software stack and ensuring reproducibility, the system relies on Docker containerization. The entire GCS AI environment, including the specific OS libraries, CUDA version, Python packages, and the agentic framework code, is packaged into a Docker container. This approach solves dependency issues and ensures that the software environment is identical and portable across team members' development machines and the final deployment on the Jetson. NVIDIA provides base container images optimized for the Jetson platform, which simplifies the process of creating a GPU-accelerated container, streamlining the overall development and deployment workflow.

4.3.2.2 Command, Control, and Communication Interfaces

While the Jetson is the brain, a set of crucial interface components connects it to the drone and the human operator.

A TP-Link AC1900 (Archer A8) Wi-Fi Router is used to create a private, local wireless network. Its sole purpose is to handle the high-bandwidth data connection between the drone’s Raspberry

Pi 5 and the Jetson. The router's AC1900 speed rating and MU-MIMO technology are key features, ensuring it can handle the simultaneous video and depth data streams with high throughput and low latency. It has more than enough throughput with 1GB up/down speeds for wired devices and a combined throughput of 1.9GB through 5GHz network. The router only has around 10m of fast speed range. To extend this, we will use an extender like the TP-Link EAP225-Outdoor which will connect to the AC1900 and extend our range up to 200M. Using this dedicated setup gives us the speed/range we need and isolates our critical data stream from any unreliable public networks.

The Holybro SiK Telemetry Radio (GCS Side) is the other half of the telemetry pair. This small radio connects via USB to the GCS and establishes the robust, long-range command and control (C2) link with the drone's Pixhawk. This is the dedicated channel used to send all MAVLink flight commands, ensuring that our ability to control the drone is never dependent on the Wi-Fi video link.

The Radiolink AT9S Pro transmitter is the handheld remote control and our primary safety system. This 10-channel transmitter communicates directly with its paired R9DS receiver on the drone, completely independent of the GCS. In any emergency, the pilot can use the transmitter to instantly override the autonomous system and take manual control, making it an essential, non-negotiable component for safe operation.

4.3.2.3 LiPo Battery Charger

A specialized LiPo Battery Charger is a critical piece of supporting equipment for the GCS. Its purpose is to safely and efficiently recharge the drone's 4S 4000mAh LiPo flight batteries. Unlike standard batteries, LiPos require a charger that can perform cell balancing, ensuring that each of the four cells inside the battery pack is charged to the precise same voltage. This process is essential for maintaining the battery's health, maximizing its lifespan, and ensuring safe operation. Without proper balancing, individual cells can become overcharged or undercharged, increasing the risk of failure or thermal runaway. We selected the Thunder Power TP610HVC for its high reliability, built-in safety features, and compatibility with our specific battery type.

4.4 Software Details

Now we move from the physical components to the intelligence and software that drive them.

4.4.1 The Agentic AI

This section details the software architecture, frameworks, and custom code that transform our collection of hardware into a cohesive, intelligent, and autonomous system. We will break down how commands are understood, how the drone is controlled, how data is managed, and how we test everything safely in a simulated environment.

4.4.1.1 The Code-Generating Agent

The core of our architecture is a code-generating agent paradigm. This approach leverages a Large Language Model not as a planner that simply chooses from a list of tools, but as an autonomous Python programmer. When a user issues a mission, the request is formatted into a detailed system prompt and sent to a smolagents.CodeAgent. This prompt is a critical piece of engineering; it uses a few-shot learning approach by providing the LLM with a perfect example of a mission request and the corresponding, correctly formatted Python script. It also provides strict rules, such as "You MUST write a complete, multi-line Python script," and specifies the exact output format. This precision in prompting constrains the powerful but sometimes unpredictable LLM to reliably produce executable code. Guided by this prompt, our locally-hosted CodeLlama-13b model, running via Ollama, generates a complete Python script to fulfill the user’s request.

4.4.1.2 The Tool Abstraction Layer

The script generated by the agent calls a series of "tools," which are well-documented Python functions that serve as a clean Application Programming Interface (API) for the drone’s capabilities. This creates a vital abstraction layer. The LLM does not need to understand the complexities of the MAVSDK library; it only needs to know it can call a function like return_to_home_and_land(). For example, a tool is defined as a simple function: @tool def

return_to_home_and_land() -> str: ... success = run_async_tool(drone_ctrl.return_to_launch())
....

This is made possible by our DroneInterface class, which is an application of the Facade design pattern. This class encapsulates all the low-level MAVSDK-Python logic. The agent's tools are simple, clean wrappers around methods in this class. This separation is crucial for maintainability and testing. If we ever needed to update the drone control library, we would only need to modify the DroneInterface class, leaving the agent and its tools unchanged. This also allows us to create a mock interface for pure software unit testing of the agentic loop without needing hardware or a simulator.

4.4.1.3 Bridging the Synchronous and Asynchronous Gap

A key technical challenge was integrating our synchronous agent framework with the asynchronous MAVSDK library. Drone control libraries like MAVSDK are fundamentally asynchronous because flight commands are not instantaneous; an action like "fly to location" can take several seconds. A synchronous, blocking call would freeze the entire program while waiting for the drone to physically travel, making it unsuitable for a reactive system. Our solution is a multi-threaded architecture.

On startup, an asyncio event loop is created and run in a dedicated background thread. When the agent calls a synchronous tool, that tool in turn calls the necessary asynchronous method from our DroneInterface. This asynchronous function, or coroutine, is not run directly. Instead, a helper function uses asyncio.run_coroutine_threadsafe() to schedule the coroutine on the background event loop. The main agent thread then waits only for that specific task to return a result. This architecture provides the best of both worlds: a simple, sequential reasoning process for the agent and a high-performance, non-blocking backend for vehicle control.

4.4.1.4 Multi-Agent Coordination for Perception

While the code-generating agent excels at executing planned missions, it cannot easily react to real-time visual information. To enable dynamic, perception-driven tasks like "find the red chair and orbit it," we are implementing a multi-agent architecture managed by LangGraph. This system uses multiple specialized agents that collaborate to solve the problem.

The workflow involves three distinct agents. A VisionAgent is responsible for perception; it takes a frame from the ZED 2i camera and a text prompt, like "Is there a red chair?", and returns structured JSON data with the object's bounding box coordinates. An OrchestratorAgent, which holds the overall mission goal, receives this 2D coordinate data. It then uses the corresponding depth information from the ZED 2i at that location to calculate a precise 3D point in space. Finally, the Orchestrator formulates a new prompt for the PilotAgent, such as "fly to these 3D coordinates and perform an orbit." The PilotAgent is our existing code-generating agent, an expert at executing flight plans.

LangGraph is the framework used to define and manage this collaborative process. It allows us to build the system as a stateful graph, where we can define the agents as nodes and the flow of information between them as edges. LangGraph excels at managing the shared state such as the calculated 3D coordinate across multiple agent calls, ensuring context is not lost. This is essential for achieving the complex, multi-step goals required by the project.

4.4.1.5 Agentic Stretch Goals

Below are some stretch goals that can enhance the functionality and intelligence of our system:

4.4.1.5.1 Semantic Filtering During SLAM

One advanced use of agentic AI would be able to filter out specific types of objects in the point cloud data during scanning, also known as semantic filtering. Below is an example pipeline:

1. An operator can type in the React dashboard, "Scan this field but ignore trees and chairs."
2. The agentic system would interpret this command and pass the relevant filtering targets (trees and chairs) to the VLMAgent.
3. The VLM agent uses object detection to tag objects in the visual feed or point cloud data.
4. The PointCloudProcessor finally filters or removes those objects during the SLAM pipeline, either pre-processing (before the point cloud is built) or post-processing (after the digital twin is complete).

This semantic filtering would allow for customizable mapping, reducing clutter for domain-specific needs and potentially allowing for map layering in the digital twin model.

4.4.1.5.2 Temporal Change Detection

Another stretch goal would be the implementation of a difference algorithm that detects environmental changes across multiple scans of the same area. An example potential pipeline:

1. **First pass**: The drone scans the environment and creates the digital twin, as per our minimum viable product.
2. **Second pass**: The drone repeats a scan of the same area but at either a later date or time.
3. **Comparison**: An AI agent can examine the difference between the point clouds by utilizing object recognition and analyzing spatial alignment.
4. **Alert generation**: The agent can call an alert service to report any significant changes, logging it in the React dashboard.

This difference algorithm would allow for monitoring for unauthorized or undesirable changes (such as the removal of a key item or structural deterioration).

4.4.1.5.3 Context-Aware Task Prioritization

A potential third stretch goal would be the prioritization of scanning certain areas or objects. Instead of performing a typical autonomous scan using an exploration algorithm, the system could give greater weight to investigate a particular event, as in this example:

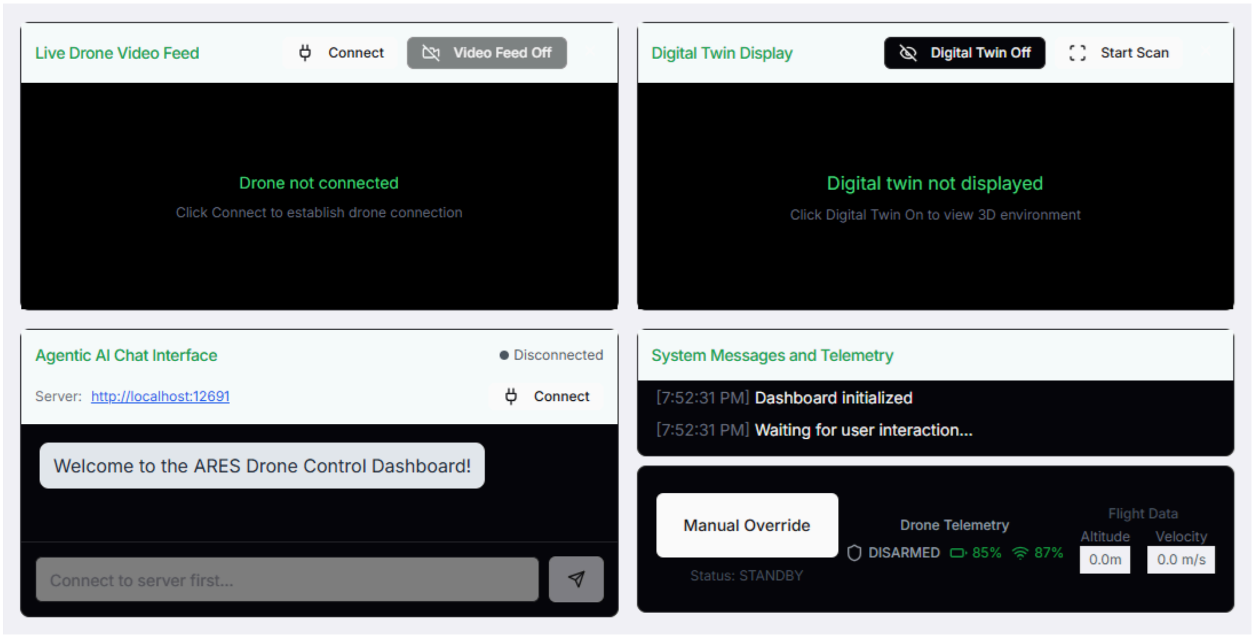
1. The operator sends a command, “Scan this area and identify all construction equipment.”
2. While scanning, the drone encounters an object unrelated to construction equipment.
3. The VLMAgent classifies the object as anomalous, sending a flag to the React dashboard.
4. The scanning is interrupted, and the drone examines the unexpected object in detail.
5. The LLM then reports the results to the operator, upon which the scan will pause and can be resumed based on the operator’s input.

The implementation of context-aware task prioritization would allow the drone to react flexibly to important or time-sensitive events.

4.4.2 React Dashboard

To facilitate real-time human interaction, control, and monitoring of our autonomous drone system, we are developing a custom React-based dashboard. For implementation of the dashboard, we will use Next.js for frontend and backend, as it is a robust React framework that supports server-side rendering and API routing. For styling, we will leverage Tailwind CSS for its utility-first approach. For components, we will utilize Shadcn for pre-built accessibility to accelerate development.

In terms of user interface and design, the dashboard is organized into a four-quadrant layout (shown in *Figure 3.0* below), with each part designed to present critical information or points of interaction in real time to the human operator. This structure not only provides a clear and intuitive display for the user experience but also allows for modularity and future scalability, with each component able to undergo independent development. The goal is to ensure the system remains transparent, responsive, and adaptable across a range of mission complexities, while also enabling rapid debugging, override control, and operator awareness through a cohesive and informative visual interface.



Drone not connected
Click Connect to establish drone connection

Digital twin not displayed
Click Digital Twin On to view 3D environment

● Disconnected
Server: http://localhost:12691

Welcome to the ARES Drone Control Dashboard!

Connect to server first...

```
[7:52:31 PM] Dashboard initialized
[7:52:31 PM] Waiting for user interaction...
```

DISARMED 85% 87%
Altitude 0.0m
Velocity 0.0 m/s
Status: STANDBY

Figure 3.0 – React Dashboard *showing the four-quadrant layout; screenshot taken from a live Next.js development server of the current build.*

4.4.2.1 Live Drone Video Feed

The top-left quadrant of the dashboard will be dedicated to displaying a live video feed from the drone's onboard camera, which will prove essential for visual situational awareness on the part of the human operator either while the agentic AI operates the drone or while the human takes manual control via an override. There will be a connect/disconnect button that will initiate or terminate the connection to the drone, and there will also be a button that can toggle whether or not the video feed is displayed.

The video feed will likely be implemented by a streaming protocol such as MJPEG over HTTP, depending on the final hardware and network configurations. Within the dashboard, the video display component will integrate with Suspense and React hooks to handle the dynamic video stream.

4.4.2.2 Digital Twin Display

The top-right quadrant will be reserved for displaying a three-dimension digital twin of the drone's surrounding environment, which will be dynamically constructed via some select simultaneous localization and mapping (SLAM) algorithm. Our current plan is that the SLAM-created data will be exported as a .ply point cloud data file, which can then be rendered in the React dashboard using Three.js / react-three-fiber. However, our precise file formatting and rendering engine are not yet finalized, as we are still exploring several viable options for implementation.

Like the video feed component, the digital twin component will also include a toggle for turning the display off or on, as well as another toggle for initiating and terminating point-cloud generation.

User interaction features under consideration include camera rotation and zoom, which will be facilitated by either the OrbitControls function from Three.js or a similarly equivalent plug-in integration. The dashboard will be programmed to continuously poll or listen for new model updates, allowing for near real-time updates to the virtual environment as the SLAM algorithm incrementally improves on the digital twin.

4.4.2.3 Agentic AI Chat Interface

The bottom-left quadrant will serve as the primary communication interface between the human operator and the agentic AI system, which will be powered by a localized large language model running in a Python environment. The interface will be designed as a chatbox, allowing natural language commands to be issued to the drone.

Like the previous quadrant components, the AI chatbox will also include a connect/disconnect button for initiating and terminating a connection from the dashboard to the agentic AI. An alternative implementation could potentially invoke some restful API (such as HTTP get/post) to implement the dialogue exchange, which would eliminate the need for establishing a socketed connection.

When the human user enters a command in the chatbox, it will be sent to the agentic AI backend for interpretation. Then, the agentic AI will eventually respond with a stepwise plan of drone actions based on the given command. At this point, the user can review the plan (as the human-in-the-loop mechanism in this system) and then choose an action from one of the following options:

- **Approve the plan**, upon which the drone will execute the corresponding drone movement commands in code,
- **Reject the plan**, upon which the drone will re-attempt to interpret the initial command from the human user, or
- **Cancel the plan**, upon which the chatbox will clear up for the human user to input additional commands.

There will also be a toggle that allows the agentic AI to bypass manual approval, providing support for both supervised and autonomous operation modes.

4.4.2.4 System Messages and Telemetry

The bottom-right quadrant will act as a central console for system logs, alerts, and critical drone data. It will display a real-time message log that captures all interactions across all four parts of the dashboard, including:

- Initialization of the dashboard
- Drone connection and disconnection events
- Video feed toggling and errors
- Digital twin and SLAM model updates
- Initiating or terminating point-cloud generation
- Agentic AI connection and disconnection events
- Chat message delivery and reception
- Execution of each drone command via agentic AI
- Manual override engagement and disengagement
- Warning alerts for boundary or object detection when in manual control
- Significant changes to drone telemetry when in manual control

In addition to this message log, this quadrant will also display live telemetry data from the drone, such as current altitude and velocity. This data is expected to be transmitted via some telemetry API or socketed stream, which the frontend React dashboard will then parse and update accordingly.

Lastly, there will be a manual override button located in this quadrant, which will allow the operator to instantly disable any ongoing autonomous functions and gain full manual control of the drone. This safety mechanism serves as another mode of protection surrounding the agentic AI, providing human oversight over the autonomous system.

4.4.2.5 Sample Use Case

Below is a sample use case for how a human operator might interface with the React dashboard, presuming that all other project components are fully activated:

1. **Initialization**, where the operator will click in sequence:
    - “Connect to Drone” in the top-left quadrant,
    - “Start Live Feed” to activate the video stream,
    - “Display Digital Twin” in the top-right quadrant,
    - “Start Digital Twin Generation” in the top-right quadrant, and
    - “Connect to Agent” in the bottom-left quadrant.

2. **Human-AI Interactivity** in the bottom-left quadrant:
   - The operator types “Find and orbit the nearest chair” and submits the message.
   - The agentic AI interprets the command, responding with a plan: “I will arm the drone and takeoff. I will then autonomously scan the area for a chair. When I find the chair, I will orbit the object. Proceed?”
   - The operator then clicks “Approve the plan”.
3. **Autonomous Execution**:
   - The agentic AI translates the plan into code and executes it in sequence.
   - The drone arms and takes off, then begins scanning its surrounding environment while navigating autonomously.
   - When the drone sees the chair (as interpreted by a visual language model agent), the drone will fly toward it.
   - The drone will begin orbiting the chair autonomously.
4. **Manual Override**:
   - The operator clicks the “Manual Override” button in the bottom-right quadrant.
   - The drone halts in place and continues hovering, awaiting further instructions.
   - The operator can then manipulate the physical drone controller for the desired drone movement.
5. **Mission Completion**:
   - The operator toggles the “Manual Approval Bypass” in the bottom-left chatbox.
   - The operator sends the command “Return to launch” in the bottom-left chatbox.
   - The agentic AI responds with its stepwise plan and immediately executes the required flight path.
   - The drone returns to the starting location and lands safely.
   - The operator exits the program.
   - The dashboard asks the operator whether or not they wish for system message logs and chat history to be saved.
   - The dashboard automatically disconnects all connections and shuts down.

Throughout this entire process, the system message log will update with virtually every button click and message sent or received.

4.4.3 Simulation

To safely and rapidly develop our autonomous control software, we rely heavily on a professional-grade simulation environment. This allows us to test complex flight logic, debug our agentic framework, and iterate on new features without risking the physical drone. Our simulation stack is built on three core, industry-standard robotics tools: Gazebo, ROS 2, and RViz.

Gazebo serves as our high-fidelity 3D robotics simulator. It creates a virtual world complete with realistic physics and, most importantly, provides a simulated model of our Holybro X500 drone. This virtual drone is equipped with simulated sensors, including a stereo camera and an IMU, that mimic the behavior of their real-world counterparts. This gives us a digital twin of our hardware to test against.

ROS 2 (Robot Operating System) is the communication middleware that connects our software to the simulator. Gazebo "publishes" the simulated sensor data onto ROS 2 "topics," and our control software "subscribes" to these topics to receive the data. Likewise, our software "publishes" flight commands, which the simulated drone in Gazebo receives and executes. This allows our DroneInterface to control the virtual drone using the exact same logic it would use for the physical one.

RViz is our 3D visualization tool. While Gazebo runs the simulation, we use RViz to "see" the data flowing through the ROS 2 network. We can visualize the point cloud being generated from the simulated camera, see the drone’s planned trajectory, or view the output of our computer vision algorithms, which is invaluable for debugging and verification. Together, this stack creates a powerful development loop that is essential for the project’s success.

4.4.3.1 On-Drone Data Streaming

The software running on the Raspberry Pi 5 is lean and specialized. Its sole purpose is to act as a high-performance data pipeline, efficiently capturing the complex data from the ZED 2i camera and transmitting it over Wi-Fi to the GCS. This process is split into two distinct streams: one for RGB video and another for depth data.

For the color video feed, we use an industry-standard streaming solution like GStreamer. A GStreamer pipeline running on the Pi 5 takes the raw RGB frames from the camera, encodes them in real-time into the efficient H.264 format, and streams them over the network using the RTSP (Real-Time Streaming Protocol). This method is highly optimized and provides a stable, low-latency video feed that is consumed by both our VLM agent and the operator's dashboard UI.

Streaming the depth data requires a more custom approach. Raw depth maps are too large to stream effectively. Therefore, a custom Python script runs on the Pi 5 that grabs depth frames using the ZED SDK. Each frame is then passed through a high-speed, lossless compression algorithm like LZ4. This dramatically reduces the data size without losing any precision. The compressed depth data is then sent over a simple TCP socket to a dedicated listener on the GCS, where it is decompressed and used to generate the 3D point cloud.

4.4.3.2 GCS Data Processing and Perception

The primary method for perception and object interaction in our system relies on the VLM agent processing the live 2D video stream. For a core mission like "find the red chair and orbit it," the VLM analyzes the RGB feed to identify the object and its location within the 2D camera frame. The system can then use the instantaneous depth data corresponding to that object to estimate distance and perform basic maneuvers relative to it. This 2D-first approach is robust and fulfills the primary requirements of our project.

As a stretch goal, we plan to implement full 3D spatial mapping. This involves a more complex GCS-side process where a Python script would listen for the compressed depth data stream from the drone. It would then decompress each frame and, using the drone's telemetry, project the depth data into a 3D point cloud. By aggregating these point clouds over time, the system would build a persistent and detailed 3D map of the environment, enabling more advanced navigational capabilities. This would not only improve our drone's ability to interpret its surroundings but also support future AI-driven decision making such as obstacle-aware path planning or persistent object tracking. While not required for the MVP, this pipeline is designed with modularity in mind so it can be incrementally introduced and tested as resources permit.

4.4.3.3 Rationale for Selecting Gazebo over Unity

Choosing the right simulation environment was a big decision for the project. A simulator serves as the primary testbed for our entire software stack. It allows for rapid, iterative development and validation in a safe, controlled, and repeatable setting before deploying on the physical drone. After careful evaluation of the leading platforms, we considered both Unity and Gazebo. While both are powerful tools, Gazebo offers a distinct and decisive set of advantages that align directly with our project's core technical requirements, making it the superior choice for our development and testing workflow.

It is important to first acknowledge the significant benefits of using a modern game engine like Unity for robotics simulation. Unity's primary strength lies in its state-of-the-art rendering engine, which is capable of producing photorealistic graphics and visually stunning environments. The high-fidelity visualization capabilities are incredibly valuable for creating compelling digital twins and for any application where aesthetic quality is paramount. Furthermore, the Unity Asset Store provides a vast marketplace of pre-made 3D models, textures, and scripts, which can dramatically accelerate the process of building complex and detailed virtual worlds.

Unity has also made considerable strides in catering to the robotics community with its Unity Robotics Hub. This includes packages for ROS (Robot Operating System) integration, which allow for communication between the simulation and external robotics software. For projects focused heavily on human-computer interaction in visually rich settings or for training vision models on synthetic data that must be as close to photorealism as possible, Unity presents a very compelling case.

However, for a project like LION – where the primary challenges are not visual fidelity but rather the intricate integration of AI, control systems, and realistic sensor physics – the benefits of Unity are outweighed by the foundational strengths of a purpose-built robotics simulator like Gazebo.

Gazebo is not a game engine adapted for robotics; it is a robotics simulator built from the ground up. This fundamental difference in several key areas are critical to the success of our project. Our decision to select Gazebo is based on its superior integration with the robotics ecosystem, its high-fidelity physics modeling, and its alignment with our open-source software stack.

The single most significant advantage of Gazebo is its native, deeply-rooted integration with ROS (and specifically, ROS2), which is a core component of our software architecture. In Gazebo, a simulated drone is not just a 3D model; it is an entity that communicates via standard ROS2 topics, services, and actions. Sensor data from a simulated camera is published directly to a ROS2 topic (e.g., /camera/image_raw, /camera/depth/image_raw), and flight commands can be sent to a ROS2 topic that a flight controller plugin subscribes to.

This native communication model is incredibly powerful. It means our Ground Control Station (GCS), agentic AI, and perception pipeline – all running on ROS2 – can interface with the simulated drone in exactly the same way they interface with the physical drone. There is no need for cumbersome middleware or translation layers like the TCP-based bridges often required for Unity. This 1:1 mapping between simulation and reality dramatically simplifies development, reduces potential points of failure, and ensures that code validated in simulation will work on the hardware with minimal changes.

While Unity excels at visual physics (what looks right), Gazebo excels at dynamic physics (what behaves right). Gazebo offers a choice of multiple high-fidelity physics engines, such as ODE (Open Dynamics Engine) and DART, which are designed to accurately model rigid body dynamics, joint constraints, friction, and aerodynamic forces. This is crucial for simulating the flight behavior of our Holybro X500 quadcopter with a high degree of realism.

Additionally, Gazebo provides a rich library of robotics-specific sensor plugins that model not just the ideal output but also the imperfections of real-world hardware. The simulated ZED 2i camera, for instance, can be configured to produce realistic levels of Gaussian noise, lens distortion, and depth inaccuracies. Simulating these non-ideal conditions is essential for testing the robustness of our perception and SLAM algorithms. Our VLM and point-cloud generation pipeline must be resilient to the noisy, imperfect data it will receive from the real camera, and Gazebo allows us to validate this resilience thoroughly.

Our system architecture relies on the MAVLink protocol to send high-level commands from our agentic AI to the drone's PX4 flight controller. Gazebo offers unparalleled support for a Software-In-The-Loop (SITL) simulation environment. With PX4 SITL, we can run the actual PX4 flight controller firmware in a simulation, which then interfaces with a Gazebo model of the drone.

This workflow is important for de-risking the project. It allows us to test the entire command chain:

1. The agentic AI generates a plan (e.g., "orbit the chair").
2. The plan is translated into a sequence of MAVLink commands.
3. These commands are sent to the PX4 SITL instance.
4. The PX4 firmware executes the commands, controlling the Gazebo drone model.
5. The drone's movement and new sensor data are fed back into our perception pipeline.

Testing this full loop in Unity would require extensive custom development to create a reliable bridge to a PX4 SITL instance. In Gazebo, this capability is a standard, well-documented, and robust feature of the ecosystem, saving us invaluable development time and providing much higher confidence in our control logic.

In summary, while Unity offers best-in-class graphical rendering, its strengths are not aligned with the primary engineering challenges of the LION project. Our goal is to build and validate a robust, intelligent, and autonomous system. This requires a simulation environment that prioritizes:

- Seamless integration with our ROS2-based software architecture.
- Realistic simulation of flight dynamics and sensor imperfections.
- End-to-end validation of the entire control loop, including the PX4 flight controller via MAVLink.

Gazebo excels in all three of these critical areas. It provides a development environment that more closely mirrors our final hardware deployment, allowing us to focus our efforts on the agentic AI and perception systems rather than on building and debugging simulation infrastructure. By choosing Gazebo, we are making a strategic decision to prioritize functional fidelity over visual fidelity, which is the correct trade-off for ensuring the success and robustness of the autonomous drone system.

4.4.3.4 The Simulation-to-Reality Gap

The transition from simulation to the real world, often termed "sim-to-real," is a well-documented challenge in robotics. Simulators, despite their sophistication, cannot perfectly replicate the complexities and unpredictability of a real environment. Several factors contribute to this gap:

- **Imperfect Models**: The mathematical models governing a drone's movement and how its internal systems function are simplified representations of reality.[1]
- **Sensor Discrepancies**: Real-world sensors are susceptible to noise, calibration errors, and environmental factors like changing light conditions, which are often idealized in simulations.
- **Environmental Dynamics**: Factors such as wind, weather, and GPS signal dropouts introduce variability that is difficult to model with complete accuracy in a simulated environment.[2]
- **Hardware Limitations**: The performance of onboard computers and other hardware can be affected by real-world conditions, an aspect that is not always fully captured in simulation.

4.4.3.5 A Staged Transition to Real-World Flight

To safely bridge the gap between simulation and reality, a methodical, multi-stage approach is necessary. This involves progressively introducing real-world components and complexities while validating system performance at each step.[3][4]

A typical staged transition includes:

1. **Pure Simulation**: All initial development and testing of algorithms occur in a completely simulated environment.

2. **Hardware-in-the-Loop (HIL) Simulation**: The actual onboard computer and flight controller are integrated into the simulation. This allows for the validation of the hardware's performance under simulated conditions.[3]
3. **Controlled Real-World Testing**: Initial flights are conducted in a safe, controlled environment, such as an indoor space or an outdoor area with safety nets. This allows for the evaluation of basic flight capabilities and the impact of real-world physics.
4. **Full Real-World Deployment**: Once the system has been thoroughly validated in controlled settings, it can be deployed in the intended operational environment.

4.4.3.6 Safety-Critical Failsafes and Human Oversight

A robust safety architecture is paramount for any autonomous drone system. This architecture should be multi-layered, combining automated failsafe mechanisms with the ability for immediate human intervention.

- **Human-in-the-Loop Control**: A human operator must always have the ability to take manual control of the drone at any moment. This is a critical safety feature that can prevent accidents in unforeseen situations.
- **Geofencing**: The drone's operational area should be strictly defined using a "geofence." If the drone attempts to fly outside this virtual boundary, it should automatically trigger a failsafe, such as returning to its launch point.
- **Loss of Communication Protocol**: The drone must have a pre-programmed response in the event of a lost communication link with the ground control station. This typically involves hovering in place for a set period before initiating a return-to-launch sequence.
- **Collision Avoidance**: A dedicated collision avoidance system should always be active, capable of overriding any other commands if an imminent collision is detected.
- **Pre-flight Checks**: A comprehensive checklist of pre-flight inspections is essential to ensure the drone is in proper working order before every flight. This includes checking battery levels, propeller condition, and GPS signal strength

4.4.4 ROS2

4.4.4.1 Why ROS 2

Our autonomous stack depends on deterministic, low-latency communication between many moving parts: the PX4 flight stack, the ZED 2i sensor suite, SLAM, AI inference, and the React dashboard.

ROS 1's single-master design and TCP-only transport could not guarantee real-time behaviour under bursty video loads. ROS 2 – built on the DDS middleware – gives us:

- **Peer-to-peer discovery** – every node can talk directly to any other without a central broker.
- **Quality-of-Service (QoS) profiles** – we tune reliability and bandwidth per topic (e.g., "best-effort" for 30 FPS RGB frames, "reliable" for pose estimates).
- **Secure DDS** – encrypted data-in-flight for sensitive telemetry

Another important factor for why we choose ROS 2 over ROS 1 is that we wanted this project to be future proof. We have already stated that the goal is for this project to lay the groundwork for many other agentic systems to be built on top of. It may slightly make sense for *us* to use ROS 1 right now, but it makes no sense for someone building an agentic system in 5 years. They should be using ROS 2, so we should use ROS 2.

4.4.4.2 The ROS Architecture

Our architecture leverages a distributed network of ROS2 nodes, with each node responsible for a specific, discrete task. This separation of concerns is critical for building a complex, maintainable system.

There will be many nodes:

- **Camera Node**: A node utilizing the zed_ros2_wrapper package will interface directly with the ZED 2i camera. It will publish synchronized RGB images, depth maps, and camera calibration data to dedicated ROS2 topics (/zed/rgb/image_raw, /zed/depth/depth_registered)

- **MAVLink Bridge Node**: The mavros node acts as a bidirectional bridge between the ROS2 ecosystem and the PX4 flight controller. It subscribes to flight command topics (such as geometry_msgs/Twist for velocity control or /mavros/setpoint_position/local for position control) that are published from the GCS. Simultaneously, it publishes the drone's telemetry data, such as battery status, altitude, and IMU readings, received from the PX4 back onto ROS2 topics for consumption by the GCS
- **Data Ingestion & Perception Nodes**: Subscriber nodes on the GCS will receive the raw sensor data streams from the drone over the wireless network. This data is then fed into our perception pipeline
  - **SLAM node** will subscribe to the image and depth topics to generate a real-time point cloud of the environment, publishing the resulting map to a /point_cloud topic
  - **React dashboard** will subscribe to RGB data so it can display a live feed to the user
  - **Exploration Node** will subscribe to the output of the SLAM node so that it can perform a next best position algorithm
- **Agentic AI Node**: The core Agentic AI system operates as a ROS2 node that subscribes to high-level processed data, such as the environmental point cloud and the drone's state telemetry. Based on the user's natural language command and its internal world model, the agent formulates a plan and publishes the corresponding flight control commands to the appropriate mavros topics, which are then relayed back to the drone for execution

This ROS2-based architecture provides several key advantages. First, modularity allows individual components like perception, control, and AI reasoning to be developed and tested independently. This simplifies debugging and allows for easy upgrades without impacting the entire system. Second, interoperability through standardized ROS2 message types allows our system to easily integrate with existing robotics tools, like Gazebo for drone simulation and RVIZ for diagnostics. Finally, the framework is inherently scalable. It makes the addition of new sensors or computational nodes extremely easy as the project's complexity grows.

4.4.4.3 SLAM

Simultaneous Localization and Mapping (SLAM) is the cornerstone of our drone's autonomous navigation capability. It is the process by which the drone builds a map of an unknown environment while simultaneously keeping track of its own position within that map. The choice of a SLAM algorithm is critical, as its accuracy, robustness, and computational efficiency directly impact the entire system's performance. After evaluating several leading SLAM solutions, we have selected RTAB-Map (Real-Time Appearance-Based Mapping) as the most suitable framework for the LION project.

Several powerful open-source SLAM algorithms are available within the ROS ecosystem, including ORB-SLAM3, Cartographer, and RTAB-Map. Each has its strengths but RTAB-Map offers a unique combination of features that aligns perfectly with our project's specific requirements.

- **ORB-SLAM3** is renowned for its high accuracy in localization, but it is primarily a visual SLAM system focused on camera tracking and sparse feature maps. It is less suited for generating the dense, detailed point clouds needed for our digital twin and environmental analysis.
- **Google Cartographer** is excellent for creating highly accurate 2D and 3D maps and excels with LiDAR data. However, its performance with RGB-D cameras can be less robust, and its loop closure mechanism is not as central to its design as it is in RTAB-Map.

RTAB-Map stands out for its robust feature set, tight ROS 2 integration, and its specific design for long-term, large-scale mapping, making it the ideal choice for our system.

- **Graph-Based SLAM with Strong Loop Closure**: The core strength of RTAB-Map is its graph-based architecture. As the drone explores, it creates a graph of keyframes (poses) connected by spatial constraints. The most critical feature is its sophisticated loop closure detection. When the drone re-observes a previously visited location, RTAB-Map recognizes it based on visual appearance, adds a "loop closure" constraint to the graph, and then re-optimizes the entire map to correct for accumulated drift. This is absolutely essential for creating globally consistent and accurate maps, especially during long missions or exploration of complex, multi-room environments.

- **Excellent Sensor Fusion and RGB-D Support**: RTAB-Map is designed to work with a variety of sensors, with particularly strong support for RGB-D cameras like our ZED 2i. It effectively fuses the visual information (RGB) with depth data to build dense 3D maps of the environment. This versatility also provides a clear path for future expansion, as it can seamlessly integrate other sensor types like IMU and LiDAR if we choose to upgrade our hardware.
- **Native ROS 2 Integration and Community Support**: RTAB-Map is a first-class citizen in the ROS 2 ecosystem. It provides a comprehensive set of well-maintained ROS 2 packages that expose all necessary topics, services, and parameters for easy integration. This tight integration significantly reduces development and debugging time, allowing us to connect it directly to our navigation stack (nav2), visualization tools, and custom agentic nodes without compatibility headaches. The active development and strong community provide invaluable resources for troubleshooting and advanced implementation.
- **Dual 3D and 2D Map Generation**: A key advantage for our architecture is RTAB-Map's ability to simultaneously produce two critical outputs from a single data stream:
    - **A dense 3D point cloud**, which is essential for our GCS-based digital twin, providing rich visual and spatial context for the human operator.
    - **A projected 2D occupancy grid (costmap)**, which is the required input for ROS 2 navigation and exploration packages like nav2_exploration.
    - This dual-output capability allows us to serve both the high-level visualization needs of the user and the low-level navigation requirements of the drone's autonomous planner without running separate, resource-intensive mapping processes.

All in all, RTAB-Map provides the best balance of performance, features, and practicality for the LION project. Its powerful loop closure ensures map accuracy, its native support for our ZED 2i camera simplifies data processing, and its seamless integration into the ROS 2 framework makes it the most efficient and robust choice for our SLAM pipeline.

4.4.4.4 Exploration Algorithm Selection

To achieve true autonomy, the drone must be able to independently explore and map an unknown environment without predefined waypoints. This capability is driven by an exploration algorithm, a critical software component that analyzes the current map, identifies unknown areas, and strategically decides where to move next to gather new information efficiently. The chosen algorithm must be compatible with our ROS 2 framework, computationally feasible for our hardware, and capable of guiding an aerial vehicle in three-dimensional space.

Exploration strategies generally fall into a few key categories:

- **Frontier-Based Exploration**: This is the most common and well-understood approach. The algorithm identifies frontiers, which are the boundaries between mapped, free space and unobserved, unknown space. The robot then selects a frontier and navigates toward it to expand the known map. This method is robust and computationally efficient but can lead to inefficient paths as the robot may travel long distances between frontiers.
- **Next-Best-View (NBV) Planning**: Primarily used for 3D reconstruction, NBV algorithms select the next camera pose that is expected to reveal the most new information about the environment's surface. This is ideal for creating detailed 3D models but is often computationally expensive, as it requires evaluating the potential information gain from many candidate viewpoints.
- **Receding Horizon and Sampling-Based Planners**: These methods, often using techniques like the Rapidly-exploring Random Tree (RRT), focus on finding a safe path into a nearby unknown region. They are excellent for navigating complex and cluttered 3D environments in real-time but may not explore the environment as systematically as frontier-based methods.

For our system, the primary choice is between established 2D-centric methods (nav2_exploration, explore_lite) that are well-integrated into ROS 2 and more advanced 3D-native planners designed specifically for drones. We are not going to use 2D-centric methods because drones are inherently 3D. Projecting the point cloud data into 2D space would be cutting ourselves short and would significantly limit the ability of the drone to explore 3D environments.

Instead, we will be using a 3D planner like Fast-Planner or EGO-Planner. These planners are natively 3D, using volumetric data structures like OctoMaps to represent the world. They integrate exploration directly into their motion planning, generating dynamically feasible, smooth trajectories that are safe for the drone to execute. The primary drawback is complexity. Integrating and tuning these advanced planners is significantly more involved than using the standard nav2 packages. They also have higher computational requirements, which will need to be carefully managed between the onboard Raspberry Pi 5 and the GCS.

We will test out different 3D planners using Gazebo simulation until we settle on one that performs the best, in simulations and in real life.

4.4.5 Coding Principles And Standards

To manage the complexity of our large distributed codebase, we are adopting a set of coding principles and standards. These practices are essential for building a robust, scalable, and maintainable codebase, preventing the divergence of our core software components: the on-drone systems, the Ground Control Station (GCS) backend, and the React frontend. Our approach is centered on decoupling, ensuring that each part of the system can be developed, tested, and updated independently without causing cascading failures.

- **Modular, Interface-Driven Design**: A core practice is the use of abstract interfaces to define the contracts between different software modules. For example, much of the logic for controlling the drone will be duplicated between the real-world drone and a potential future simulation. To handle this, we will define an interface, such as IFlightController, which includes methods like ExecuteMove(targetPosition) and GetTelemetry(). This interface can then be implemented by a RealDroneController (which translates calls into MAVLink commands) and a SimulatedDroneController (which interacts with a simulation engine). By writing the agentic AI to interact with the IFlightController interface, the AI remains agnostic to whether it is controlling a physical drone or a digital twin. This approach dramatically improves testability and flexibility
- **Centralized Shared Libraries**: To prevent code drift and ensure consistency across the project, we will maintain a shared library for common data structures, interfaces, and

utilities. This library will serve as the single source of truth for critical code artifacts, including:

  - **ROS2 Message Definitions**: Standardized formats for streaming sensor data (e.g., camera imagery, depth data, IMU readings) and control commands.
  - **Data Transfer Objects (DTOs)**: The data structures used in the API between the React dashboard and the GCS backend, ensuring both frontend and backend agree on how information is exchanged.
  - **Core Data Structures**: Definitions for agentic plans, states, and tool schemas used within the LangChain/LangGraph framework.
  - **Utility Functions**: Reusable functions for tasks like coordinate transformations, data parsing, and validation.

- **Standardized APIs and Communication Protocols**: Our system's components operate as independent services that communicate through well-defined APIs. These standardized protocols ensure that as long as a component adheres to the established contract, its internal implementation can be modified or replaced without affecting the rest of the system. The key communication protocols are:
  - **MAVLink**: The protocol governing communication between the GCS and the drone's PX4 flight controller for command and control.
  - **ROS2**: The internal messaging backbone of the GCS, used for routing high-throughput sensor data from the drone to various processing and AI nodes.
  - **REST/WebSocket API**: The interface between the React dashboard frontend and the GCS backend, enabling user interaction and data visualization.

Environment Consistency with Containerization: To eliminate the "it works on my machine" problem and streamline deployment, we are all using the same version of ubuntu and the same version of software we use (like gazebo and px4 sdk). We are also all using a python venv.

By adhering to these principles, we are building a decoupled architecture where components are interchangeable and independently maintainable. This strategy not only mitigates complexity during development but also lays a solid foundation for future extensions, such as integrating new sensors, supporting multi-drone operations, or adapting the system to new robotic platforms.

4.5 Testing Plan

4.5.1 Prototyping Individual Components

The first post-research phase of our project will focus on the creation and validation of the numerous individual components that comprise our overall autonomous drone system. In consideration of hardware requirements, this includes the running of a PX4-Gazebo simulated model of our Holybro drone, to ensure it can meet the various necessary criteria for our ultimate deliverable (our minimum viable product). In addition, this includes the validation of our many physical components that will be a part of our drone and ground control station system, including but not limited to:

- The Holybro Pixhawk 6C flight controller
- The Zed 2i stereo camera
- The battery and corresponding charger for the drone and its on-board components
- The Raspberry Pi for the drone
- The Jetson Orin Nano for the ground control station
- The radio receiver and transmitter for manual override to control drone movement
- The TP-Link wireless router

In parallel with the hardware efforts, software prototyping will involve the development of early versions of the core modules that will power our system. One of the relevant tasks will be the experimentation with a number of photogrammetry tools, particularly our simultaneous localization and mapping (SLAM) algorithms to transform sample image data and even potentially video feed data into three-dimensional point clouds. Another portion will be the development of our real-time monitoring dashboard built with React, which will visualize system outputs such as live video feed, digital twin rendering, agentic AI dialogue, and drone telemetry, with the primary goal of ensuring that the frontend design and user interactivity are ready and in place for future backend integration with other modules. An additional requirement will be the installation of localized large language models via Ollama, testing out different types and different parameter counts for both speed and efficacy. A fourth aspect will be the implementation of a simple prototype of an AI agent that can interpret human commands and output MAVSDK Python code for drone movement, utilizing the SmolAgents framework for tool calling. Finally, the last part will be the evaluation of different storage solutions for our backend, which will need to house image data, video data, point cloud data, and other project-critical data.

4.5.2 Building Component Interactivity

Once we have validated each of our major components individually, we will proceed into the next phase of our project, where the focus will shift toward integrating these subsystems toward a functional and interactive pipeline. On the hardware side, we will need to build our drone together with its number of on-board components and test its operation once built, while ensuring that the individual tests from the previous phase remain valid. We will also need to validate data flow between the drone and the ground control station, including streaming protocols, file format conversions, and methods of encryption.

On the software side, we will need to ensure that our SLAM algorithms can operate on the incoming feed from the drone and can output the digital twin point cloud appropriately. Regarding the React dashboard, we will need to build backend integration and APIs for linking the existing components with the corresponding drone video feed, the point cloud generated by the SLAM module, the SmolAgents agentic framework, and drone telemetry derived from sensor data. We will need to migrate our agentic AI to operate on the Jetson Orin Nano, along with any other requisite software components. Through all of these, our backend solution will need to work effectively as we integrate it accordingly.

4.5.3 End-to-End Testing and Validation

Following successful unit and integration tests from our previous phases, we will proceed to our final phase of end-to-end testing and validation. At this point, we will commence with live drone tests involving all hardware and software components, making sure that our agentic AI, our React Dashboard, our SLAM module, our backend, and the drone itself work together in concert. By this point, our system should be able to execute autonomous missions with a human in the loop, as per the specifications of our minimum viable product.

5. Conclusions

5.1 Results

The project's result demonstrated the core workflow of translating a natural language command into a physical drone action. The system's agentic AI workflow proved effective for coordinating perception and action. The VisionAgent, utilizing the VLM, was able to identify specified objects within a live 2D video feed. A key result was the system's ability to correlate this 2D visual data with depth information to derive a 3D target point for the drone to navigate towards.

The PilotAgent demonstrated its capability to generate valid MAVSDK flight scripts in real-time based on the dynamic inputs provided by the orchestrating agent. The physical drone, receiving these commands from the GCS via the MAVLink protocol, executed the commanded maneuvers, such as orbiting a designated point. This validated the end-to-end functionality of the control loop, from the initial user command to the final drone movement.

5.2 Summary

The LION project was undertaken to develop an autonomous drone system controlled by natural language. The system was designed using a GCS-centric architecture, where the drone itself functions as a mobile sensor platform, while all intensive AI computation is offloaded to a ground control station. This approach was chosen to maximize the system's AI capabilities without being limited by the size, weight, and power constraints of the drone.

The final system consists of a Holybro X500 V2 drone equipped with a ZED 2i stereo camera and a Raspberry Pi 5 for data streaming. This airframe communicates with a Ground Control Station powered by an NVIDIA Jetson, which runs the multi-agent AI framework. The software architecture uses a VisionAgent for perception, a code-generating PilotAgent for action, and an OrchestratorAgent to manage the mission flow. The project serves as a proof-of-concept for integrating these complex hardware and software components to create a more intuitive and interactive autonomous system.

6. Administration

6.1 Budget and Financing

The LION Autonomous Drone project was allocated a total budget of $2,000. Our procurement strategy focused on maximizing the technical capabilities of the platform while maintaining cost-effectiveness. The total planned expenditure is $1,560.16 (+ shipping added later), keeping us comfortably under budget as detailed in our Purchases List (*Figure 4.0*). This was achieved through a combination of online purchasing for specific components and by securing critical, high-value hardware like the NVIDIA Jetson, Raspberry Pi 5, and a professional-grade battery charger through UCF labs and faculty sponsorship. This approach allowed us to allocate a significant portion of our funds toward the core drone kit and the advanced stereo camera, which are the most critical hardware for the project's success.

The following table details the project's expenditures:

Item	Price	Amount	Source
Holybro X500 V2 PX4 Dev Kit	$719.00	1	Online
OVONIC 4S LiPo Battery 4000mAh 14.8V 130C RC (2 pack)	$74.88	1	Online
UBEC Converter: ZORZA 2Pcs UBEC 5V 5A	$9.29	1	Online
ZED 2i Stereo Camera	$520.00	1	Online
Thunder Power RC TP610HVC	$0.00	1	Lab
Raspberry Pi 5	$0.00	1	Lab
Jetson Orin Nano (8GB)	$0.00	1	Faculty
Radiolink AT9S Pro 10/12 Channels Radio Transmitter and Receiver R9DS, Long Range for Airplane/Jet/FPV Racing Drone/Quad/RC Truck Car/Boat and More (Mode 2 Left Hand)	$117.00	1	Online
TP-Link AC1900 Smart WiFi Router (Archer A8) -High Speed MU-MIMO Wireless Router, Dual Band Router for Wireless Internet, Gigabit, Supports Guest WiFi	$50.00	1	Online

TP-Link EAP225-Outdoor | Omada AC1200 Wireless Gigabit Outdoor Access Point | Business WiFi Solution w/ Mesh Support, Seamless Roaming & MU-MIMO | PoE Powered | SDN Integrated | Cloud Access & App	$69.99	1	Online
Totals	$1,560.16	10	

Figure 4.0 – Purchasing List Table From Purchase Google Sheet

Spending Allocation

The allocation of our spent budget is: Drone Kit & Power: 54%, Perception System: 35%, Control & Communication: 11%

6.2 Jira Artifacts

6.2.1 Product Backlog

The broader product backlog includes several development tasks that have been scoped and prioritized but are scheduled for later sprints. These items span various stages of the project's lifecycle, from drone assembly and backend setup to AI-driven mission execution. For instance, the team plans to assemble the drone platform (LAD-34) once all parts are received and validated. Simultaneously, efforts to start the backend repository (LAD-37) will begin to support dashboard data ingestion and AI output logging.

Hardware configuration also remains an ongoing priority, with tasks like setting up the Raspberry Pi and other compute devices (LAD-50) and checking out equipment from the lab (LAD-60) to ensure test-readiness. On the AI side, the backlog includes defining multi-step agentic missions in LangGraph (LAD-70) and further refining drone-specific tooling tasks (LAD-75) to support swarm behavior and task handoffs. Additionally, a GCS documentation task (LAD-57) will outline the data handling, rendering, and visualization workflows that bridge ROS output with point cloud processing and frontend rendering.

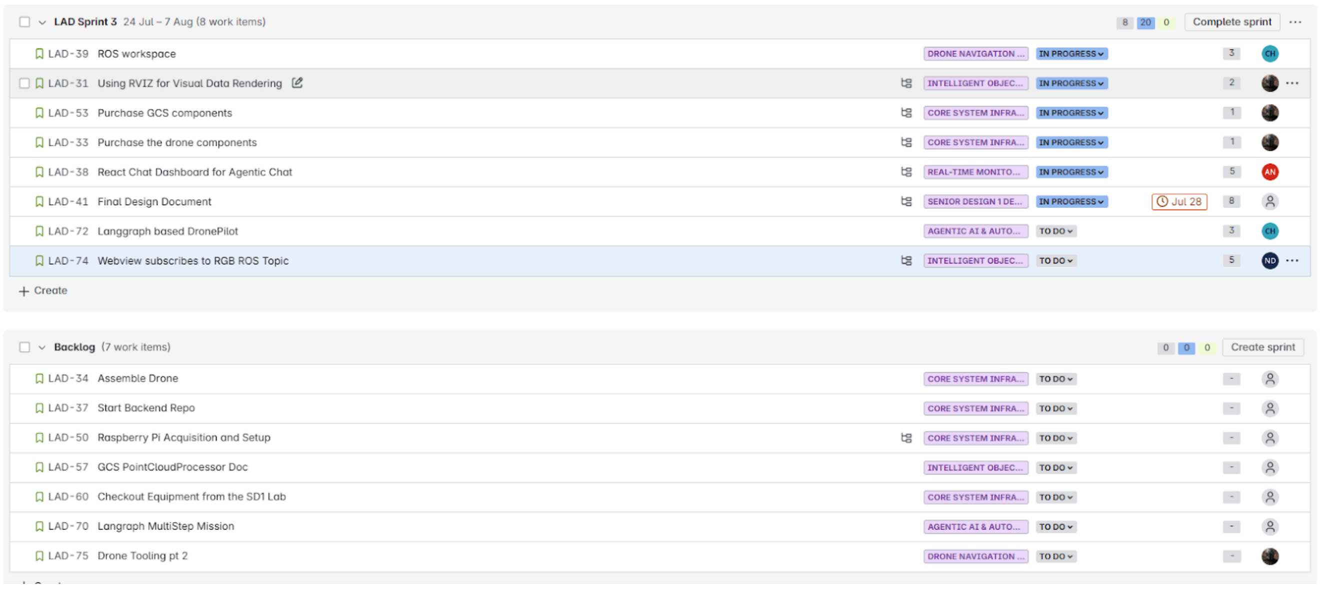
This backlog complements the current sprint by queuing up development efforts that depend on the outcomes of foundational tasks like hardware setup and system integration. Once Sprint 3 concludes, many of these stories will transition into active work, supporting our progress toward a functional MVP.

6.2.2 Sprint Backlog

The focus of LAD Sprint 3 (July 24 – August 7) is to progress from foundational planning into early implementation, particularly with a strong emphasis on both hardware procurement and the integration of agentic software components. Several key tasks are being tackled across domains such as visualization, AI workflows, and system setup. For instance, team members are configuring a functional ROS workspace (LAD-39) to support inter-process communication and hardware control. Simultaneously, the team is testing RVIZ (LAD-31) to evaluate its viability for live visual data rendering within our digital twin pipeline. Although ultimately deprioritized in favor of a web-based streaming approach, the RVIZ investigation helps inform our visualization trade-offs.

On the hardware side, components for both the drone platform (LAD-33) and the ground control station (LAD-53) are being purchased and documented to ensure readiness for system assembly in the next sprint. In parallel, efforts are underway to prototype a React-based GCS dashboard (LAD-38), including a chat interface for issuing instructions to an underlying LangGraph-based AI. This AI interface is represented by LAD-72, which seeks to implement an autonomous planning pipeline that integrates with the drone's control layer. Additionally, LAD-74 focuses on integrating video streaming by subscribing a WebView to the RGB ROS topic, helping pave the way for web-based live feeds via MJPEG. The final design document (LAD-41) is also being wrapped up in this sprint to reflect current architectural decisions and research summaries. While some tasks remain in the "To Do" phase, the sprint overall marks a significant shift from research into tangible development.

A snapshot of our Jira board showing our current sprint and backlog is shown on the next page (see *Figure 5.0*).


LAD Sprint 3 24 Jul – 7 Aug (8 work items) 8 20 0 Complete sprint

LAD-39	ROS workspace	DRONE NAVIGATION ...	IN PROGRESS		3	CH
LAD-31	Using RVIZ for Visual Data Rendering	INTELLIGENT OBJEC...	IN PROGRESS		2	
LAD-53	Purchase GCS components	CORE SYSTEM INFRA...	IN PROGRESS		1	
LAD-33	Purchase the drone components	CORE SYSTEM INFRA...	IN PROGRESS		1	
LAD-38	React Chat Dashboard for Agentic Chat	REAL-TIME MONITO...	IN PROGRESS		5	AN
LAD-41	Final Design Document	SENIOR DESIGN 1 DE...	IN PROGRESS	Jul 28	8	
LAD-72	Langgraph based DronePilot	AGENTIC AI & AUTO...	TO DO		3	CH
LAD-74	Webview subscribes to RGB ROS Topic	INTELLIGENT OBJEC...	TO DO		5	ND

+ Create

Backlog (7 work items) 0 0 0 Create sprint

LAD-34	Assemble Drone	CORE SYSTEM INFRA...	TO DO	-
LAD-37	Start Backend Repo	CORE SYSTEM INFRA...	TO DO	-
LAD-50	Raspberry Pi Acquisition and Setup	CORE SYSTEM INFRA...	TO DO	-
LAD-57	GCS PointCloudProcessor Doc	INTELLIGENT OBJEC...	TO DO	-
LAD-60	Checkout Equipment from the SD1 Lab	CORE SYSTEM INFRA...	TO DO	-
LAD-70	Langraph MultiStep Mission	AGENTIC AI & AUTO...	TO DO	-
LAD-75	Drone Tooling pt 2	DRONE NAVIGATION ...	TO DO	-

Figure 5.0 – Jira Board *showing our current sprint and our backlog*

6.2.3 Milestone Chart

The development process for our autonomous drone system was structured around a series of monthly milestones tied to Agile sprints. These milestones (see *Figure 5.1*) reflect major technical and integration achievements and were used to track progress toward our Minimum Viable Product (MVP). Each milestone correlates with specific Jira Epics and Stories, ensuring alignment between development goals and project planning tools. Early milestones focused on system architecture design, platform evaluation, and research into key enabling technologies such as SLAM, photogrammetry, and LangChain-based agentic workflows. As the project advanced, milestones became more implementation-driven, including tasks like integrating the flight controller with the companion computer, streaming live sensor data, and testing natural language mission execution.

Sprint retrospectives were conducted at the end of each phase to evaluate what worked well, what needed improvement, and how best to adjust our workflow moving forward. This iterative approach allowed us to remain responsive to technical challenges, reallocate resources effectively, and quickly prototype new solutions when blockers emerged. Notably, our use of Jira allowed us to break down large milestones into manageable deliverables, assign tasks clearly, and monitor progress across subsystems in parallel. The result was a structured yet flexible development cycle that emphasized continuous progress while remaining adaptable to the evolving needs of the project.

Month	Milestone	Key Deliverables
June	Foundational Research & System Architecture	- Defined MVP scope and system goals - Researched drone hardware platforms, sensors, and onboard compute options - Evaluated agentic AI frameworks (LangChain/LangGraph, SmolAI) - Assessed Unity, photogrammetry tools (COLMAP, Meshroom), and point cloud workflows - Drafted initial system diagrams and integration roadmap
July	Hardware Selection & Early Software Prototypes	- Finalized drone platform and supporting sensors - Developed MJPEG video stream using OpenCV + Flask - Prototyped React-based dashboard - Researched RViz integration and documented limitations - Set up photogrammetry testing pipeline and point cloud processing workflow in Unity

Figure 5.1 – Milestone Chart

7. Acknowledgments

We would like to express our deepest gratitude to the many individuals and organizations who made the LION Autonomous Drone project possible.

7.1 Sponsors

This project was generously sponsored by Serco, whose support enabled our team to explore advanced drone autonomy and AI integration in a real-world engineering context. We especially thank Maddy Braganca, Brian Babukovic, Blake Bergstrom, and Chris Griffith for their invaluable input, encouragement, and for taking the time to meet with our team throughout the semester. Their technical insights and willingness to provide feedback helped shape the direction of our development and kept the project grounded in practical, impactful goals.

7.2 Coordinators

We are also grateful to our academic coordinators: Dr. Matthew Gerber, Dr. Richard Leinecker, and Dr. Mohsen Rakshan. Their consistent guidance, feedback on our deliverables, and willingness to support both technical and administrative aspects of the project were critical to our progress. Their mentorship helped us balance research ambitions with practical design execution.

7.3 Facilities and Equipment

We thank Dr. Richard Leinecker for providing the NVIDIA Jetson platform, which enabled us to develop and test AI models locally. We are also grateful to Serco for providing a dedicated meeting space for collaboration and technical review sessions. Additionally, we acknowledge the Software Development 1 (SD1) and Mechanical Engineering (MAE) Lab at the University of Central Florida, which served as our primary workspace and gave us access to critical hardware including a Thunder Power RC TP610HVC charger and a Raspberry Pi 5, both of which were instrumental in powering and integrating our drone components during testing.