Project ARES: Final Design Document

Group L06

LION Autonomous Drone

Team Members

Rayyan Jamil

Abraham Ng

Nathaniel D'Alfonso

Connor Hallman

Sponsors

Madalyn Braganca

Christopher Griffith

Serco

IDesign Coordinators

Matthew Gerber

Richard Leinecker

COP4935, Fall 2025

11/24/2025

Table of Contents

1. Executive Summary

The ARES Autonomous Drone project represents an advanced robotics initiative designed to fundamentally bridge the gap between high-level human intent and complex robotic action. The goal is to develop an intelligent system that can execute complex missions given in natural language. This objective is powered by an AI architecture that includes a YOLOv8n vision model for real-time perception, a Large Language Model (LLM) for reasoning, and a LangGraph AI agent to orchestrate these intelligent components.

Our GCS-centric architecture offloads intensive AI agent workflows from the drone to a ground control station (GCS). This approach keeps the drone lightweight, agile, and power-efficient. This division of labor also allows us to maximize AI performance by leveraging GCS computing power.

The physical implementation consists of a S550 Hexacopter equipped with a ZED 2i stereo camera and a Jetson Orin Nano companion computer. The Jetson's CUDA support is critical for handling the streaming of RGB video and depth data to the GCS. Within the GCS, the agentic AI processes the live sensor data, creates an actionable plan, and translates it into precise flight instructions. These commands are then transmitted to the drone's flight controller via the MAVLink protocol, enabling fully autonomous flight and mission completion.

The project's Minimum Viable Product (MVP) will demonstrate this workflow with key objectives including real-time 3D point cloud generation, object identification and localization, and the successful execution of natural language commands such as "fly five feet up, then find and orbit the red chair." Ultimately, LION-ARES provides an extensible foundation for human-robot interaction research and a proven framework for real-world autonomous AI deployment.

2. Introduction

2.1 Project Significance

Creating an agentic drone system is a significant achievement and milestone in the world of autonomous robots, expanding the utility of drones to a realm of use cases previously thought impossible. Although the world already has limited autonomous systems such as Roombas, self-driving cars, warehouse robots, and other specialized systems that perform niche activities, these systems lack true agency. They do not have broader decision-making capabilities nor goal-driven initiatives. Agentic autonomy is more than merely executing preprogrammed tasks; it requires the ability to comprehend dynamic environments, generate and evaluate plans, adapt to changes in real time, use the appropriate tools, and pursue objectives without constant human oversight. Our framework pioneers the enablement of agentic intelligence with drones.

We believe our project can make an impact in three ways. First, drone designers can use our project to expedite development of agentic drones. Second, our framework can be expanded to higher-level systems like multi-drone teams and role-based behaviour. Third, our framework can be extended to other types of vehicles, such as land vehicles, aquatic surface vehicles, space rovers, and more. Our project lays the base foundation for drones to operate agentically, with code that is both modular and open source. Our successors only need to define new tools for their drones or edit the implementation to fit their use case.

Drones are already becoming commonplace in many industries such as weddings, real estate, and agriculture. However, current drones require significant attention from a human operator. By removing this handicap, our project can make drones cheaper to operate, more accessible to more users, and more practical for many industries.

2.2 Project Motivation

This section features individual member explanations regarding project motivations.

2.2.1 Rayyan Jamil

For me, the LION Autonomous Drone System project represented a multidisciplinary challenge, positioned at the intersection of my interests in robotics, information technology, and software engineering. I was motivated by the complex technical roadblocks inherent to creating intelligent cooperation between an autonomous aerial platform and its human operator, and I was inspired by the challenge of designing an AI system able to adapt and make robust decisions in dynamic, real-world scenarios.

A key draw of this project was 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, provided a link between abstract AI concepts and their practical application in an airborne system.

Furthermore, this project was distinguished by its strict reliance on live, real-world data. Unlike many AI development workflows using simulated or pre-recorded datasets, this project's commitment to operating exclusively on live sensor streams presented greater authenticity and difficulty. The system had to process and react to unpredictable and often imperfect data from real-world operation, forcing a more robust design.

Finally, the project's heavy focus on AI was a significant motivating factor. Allowing Large Language Models (LLMs), agentic workflows, and Visual Language Models (VLMs) to operate autonomously in the physical world with tangible consequences was an invaluable opportunity, in contrast to controlled environments. This project provided 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 was fourfold. Foremost, I aimed to deepen my understanding of agentic AI, including the large language models and machine learning underpinning it. My prior background working with AI was limited to calling an AI model via API from a project frontend, along with using AI chatbots. Through this project, I hoped to improve my ability to interact with and integrate large language models as well as to better understand their technical foundations and applications.

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

Third, I wanted 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 anticipated that being able to work on the React dashboard for this project would both utilize my previous background with React on the frontend and reinforce my backend integration skills in writing API calls.

Lastly, I hoped 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 was critical to increasing my productivity and my understanding of the typical workflow and environment for software developers. Additionally, the local job market had a high concentration of defense contractors and related companies, and I anticipated that this project would be instrumental in helping me land my first job after graduation.

2.2.3 Nathan D’Alfonso

My motivation for joining this project stemmed from a long-standing interest in AI, autonomous systems, and robotics. When I first learned about the project opportunity, 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 aligned closely with my interests in multi-agent systems, machine learning, and embedded hardware design.

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

I believed 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 was not just a technical goal but a human-centered one, especially in scenarios where autonomous behavior required trust.

Additionally, I appreciated the creative freedom that Serco gave us in shaping the system’s design. This flexibility allowed us to explore innovative approaches and iterate on ideas instead of following rigid client specifications. Working with a multidisciplinary team of peers made this a uniquely collaborative and fulfilling experience.

Overall, this project gave 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 was excited by the breadth and depth of the project. I saw 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 this project stemmed from a long-held desire to work with physical systems. While I had focused on web development throughout my computer career, the idea of writing code that translated into tangible, real-world action was something I had always wanted to do. Working with hardware presented a unique and exciting set of challenges that I had previously been intimidated by, and this project offered the perfect opportunity to overcome that. I was also excited by the agentic part of this project. Being able to architect a system that reasons and acts autonomously was a priceless opportunity, as AI has seized the forefront of computer science and nearly every aspect of modern-day life. I was excited to contribute to a field with profound implications for researchers and everyday people.

This project had challenges that would push me out of my comfort zone, accelerating my growth and learning. I anticipated pushing my capabilities, being forced to grapple with unfamiliar concepts in robotics, aerial systems, and agentic AI. Also, learning Python had been on my to-do list. Importantly, this experience would fundamentally expand my problem-solving toolkit, creating a new mental framework of solutions and strategies that would allow me to tackle related technical concepts in the future.

Beyond the academic and technical benefits, I was excited to apply my existing skills. A polished and intuitive user experience would be critical to the project's success, and I was eager to leverage my web development experience to build the React dashboard for our project. Developing my skills in autonomous systems, AI integration, and robotics would open doors to exciting job opportunities in drone systems, agentic AI development, and with defense contractors, aligning well with my professional goals.

Last but not least, I appreciated Serco for the creative freedom they gave us, as well as the University of Central Florida and my teammates for the opportunity to collaborate on this project toward more learning opportunities and real-world applicability.

2.3 Broader Societal Impacts

The LION Autonomous Drone project has the potential to generate wide-ranging societal benefits while also raising important questions about safety, security, and responsible development. Our system supports accessibility by enabling users to control aerial platforms using natural language, lowering barriers for individuals with mobility impairments, motor limitations, or limited technical backgrounds. Traditional UAV interfaces require fine motor control and manual piloting experience; by contrast, a voice-driven or text-driven command interface dramatically broadens who can effectively operate advanced robotics. This technological democratization improves the inclusivity of STEM fields and expands opportunities for underrepresented groups to participate in research and field operations.

From an environmental standpoint, autonomous drones reduce the need for intrusive, fuel-consuming survey methods. The ability to perform non-invasive aerial scans minimizes disturbance to fragile ecosystems, reduces human travel into environmentally sensitive zones, and generates richer digital twin datasets for environmental monitoring. Researchers can track erosion, vegetation changes, pollution spread, and wildlife impacts with greater precision and lower ecological footprint.

In education and workforce development, the project provides a model for integrating modern robotics with AI-driven autonomy. Because our system is built using widely accessible technologies, such as React, Python, ROS 2, Jetson hardware, and open-source SLAM tools, students and research teams can reproduce our techniques without relying on proprietary ecosystems. This fosters a stronger pipeline of developers with hands-on experience in human-in-the-loop autonomy, multi-agent systems, and real-time spatial computing.

The project also intersects with national security and public safety. Autonomous systems capable of mapping disaster areas, locating survivors, and providing rapid situational awareness can enhance emergency response readiness. However, their capabilities also introduce important security concerns. Our system collects spatial, visual, and potentially sensitive environmental data, meaning that strong controls are required to prevent unauthorized access, tampering, or data interception. During development, we implemented encrypted wireless communication, operated in isolated test sites, and enforced strict control over sensor data. Long-term scaling will require continued adherence to best practices in cybersecurity, authentication, and encrypted telemetry.

Safety and ethical concerns also require attention. Autonomous systems can act unexpectedly when given ambiguous or poorly structured commands. To mitigate this, we incorporated a strict human-in-the-loop architecture, with manual override capability, real-time telemetry visualization, and bounded test environments to prevent uncontrolled behavior. As autonomy increases in future iterations, such as multi-drone coordination or semi-autonomous swarm behaviors, these risks will need proportional safety strategies, including standardized geofencing, agent explainability, and real-time anomaly detection.

In summary, the broader societal impacts of our autonomous drone project include but are not limited to improvements in accessibility, environmental stewardship, STEM education, emergency response capability, and autonomous system safety. These benefits coexist with new ethical and security challenges, which our design directly addresses through privacy-focused data handling, controlled autonomy, and explicit human oversight.

2.4 Legal, Ethical, and Privacy Issues

2.4.1 Test Flight Location

At one point, we had considered doing test flights on the grounds of the University of Central Florida. However, operating on campus was not a facile endeavor, with a myriad of administrative formalities and procedures we would have to fulfill and follow. Additionally, these issues would prohibitively set back our project timeline. As a result, we decided to pursue drone operations elsewhere, particularly in public parks that permitted drone flight without having to deal with excessive and cumbersome bureaucratic red tape.

2.4.2 FAA Regulations

Our primary legal concern subsequently shifted from compliance with the regulations of the University of Central Florida to general compliance with the regulations of the Federal Aviation Administration. All outdoor drone use at the time of our project fell under FAA jurisdiction, including private spaces as well as publicly accessible parks, which were the areas we were most likely to operate our drone test flights. Specifically, any unmanned aircraft that weighed more than 0.55 pounds would need to be registered accordingly, which our drone clearly will exceed. In addition, our drone operator(s) would need to pass the requisite FAA tests to be appropriately certified. To fulfill these legal obligations, we ensured that all registrations and certifications necessary were completed before we conducted any of our live drone tests, with one of our members obtaining the requisite licensure.

2.4.3 Agentic Autonomy and Human Oversight

A key ethical consideration in the deployment of our system was 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 included a structured human-in-the-loop design, with the human operator being granted live video feed, agentic AI plans, and drone telemetry to keep up-to-date in their knowledge of the current state of the system in operation. In addition, the human operator was able to engage in a manual override at any point in time to take full control of the drone and immediately block agentic AI control and terminate any flight plans in progress. Lastly, our project scope limited the drone to specific test environments that constituted bounded enclosures with limited obstacles and uniform terrain, as we wanted to avoid operating the drone in overly complicated scenarios while we worked on troubleshooting our agentic AI models, our ROS topics, and our PX4 / MAVSDK commands. These safety measures that we took during our live drone deployment sessions reduced the likelihood of unpredictable behavior and ensured that any critical decisions would remain under human oversight and discretion. Our project's approach thus mitigated common concerns about the trustworthiness of autonomous behavior and demonstrated a commitment to responsible agentic AI usage.

2.4.4 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 significantly lowered this risk by implementing a variety of operational protocols. Foremost, we selected to conduct our drone tests in areas isolated from heavier human congestion and at times of the day conducive to limited human activity, reducing the likelihood of encountering unanticipated foot traffic. Furthermore, we attempted to clearly signal the area and zone of our drone operations by the physical placement of our team members, in order to deter other people from entering the testing site. Finally, we ensured all captured data would be used strictly for point cloud generation sent to our React dashboard,

and we ensured that all identifiable information captured by camera that we used for footage was either removed or rendered unintelligible. Together, these precautions strengthened the privacy standards of our project.

2.4.5 Data Security

Given the potentially sensitive nature of the spatial and visual data that our drone system could collect as well as the human and agentic AI interactions that our React dashboard would facilitate, data security was a critical area of concern. Without the proper safeguards, our project's data could have been intercepted or exploited, whether through our local wireless communication protocols or through global communication to outside large language models. To address these issues, we elected to adopt a privacy-centric architecture for our project.

All wireless communication between the drone and the ground control station was encrypted using industry-standard protocols to secure all transferred data. Additionally, the limited range of our network alongside operating in an open field with high visibility also meant that intruders would likely be physically seen before they could be in range to effectively infiltrate or read our wireless communications.

We also tried utilizing locally hosted large language models through Ollama. We had hoped to avoid the use of online models, which could collect user commands or contextual data during our drone missions, and we had also hoped to circumvent the possibility of data being intercepted in transit between our system and online models. Unfortunately, due to issues with latency and response quality, we needed to move from local Ollama models to the online Gemini 2.5 Flash-Lite model.

Nevertheless, our technical safeguards assisted in verifying that all data from the drone, from the human operator, and from the large language models would remain secure and compliant with best practices regarding data security.

3. Project Characterization

3.1 Objectives and Goals

The primary objective of the LION Autonomous Drone project is to create an intelligent system capable of transforming natural-language instructions into safe, reliable, and autonomous drone actions. The system is designed to bridge the gap between human intent and robotic execution by integrating an agentic AI reasoning pipeline, real-time sensor processing, and a ground control station (GCS)-centric architecture. This enables operators to issue high-level commands such as “scan the field” or “orbit the red object”, while the system automatically interprets the request, generates a flight plan, executes the mission, and provides transparent feedback for each step of the mission.

A core goal of the project is to demonstrate a complete, end-to-end autonomous mission flow with minimal manual intervention. This includes interpreting the natural-language command, parsing it into discrete tasks, autonomously generating and executing a flight plan, performing real-time perception using onboard sensors, and returning actionable results to the dashboard. Achieving this flow validates the connectivity and coordination between the language model, the agentic reasoning system, the Jetson companion computer, and the PX4 flight controller. The system must execute these actions predictably, safely, and with full human-in-the-loop visibility.

Another major objective is to ensure that the system is modular, extensible, and adaptable to future mission needs. The architecture is intentionally designed to support the addition of new sensors, new agentic capabilities, and even multiple drones without requiring significant redesign. This modularity allows future developers to

expand the system toward advanced behaviors, such as swarm coordination, cooperative mapping, semantic understanding, or long-duration mission autonomy. Likewise, the GCS-centric approach ensures that the system remains flexible as more powerful language models, perception tools, or mapping pipelines become readily available.

A further objective is the integration of real-time spatial understanding through SLAM-based and photogrammetry-informed workflows. The system must be capable of streaming live RGB and depth data, generating a digital twin of the scanned environment, and providing the operator with clear situational awareness. The dashboard must present this data intuitively, including three-dimensional point cloud reconstructions, mission logs, and system telemetry. To support future post-mission analysis, the system needs to incorporate a snapshot feature that will allow operators to store digital twin point cloud reconstructions locally for review, comparison, or re-processing.

Finally, safety, transparency, and human oversight remain foundational goals. Every autonomous action must be interpretable, interruptible, and visible to the operator. Manual override capability, real-time telemetry, agent reasoning summaries, and bounded test-environment constraints are requirements to ensure that the system behaves responsibly and predictably. These principles form the backbone of the project and ensure that the LION Autonomous Drone system supports future research, operational deployments, and continued evolution toward advanced autonomous capabilities.

3.2 Specifications and Requirements

To support the objectives of the LION Autonomous Drone System, we established a detailed set of functional, software, system, and user-driven requirements that define the capabilities necessary for reliable autonomous operation. These requirements ensure that the system can interpret human intent, process real-time environmental data, execute flight missions safely, and remain extensible for future enhancements such as multi-drone coordination and advanced perception workflows.

From a functional standpoint, the system must enable the operator to issue natural-language commands and receive a correct, autonomous response from the drone. The agentic AI must interpret the command, decompose it into actionable tasks, generate the corresponding flight plan, and monitor mission progress while adapting to sensor updates or environmental changes. The agent must also manage internal memory, reason over multiple steps, and request clarification when a command is ambiguous or unsafe. These capabilities ensure that autonomy remains predictable, consistent, and aligned with operator intent.

To support these behaviors, several core software components form the foundation of the system. The React Dashboard functions as the central user interface, providing real-time video streams, 3D point-cloud visualization, system telemetry, mission logs, and a natural-language command console. It also supports the newly implemented Snapshot feature, which allows operators to save point-cloud reconstructions directly to the device for later review. The agentic AI layer, built using LangChain and LangGraph, is responsible for interpreting user instructions, selecting appropriate tools, generating flight plans, and coordinating with perception modules and the flight controller. The system further incorporates localized large language models (LLMs), enabling secure, offline operation where mission data, sensor streams, and user

prompts remain fully contained within the ground control station without relying on external networks.

Several system requirements also govern safe and effective deployment. Human-in-the-loop oversight is mandatory, ensuring operators can intervene at any time through manual override and maintain full visibility into the drone's decisions and telemetry. The architecture must treat the drone as a distinct autonomous agent, broadcasting its state, position, and sensor data independently to the GCS. This separation supports future scalability for multi-drone missions. The system must also support real-time data integration, continuously ingesting RGB video, stereo depth, GPS, IMU, SLAM outputs, and point-cloud updates to support accurate mapping, flight adaptation, and dashboard visualization.

To validate that the system's requirements align with real-world operational scenarios, a collection of user stories was defined. These user stories capture the expected interactions, constraints, and mission needs that operators may encounter. In the finalized system, the following user stories apply:

- As a search and rescue operator, I want the drone to autonomously locate people in a hazardous environment so I can identify survivors without entering unsafe areas.
- As an engineer, I want the drone to generate a 3D map of a construction site so I can remotely monitor structural progress.
- As a technician, I want to override the drone at any moment so I can ensure safety during field testing or unexpected conditions.
- As an operator, I want to save point-cloud snapshots during or after a mission so I can review the spatial reconstruction and compare it with future scans.
- As a drone coordinator, I want the system to ask for clarification when given an ambiguous instruction so it does not take unsafe or unintended actions.

- As a security officer, I want all model inference and data processing to occur locally so sensitive environmental data never leaves the system.
- As a disaster-response analyst, I want the drone to highlight unusual objects or anomalies so I can quickly identify environmental damage or changes.
- As a researcher, I want the drone to operate fully offline so missions can be run in remote locations without internet connectivity.

These user stories ensure that the final system is intuitive, safe, mission-relevant, and aligned with real operational expectations across engineering, environmental, and emergency-response domains. Collectively, the specifications and requirements form a robust foundation for the LION Autonomous Drone System and provide a clear roadmap for future scalability, additional agents, and more advanced forms of autonomous behavior.

3.3 Concept of Operations

The LION Autonomous Drone System operates under a centralized architecture in which the ground control station (GCS) performs all major reasoning, planning, and perception tasks, while the drone serves as the physical agent executing those decisions. This GCS-centric approach allows the system to leverage more powerful computing resources on the ground, maintain a secure and controlled operational environment, and support future scalability to multiple autonomous agents without increasing onboard computational load.

The operational workflow begins when the human operator submits a natural-language command through the React dashboard. This instruction is forwarded to the agentic AI, which interprets the operator’s intent, decomposes the request into structured tasks, and generates a corresponding mission plan. The agent then converts these tasks into

MAVLink-compatible commands, which are executed through the PX4 flight controller using low-latency communication.

Throughout the mission, the drone streams RGB video, stereo depth data, odometry, and telemetry back to the GCS. These data streams support real-time SLAM-based mapping, visual situational awareness, digital twin construction, and autonomous plan refinement. The React dashboard displays synchronized video feeds, point-cloud reconstructions, agent reasoning summaries, and live position tracking, ensuring that human operators remain fully informed about system actions and mission progress at all times.

A critical element of this concept of operations is the requirement for human-in-the-loop oversight. Operators retain permanent override authority through the dashboard or RC controller. At any moment, the operator may interrupt the agent, halt autonomous behavior, and assume manual control. This safeguard ensures responsible use of autonomy, especially during early development stages or when operating in unpredictable environments.

The system also incorporates a new snapshot pipeline, which allows operators to save point-cloud reconstructions at any stage during the mission. When activated, the GCS packages the current reconstruction, associated metadata, and digital twin state into a compact snapshot object stored in IndexedDB. This enables later review, comparison, and offline analysis without requiring the drone, network connectivity, or re-execution of the mission. The snapshot feature supports long-term research workflows, mission auditing, and incremental mapping of environments.

Overall, our project emphasizes safety, transparency, adaptability, and extensibility. This hybrid human-AI operational model ensures effective field deployment while maintaining predictable system behavior, clear operator oversight, and seamless integration of perception, planning, and control.

4. Research

4.1 Project Proposals

Before we settled on our final candidate for how our minimum viable product would be expressed, we decided that each team member should individually brainstorm a variety of project proposals. Next, we gathered together to examine them and evaluate them in a group setting as a joint effort. We then took several days to individually ponder and vote on our top candidates. Finally, we met together again and collectively narrowed down the options until settling on our final proposal. In addition, we consulted with our sponsor to determine if and how our proposals would meet the desired requirements and specifications expected of us.

All of our initial project proposals are documented in the following sections. In particular, the unselected project proposals represent potential applications of our project for any future teams that want to build upon our project by taking our efforts into different directions or by working toward unique applications of our product.

4.1.1 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.

4.1.2 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.

4.1.3 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.

4.1.4 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.

4.1.5 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.

4.1.6 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.

4.1.7 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.

4.1.8 Rationale for Final Proposal Selection

In narrowing down our proposal candidates, we evaluated them according to the following 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.

4.2 Hardware Research

Following the definition of the ground control station (GCS)-centric architecture, the selection of an appropriate physical platform was a paramount design decision. The system required a reliable, programmable airframe capable of serving as a mechanical host for the GCS “brain”, rather than a proprietary, closed ecosystem. This selection process involved a rigorous, multi-stage evaluation of mechanical stability, propulsion efficiency, payload capacity, and procurement feasibility. Despite unanticipated setbacks during the selection and purchasing process, we were eventually able to successfully acquire a workable and constructable drone adequate for our project.

4.2.1 Evaluation of Drone Platforms

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 but not limited to: 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. While these commercial drones offer superior mechanical polish, their locked software development kits (SDKs) and closed API ecosystems presented insurmountable barriers to the custom agentic control required by the project.

Finally, fully custom builds based on hobbyist frames were deemed too resource-intensive regarding PID tuning and hardware calibration.

4.2.2 Camera Selection: The ZED 2i Stereo Camera

Our team conducted an analysis of sensor options for real-time spatial data acquisition. We settled on selecting 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 airframes under consideration.

4.2.3 Initial Platform Selection: Holybro X500 V2

The initial phase of research identified the Holybro X500 V2 PX4 Development Kit as the technical “gold standard” for the project. The X500 V2 was particularly attractive due to its widespread adoption in the academic community and its carbon-fiber construction, which offers a high stiffness-to-weight ratio necessary for stable flight; the platform represented the modern state-of-the-art in research robotics at the time of our project.

One primary advantage of the X500 V2 was its streamlined “Almost-Ready-to-Fly” (ARF) assembly process. The kit would have arrived with the motors and electronic speed controllers (ESCs) pre-installed, significantly reducing the most tedious and error-prone aspects of constructing a drone. No soldering or welding would have been required to build the drone. In addition, the X500’s mechanical design includes integrated payload rails and a standardized mounting interface, making it technically ideal for hosting the project’s sensor suite, including the ZED 2i stereo camera and the companion computer.

The 500mm wheelbase placed the drone in a size class 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 would have provided 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 would have provided a secure, standardized mounting point for the ZED 2i camera, ensuring it has a clear, forward-facing view and is isolated from vibrations.

Originally, the selection of the Holybro X500 drone constituted a critical strategic decision that would save dozens of hours of assembly and significantly reduce the chances of errors or setbacks. The kit's pre-installed propulsion system and wiring harness would have significantly mitigated the risks associated with manual mechanical assembly, allowing our team to allocate resources immediately toward software integration rather than mechanical troubleshooting. The ease of construction would have facilitated our team spending most of our focus and time on the focal innovations of the project: the software, AI integration, and control systems.

Furthermore, the selection of this kit was fundamentally driven by its open-source core. The platform was built around the Pixhawk 6C flight controller running the PX4 Autopilot firmware. This open-source stack would allow for access to the drone's control systems via the MAVLink protocol. This well-documented, standardized communication protocol was planned as the essential bridge to permit our GCS-based agent to send high-level programmatic commands to the drone. This open-source protocol stood 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.

Lastly, the modular design of the X500 platform supported future sensor integration and experimentation, in the event of our project evolving to require additional payloads.

4.2.4 Procurement Constraints

Despite the technical superiority of the Holybro X500 V2, the project encountered significant barriers regarding acquisition. During the procurement phase, the Holybro X500 V2 experienced severe global supply chain shortages, resulting in extended lead times and unpredictable availability. Relying on a platform with such volatile stock levels posed a critical risk to the project's timeline, potentially delaying the integration and testing phases.

In addition, we discovered that the process of procurement through the University of Central Florida was unclear and lengthy, with multiple points of contact redirecting us to other people and with approvals being delayed for reasons unknown to the team. Compounded with the aforementioned supply and availability issues of the Holybro X500 V2, we were forced to examine and pursue alternative selections for our drone platform.

Simultaneously, a budgetary analysis revealed that the high unit cost of the X500 V2 would consume a disproportionate amount of the allocated funds, leaving us minimal flexibility in the event of unanticipated equipment purchases in the event of an emergency.

4.2.5 Final Platform Selection: S550 Hexacopter

Consequently, the team conducted a comparative market analysis and pivoted to the S550 Hexacopter platform. While the S550 utilizes an older generation of propulsion technology compared to the X500 V2, resulting in slightly lower electrical efficiency, it presented a compelling solution to both the availability and cost challenges. The platform was widely available from multiple domestic suppliers, eliminating supply chain risks.

In addition, the S550 hexacopter was acquired at approximately half the cost formerly allotted to the acquisition of the Holybro X500 V2.

Despite the older motor design, the S550's hexacopter configuration provided six points of thrust compared to the X500's four. This redundancy enhanced flight stability and provided a substantial lifting envelope that was able to accommodate the combined weight of the Jetson Orin Nano, the ZED 2i camera, and the necessary voltage regulation hardware.

A critical requirement for the platform change was that it must not compromise the software architecture designed during the research phase. Fortunately, both the X500 and the selected S550 platform support the open-standard Pixhawk flight controllers and the PX4 Autopilot firmware. This commonality was vital, as it ensured that the control logic, based on the open MAVLink protocol, remained platform-agnostic and compatible with our software approach and development.

By utilizing the S550 frame with a standard Pixhawk 2.4.8 flight controller, the team was able to achieve an optimal balance of modularity, affordability, and logistical reliability. This approach ultimately facilitated the seamless deployment of the GCS-centric agent and ROS-based sensor fusion without exceeding the project's financial limitations or succumbing to supply chain delays. The final hardware design represents a strategic engineering compromise, prioritizing accessible hardware and open software standards to ensure the timely delivery of advanced autonomous capabilities.

4.3 Agentic and General AI Research

One of the critical goals of our project was 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 was to be carried out by an intelligent agent utilizing a variety of tool calls, enabling the agent to interact dynamically with its surrounding environment via the drone as well as with the human-in-the-loop operator. Our ultimate aim was to permit operators to issue simple, intuitive commands and having the agents respond and execute them in real time.

To accomplish this, we conducted research into the following software components:

4.3.1 Model Context Protocol

The Model Context Protocol (MCP) was 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. This protocol subsequently saw rapid widespread adoption across the AI ecosystem in the months following its release, as it enabled 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 but not limited to:

- **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
- **Toolkits and algorithms** that may be commonly called by agents

One potential application of the MCP to our agentic system would have been to have our agentic AI act as an MCP client, permitting it to make arbitrary tool calls to an MCP server to access additional data. However, we eventually designated MCP integration as a goal for future iterations and development of our project.

4.3.2 Ollama and Localizing Models

In alignment with our aims of data localization and security, we planned to operate our large language models locally instead of accessing them online. We thus looked into Ollama, a platform optimized for the deployment of lightweight language models in local environments. Using Ollama would have come with the following benefits:

- **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 large language models
- **Low-latency inference**, with real-time responsiveness even on limited hardware
- **Lower costs**, by avoiding having to run API calls for proprietary LLMs

In our research into localized models, we encountered several drawbacks when compared to calling APIs for a typical industry-leading online large language model, such as slower tokenization and generation, along with weaker output with lower parameters. To combat these issues, we tried quantization, an optimization technique that reduces the precision of the model's numerical weights (such as converting from 16-bit floating-point numbers to 4-bit integers), which drastically reduced model file size and memory footprint and led to faster inference speeds on resource-constrained devices with a negligible trade-off in response quality. We also attempted to tweak our tools, prompting, and failsafe checks in our agentic AI software loop. Lastly, we investigated a variety of Ollama models, as detailed in the following section.

Nonetheless, we were unable to achieve sufficient latency and response quality via Ollama, necessitating that we pivot to industry-standard online large language models.

4.3.3 Large Language Model Research

Our research into large language models (LLMs) initially focused on identifying models that would be suitable for running locally via the Ollama platform. During our analyses, we sought to find a balance between reasoning capability, performance, and resource consumption.

We investigated several prominent open-source models, including but not limited to:

- **Mistral Models** (such as Mistral 7B), highly capable general-purpose models known for their excellent performance-to-size ratio, well-suited for understanding user intent and general-purpose planning
- **CodeLlama Models** (such as CodeLlama 13B), specifically fine-tuned by Meta for code generation, making them an ideal choice for our primary CodeAgent paradigm where the model's primary task would be to write correct Python scripts to execute missions

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.

Unfortunately, these Ollama models proved to be inadequate for the latency and response quality we needed for live drone operation via agentic AI, and so toward the latter stages of our project, we instead used **Gemini 2.5 Flash-Lite**, a cost-effective online large language model from Google optimized for low latency. We found that Gemini 2.5 Flash-Lite provided quality output quickly enough for agentic drone execution, with only minimal cost overhead due to the relative brevity of our prompts.

4.3.4 SmolAgents Agentic Framework

The final piece of the software puzzle was the agentic framework, the software that would orchestrate the large language model and its tools. After evaluating several options, we initially 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 offered a vast number of features, their complexity would introduce significant overhead and a steeper learning curve. We found that SmolAgents was facile for rapid deployment, critical for our early stages of software development and prototyping.

The rationale for first selecting SmolAgents was twofold:

- **Simplicity and Clarity**: The framework had 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 proved invaluable for rapid prototyping and debugging in a research context in our initial stages of software development for agentic AI.
- **Lightweight Performance**: Its minimal overhead ensured that the framework itself consumes very few resources, which was critical when running the agentic AI concurrently with other software components of our project (such as Gazebo).

This choice allowed 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 on the following page (see *Figure 4.1* on the following page).

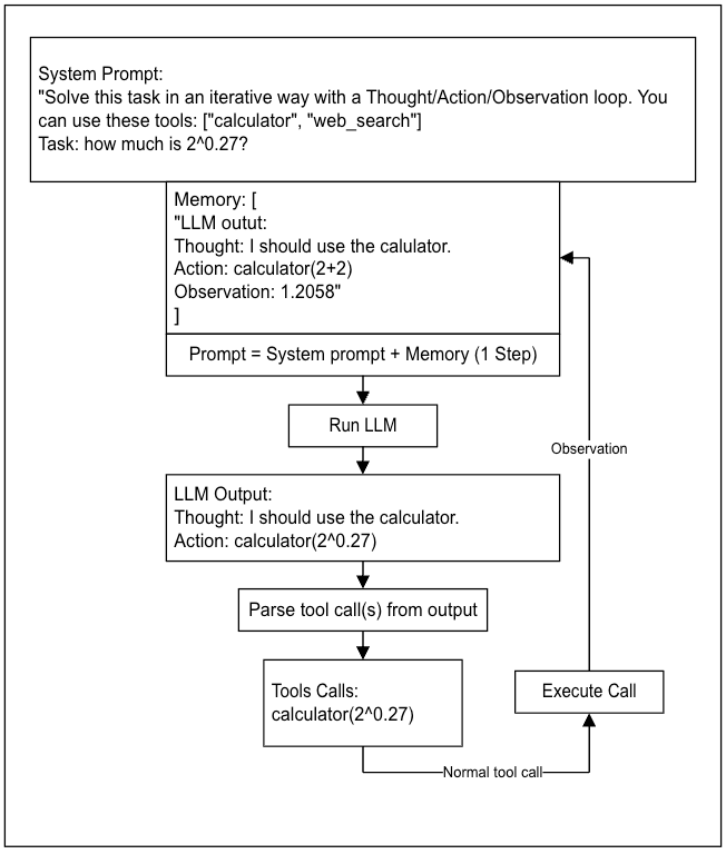

***Figure 4.1 – SmolAgents Diagram**: General Workflow Concept*

4.3.5 LangChain and LangGraph

After working with SmolAgents in the early stages of software development, we decided to migrate over to LangChain and Langraph. In our research, we found that LangGraph was significantly more flexible than SmolAgents, especially when it came to routing execution. SmolAgents functioned like a black box, in that we gave it agent tools and hoped for the desired result; on the other hand, LangGraph gave us fine control to analyze each step of the process and debug accordingly, allowing us to block commands, reprompt the agent, and cancel missions.

In our investigative stage, we also found that LangChain and LangGraph would grant us the ability to establish a controlled flow involving multiple agents, and both would be

compatible with the Model Context Protocol, further pushing us to transition out of SmolAgents, although our final product ended up not being multi-agent nor did it invoke the Model Context Protocol.

During our initial stage of working with LangChain and LangGraph, we set out to implement the following pipeline for our agentic system (see *Figure 4.2* below):

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.

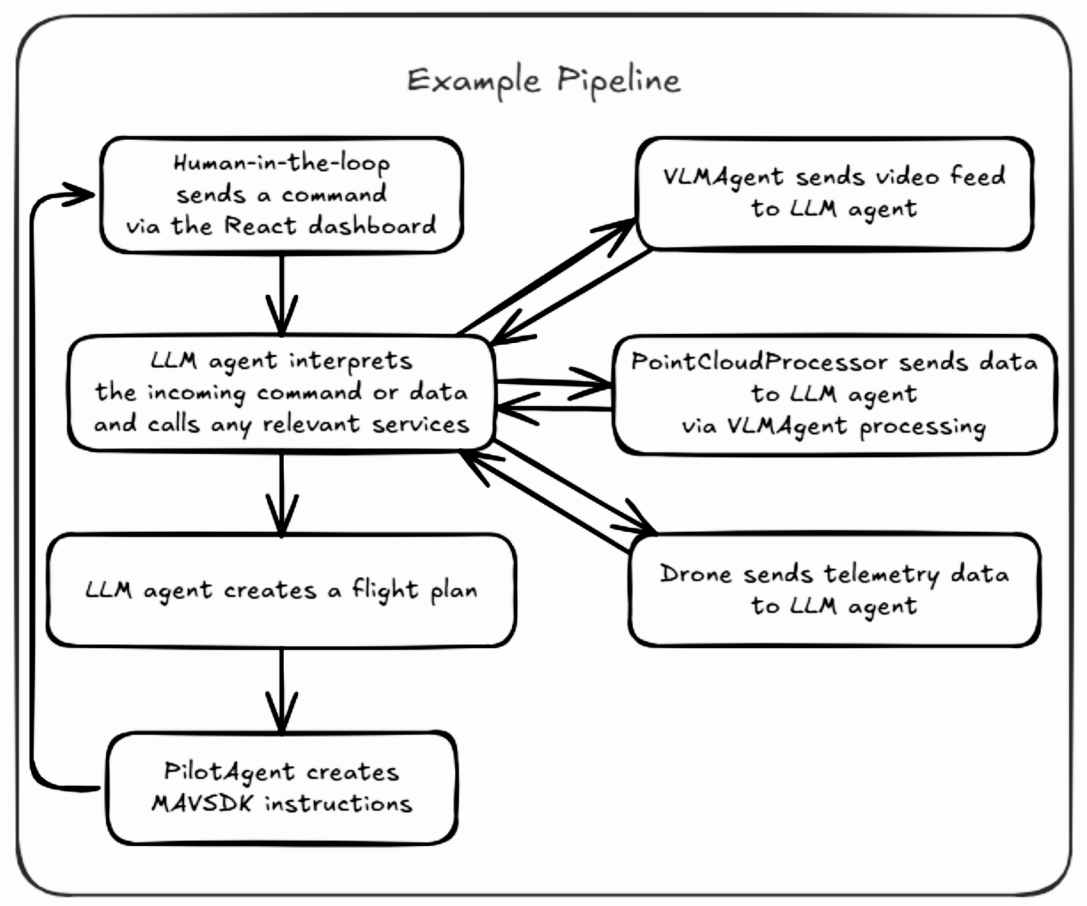

Figure 4.2 – Early Template Workflow *showcasing a design for LangGraph*

Additional reasons we elected to use LangChain and LangGraph included the ability to facilitate conditional branching, such as if we needed a point cloud processor to forward its data to the command interpreter agent in the event of failure or latency in a visual language model agent engaging in object detection. Furthermore, LangGraph and LangChain would allow the overall system to cycle through a series of tasks or repeat a specific task, as we anticipated needing a drone agent to continue proceeding with autonomous navigation for multistep missions, and we also envisioned needing a vision language model agent to continuously process image data until an object was discovered.

4.3.6 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 have offered several potential 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.

In the course of our project's progression, we decided to postpone dockerization until the final stages of development. However, toward the end of our project, our sponsor informed us that dockerization was no longer a hard requirement among the project specifications, and so we moved dockerization to future goals for subsequent teams to pursue in developing our project further.

4.3.7 Vision Language Model Research

For the system to perceive its environment, our research extended to vision language models (VLMs). A vision language model 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. A VLM would allow the agent to effectively ask questions about what the drone sees.

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

However, we ran into a few difficulties with this approach. The VLMs we tested ran slowly, especially when run locally. A single inference, such as asking the VLM if it saw a particular object in an image could take up to eight seconds; this was not acceptable performance for real-time classification. Additionally, we found that VLMs were not effective at identifying boundaries for where an object would be located in an image.

4.3.8 YOLO Vision Model Research

YOLO was the solution to the issues of inference latency and lack of coordinates that we encountered with VLMs. YOLO is an object detection layer optimized for speed that not only identifies objects but also provides a confidence rating, and it also locates

where they are in the image with bounding boxes. Inference speed proved the main reason for our pivot, since our drone's vision system needed to process frames at a rate of at least 30 FPS in order to perform tasks such as visual servoing (adjusting its position to keep a target in frame) or dynamic obstacle avoidance. These factors made YOLOv8n (You Only Look Once, version 8 nano) an optimal candidate.

As the smallest variant in the YOLOv8 family, the Nano model contained approximately 3.2 million parameters. This lightweight architecture allowed it to achieve inference times as low as 10–15 milliseconds on our drone's Jetson Orin Nano when optimized with NVIDIA TensorRT, significantly outperforming the multi-second latency we encountered with vision language models.

In our GCS-centric architecture, YOLOv8n was planned to serve the function as our agent's "reflex" system, operating in parallel with the VLM's "cognitive" system. While the VLM would be queried intermittently to identify novel objects or interpret complex scene semantics, YOLOv8n would run continuously to track predefined classes (e.g., people, vehicles, or specific landing markers). This dual-model approach would create a hierarchy of perception, with the VLM designating a target (e.g., identifying which object is the red chair) and YOLOv8n taking over the high-speed tracking of that object to guide the pilot agent's control inputs.

Edge optimization research also focused on the specific deployment of YOLOv8n on the Jetson Orin Nano. Unlike standard desktop deployments, edge implementation required converting the PyTorch model weights into a TensorRT engine. This process optimized the neural network layers specifically for the Orin's GPU architecture, utilizing FP16 (half-precision) arithmetic to maximize throughput without compromising detection accuracy. This ensured that the heavy computational load of the object detector would not starve any co-located large language models of necessary system resources.

4.4 Navigation and Virtualization

4.4.1 Evaluating Mapping Approaches

To enable real-time 3D environment reconstruction, we 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.

4.4.2 RViz For Data Visualization

It was immediately clear that we needed a way to visualize what was going on with our sensors and processes. Gazebo lets us see the drone and the environment, but it is restricted to a third-person view. We needed to visualize the sensor data, video feed, point clouds, and occupancy grids.

For development, we used RViz (meaning Robot Vision), a visualization tool built into the Robot Operating System (ROS) ecosystem, to visualize ROS topics. RViz has a comprehensive suite of visualizations for common data types like point clouds, paths, poses, and occupancy grids. It was a very useful tool for debugging our SLAM module. We could visualize the digital twin, the drone moving, and video feed simultaneously which helped diagnose issues such as incorrect coordinate axis or bad odometry data.

However, RViz was not a practical solution for the user facing portion of our project. It has a clunky interface, requires manual startup, overwhelms users with options, and is in a separate window. To get the seamless user experience we wanted, we needed to bring the visualization directly into the dashboard.

4.4.3 Automated Photogrammetry and Point-Cloud Processing

To reduce the amount of manual effort required during scan reconstruction, we planned a fully automated photogrammetry pipeline built around a headless deployment of COLMAP. The system would be designed to continuously monitor a designated folder for incoming image sets from the drone and automatically trigger Structure-from-Motion (SfM) and Multi-View Stereo (MVS) processing when detecting new datasets. This workflow would leverage Python automation scripts, GPU-accelerated COLMAP configurations, and pre-compiled binaries of COLMAP and OpenMVS to generate dense 3D point clouds and textured meshes without operator intervention.

To further refine reconstruction quality, we evaluated additional tools such as OpenMVS, CloudCompare, and Open3D. CloudCompare would provide valuable preprocessing capabilities, including radius-based outlier removal, voxel-grid downsampling to improve performance, and normalization routines to maintain consistent alignment across missions. Open3D could serve as a flexible Python-based framework for scripted filtering, mesh generation, and automated geometric cleanup, offering an alternative pipeline for post-processing reconstruction results.

Together, these tools had the potential to form a robust scan-to-model pipeline capable of supporting our project's requirements. The automated reconstruction workflow would enable a so-called "one-click" experience that reduces operator workload while supporting high-quality spatial modeling. The design also could integrate well with the React dashboard for in-browser model viewing, downloading, or cloud archival. This foundation would position the system for advanced capabilities such as mission-to-mission differencing or multi-drone scene merging in future phases.

4.4.4 Rendering in Unity Using Pcx

To support interactive 3D visualization of reconstructed environments, we evaluated Unity as our rendering engine due to its flexibility, wide file-format support, and compatibility with WebGL. Unity's extensibility, combined with the Pcx plugin, made it effective for large point clouds using optimized GPU shaders and custom materials.

Pcx would natively support .ply file imports, which aligned with our tentative COLMAP-to-Unity workflow. Through testing, we confirmed that the plugin could render millions of points with minimal performance degradation, making it suitable for both desktop and browser-based visualization. We experimented with shader-based coloring techniques, such as elevation gradients, density visualization, and classification overlays, to improve spatial understanding for operators who must quickly interpret 3D data in time-sensitive scenarios such as disaster response.

By adopting Unity and Pcx as the default visualization stack, we would be able to establish a foundation that is technically robust, visually clear, and highly extensible. This experimentation also provided insight into performance trade-offs for increasingly dense scans, helping us set realistic visualization targets for future iterations of the system.

4.4.5 Preparing for Real-Time and WebGL Rendering

While our MVP focused on offline and semi-real-time visualization, we also explored approaches to enable future real-time 3D rendering directly within the browser. Preliminary WebGL builds demonstrated that point-cloud models exported from Unity and processed with Pcx could be rendered in modern browsers without requiring specialized workstations or local software installations.

However, WebGL would introduce practical constraints, including GPU memory limitations, shader compatibility issues, and longer load times for high-density point clouds. To mitigate these challenges, we studied optimization strategies such as Level-of-Detail (LOD) management, occlusion culling for non-visible geometry, and shader-level simplification techniques that reduce displayed point density while preserving perceptual fidelity. These methods could increase scalability and allow the system to handle larger environments or stream ongoing SLAM data more efficiently.

Beyond performance improvements, we considered how WebGL-based visualization could support interactive features such as operator annotations, object labeling, and point selection tools. These capabilities would enable richer mission review and collaborative analysis. Preparing the system for WebGL-based rendering would ensure that future versions of the dashboard could evolve into a fully online, real-time spatial mapping interface capable of supporting larger missions, multi-drone operations, and more advanced autonomy workflows.

4.4.6 Gazebo vs. Unity

Choosing the right simulation environment was a big decision for the project. The simulator would serve as the primary testbed for our entire software stack, allowing for rapid, iterative development and validation in a safe, controlled, and repeatable setting before deploying on the physical drone. Although Unity was our initial foray into simulated environments, we discovered that Gazebo was another strong candidate. While both are powerful tools, Gazebo offered a distinct and decisive set of advantages that aligned 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 featured a state-of-the-art rendering engine, capable of producing photorealistic graphics and visually stunning environments. The high-fidelity visualization capabilities would be incredibly valuable for creating compelling digital twins and for any application where aesthetic quality is paramount. Furthermore, the Unity Asset Store provided a vast marketplace of pre-made 3D models, textures, and scripts, which could dramatically accelerate the process of building complex and detailed virtual worlds.

Unity also made considerable strides in catering to the robotics community with its Unity Robotics Hub. This included 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 presented a very compelling case.

However, for a project like LION-ARES, where the primary challenges were not visual fidelity but rather the intricate integration of AI, control systems, and realistic sensor

physics, the benefits of Unity were outweighed by the foundational strengths of a purpose-built robotics simulator like Gazebo.

Gazebo was not a game engine adapted for robotics, but rather a robotics simulator built from the ground up. Consequently, our decision to pivot to Gazebo was based on its superior integration with the robotics ecosystem, its high-fidelity physics modeling, and its alignment with our open-source software stack.

Gazebo offered a significant advantage with its native, deeply-rooted integration with ROS (and specifically, ROS 2), a core component of our software architecture. In Gazebo, a simulated drone would function not merely as a three-dimensional model, but instead as a full entity that can communicate via standard ROS 2 topics, services, and actions. Sensor data from a simulated camera could be published directly to a ROS 2 topic (e.g., /camera/image_raw, /camera/depth/image_raw), and flight commands could be sent to a ROS 2 topic to which a flight controller subscribes.

This native communication model was well-suited for our project, permitting our ground control station (GCS), agentic AI, and perception pipeline to all interface with the simulated drone via ROS 2 in the exact same way they would interface with the physical drone. This simple portability would eliminate the need for any cumbersome middleware or translation layers like the TCP-based bridges that are often required for Unity. This 1:1 mapping between simulation and reality would dramatically simplify development, reduce potential points of failure, and ensure that code validated in simulation will work on the hardware with minimal changes.

While Unity excelled at visual physics (what looks right), Gazebo excelled at dynamic physics (what behaves right). Gazebo offered a choice of multiple high-fidelity physics engines, such as ODE (Open Dynamics Engine) and DART, all of which were designed to accurately model rigid body dynamics, joint constraints, friction, and aerodynamic

forces, crucial for simulating the flight behavior of our drone with a high degree of realism.

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

In summary, while Unity offered best-in-class graphical rendering, its strengths were not aligned with the primary engineering challenges of the LION-ARES project. Our goal was to build and validate a robust, intelligent, and autonomous system, requiring a simulation environment that prioritized:

- Seamless integration with our ROS-2-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 excelled in all three of these critical areas. It would provide a development environment that could more closely mirror 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 made a strategic decision to prioritize functional fidelity over visual fidelity, the correct trade-off for ensuring the success and robustness of the autonomous drone system.

4.4.7 ROS 1 vs. 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. As a result, we needed a systematic, uniform method of transferring data, which led us to the Robot Operation System (ROS).

ROS has two major versions, leading our team to discuss and deliberate between them. 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 node without a central broker.
- **Quality-of-Service (QoS) profiles**: We can tune reliability and bandwidth to be customized for each topic (e.g., "best-effort" for 30 FPS RGB frames, "reliable" for pose estimates).
- **Secure DDS**: Data-in-flight for sensitive telemetry is encrypted.

Another important factor for why we choose ROS 2 over ROS 1 is that we wanted this project to be future-proof. One of our primary aims of this project is to lay the groundwork and foundation for many more agentic systems in the future. Consequently, although ROS 1 may have had some advantages over ROS 2 for our current project development, it will make little sense for someone several years from now to build their own agentic system via ROS 1.

As a result, we decided to pursue ROS 2 as the pseudo-operating-system backbone for our drone-agentic system.

4.4.8 SLAM Algorithms

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 was critical, as its accuracy, robustness, and computational efficiency directly impact the entire system's performance

Several powerful open-source SLAM algorithms were available within the ROS ecosystem, including ORB-SLAM3, Cartographer, and RTAB-Map. Each had its strengths, but **RTAB-Map (real-time appearance-based mapping)** offered a unique combination of features that aligned perfectly with our project's specific requirements. Two alternatives we considered were:

- **ORB-SLAM3**: renowned for its high accuracy in localization, but used primarily as a visual SLAM system focused on camera tracking and sparse feature maps; less suited for generating the dense, detailed point clouds needed for our digital twin and environmental analysis
- **Google Cartographer**: excellent for creating highly accurate 2D and 3D maps and excels with LiDAR data, but with degraded and less robust performance when utilizing RGB-D cameras

RTAB-Map stood 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 was its graph-based architecture. In the algorithm, the drone would make a graph of keyframes (poses) connected by spatial constraints while exploring. Critically, the algorithm had sophisticated loop closure detection: when the drone re-observed a previously visited location, RTAB-Map would

recognize it based on visual appearance, add a “loop closure” constraint to the graph, and then re-optimize the entire map to correct for accumulated drift. This feature was essential for accurately creating globally consistent maps, especially for long missions or exploration of complex, multi-room environments.

- **Excellent Sensor Fusion and RGB-D Support**: RTAB-Map was designed to work with a variety of sensors, with particularly strong support for RGB-D cameras like our ZED 2i. It effectively fused the visual information (RGB) with depth data to build dense 3D maps of the environment. This versatility also provided a clear path for future expansion, with seamless integration with other sensor types like IMU and LiDAR if we chose to upgrade our hardware.
- **Native ROS 2 Integration and Community Support**: RTAB-Map was well-integrated with the ROS 2 ecosystem, replete with a comprehensive set of well-maintained ROS 2 packages that exposed all necessary topics, services, and parameters for easy integration. This tight integration would significantly reduce 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 community would be invaluable resources for troubleshooting and implementation.
- **Dual 3D and 2D Map Generation**: A key advantage was RTAB-Map's ability to simultaneously produce two critical outputs from a single data stream:
    - **A dense 3D point cloud**, essential for our GCS-based digital twin, providing rich visual and spatial context for the human operator
    - **A projected 2D occupancy grid (costmap)**, required for ROS 2 navigation and exploration packages like nav2_exploration

This dual-output capability would allow 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.

4.4.9 Exploration Algorithms

To achieve true autonomy, the drone must be able to independently explore and map an unknown environment without predefined waypoints. This capability is typically driven by an exploration algorithm, a critical software component that analyzes the current map, identifies unknown areas, and strategically decides where to move to next. The chosen algorithm would need to 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.

Our initial plan was to test out different 3D algorithms via Gazebo until we found an optimal choice, but exploration was relegated to a stretch goal that we did not finish.

4.5 Multi-Drone Coordination and AI Research

Although the Minimum Viable Product for the LION-ARES project centered on a single autonomous drone, the architecture was intentionally designed to support future expansion into multi-drone systems. During early research, we explored how coordination frameworks, communication strategies, and agentic workflows could be extended to enable multiple drones to collaborate on large-scale missions. These investigations guided key architectural decisions and ensured that our system would remain scalable and modular, even if full multi-agent capabilities eventually fell beyond the scope of this current project iteration of autonomous drones.

4.5.1 Planning for Scalable Multi-Agent Systems

From the outset, we recognized that any robust autonomous drone framework would eventually require support for more than one aerial platform. To prepare for this, we researched methodologies that enable multiple drones to divide work efficiently and avoid interference with one another. While our MVP focused on a single unit, the long-term roadmap required establishing principles that would allow the system to evolve seamlessly into multi-agent coordination.

One foundational strategy we identified was grid-based sectorization. In this method, the mission region is divided into discrete spatial cells, with each drone assigned its own non-overlapping sector. This approach potentially simplifies early coordination by preventing redundant coverage, enabling predictable map merging, and reducing the risk of collision or path overlap. The strategy also integrates well with stretch-goal workflows such as post-processing or environment differencing, where keeping scans organized by sector improves clarity and file management.

These early explorations shaped the architectural direction for our project by demonstrating that even simple coordination techniques can produce meaningful multi-drone behavior. Establishing clear separation of work areas, predictable behaviors, and modular mission boundaries laid the groundwork for more advanced agentic and swarm-level coordination models in the future.

4.5.2 Task Distribution and ROS-Based Communication

To support more autonomous coordination, we investigated how ROS 2 could be used for multi-drone messaging and distributed task allocation. ROS naturally provides the abstraction layers needed for multiple agents to broadcast telemetry, camera feeds, SLAM data, and mission status while maintaining clear separation between drones.

Our research focused on the importance of assigning namespaced ROS topics to each drone (e.g., */drone1/camera*, */drone2/odom*), in order to prevent conflicts in message streams and ensure that sensor data is traceable to its originating platform. With this structure, each drone can independently publish:

- Position and orientation
- Sector completion status
- SLAM updates
- Obstacle or anomaly findings
- Battery or health information

We also studied lightweight publisher/subscriber nodes capable of enabling basic inter-drone signaling. For example, when Drone A finishes scanning its sector, it can publish a *“sector_complete*” message that triggers Drone B to begin its assigned region or expand its search area. This creates a scalable foundation for more complex behaviors such as cooperative coverage, dynamic reassignment of tasks, and collaborative map building.

These ROS-based strategies validated that multi-drone communication does not require complex swarm algorithms at the start, and that even simple messaging channels can support semi-autonomous collaboration within a centralized GCS-supervised architecture.

4.5.3 AI-Driven Coordination Frameworks

Beyond low-level messaging, we explored higher-order multi-agent coordination using agentic AI frameworks such as LangChain and LangGraph. These tools support independent “agents” that can plan, observe, communicate, and refine their decisions based on mission feedback.

This research demonstrated how a multi-drone system could evolve into a fully agentic architecture. For example:

- Drone A, acting as an agent, scans Sector 1 and identifies incomplete coverage.
- It sends a structured request to Drone B via LangGraph’s communication layer.
- Drone B autonomously adjusts its flight plan to assist with that region.
- The GCS supervises the interaction, ensuring safety and global mission consistency.

Alternatively, the system may employ a centralized planner agent running at the GCS level. This planner receives updates from each drone and dynamically assigns tasks based on battery levels, map completeness, or operator-defined objectives. This structure supports distributed execution while preserving centralized oversight.

While full implementation of multi-agent planning exceeded the MVP’s scope, the research validated our decision to use a GCS-centric architecture and helped shape the modularity of the agentic AI pipeline.

4.5.4 Swarm Behavior and Control Models

To further understand how multiple drones could collaborate autonomously, we examined several well-established control models used in swarm robotics and multi-agent systems. These models helped guide how we could later design scalable coordination behaviors:

Boids Algorithms:
This model simulates flocking behavior through three core rules: attraction, repulsion, and alignment. Though originally developed for animation, Boids algorithms have become fundamental in swarm robotics as they allow decentralized coordination without requiring a central controller. This model provides a blueprint for emergent group behavior in future LION deployments, such as drone formations, follow-the-leader navigation, or coordinated scanning.

Potential Fields:
This approach models goals and obstacles as attractive or repulsive forces in the environment. Drones follow gradient vectors toward mission objectives while avoiding collisions dynamically. Potential fields support continuous, adaptive motion planning and pair well with real-time mapping tools such as SLAM, making them useful for distributed coverage or hazard avoidance missions.

Behavior Trees:
Behavior trees organize complex decision-making into modular, prioritized branches that execute conditionally at runtime. They provide a scalable method for managing high-level logic and allow drones to react to environmental changes, operator commands, or agentic requests. Their readability and modularity make them practical for developing reusable coordination logic across multiple drones.

Although these control models were not implemented in the MVP, they heavily influenced our architectural design. They provided the theoretical foundation for future features such as autonomous formation flight, distributed task assignment, cooperative object detection, and adaptive mission planning.

4.5.5 Planning Around Human-AI Collaboration

Throughout all multi-drone research, we placed significant emphasis on maintaining clear human-AI collaboration principles. Even in highly autonomous multi-agent systems, human oversight remains essential for safety, transparency, and accountability.

We explored ways to ensure operators can override decisions, monitor multi-drone mission state, and receive clear explanations of the system's reasoning. By reporting all coordination decisions back to the centralized GCS, we preserved the operator's role as the final authority in mission execution. Integration of natural-language summarization through LLMs also opens the door for more intuitive collaboration in the future. For example, commands such as *"Send a second drone to support Sector C*" can be routed through the AI planner, translated into actionable drone behavior, and executed safely.

This human-centered design philosophy ensures that even as multi-agent autonomy expands, the system remains understandable, predictable, and controllable. It supports responsible deployment and ensures the system can be trusted in sensitive, critical, or safety-constrained environments.

4.6 Ground Control Station Research

The most foundational design decision was the selection of our core computing architecture, as this choice would dictate 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 ground control station (GCS) centered architecture, which strategically splits processing tasks between the drone and a powerful ground station (see *Figure 4.3* below for an analysis summary).

Criteria	Drone-Centric	GCS-Centric
Computational Power	**Limited**: Smaller, quantized AI models with reduced performance must be used due to size, weight, and power constraints of an onboard computer.	**High**: Powerful GPUs (e.g., NVIDIA Jetson or desktop-class) on the ground can be leveraged to use larger, more accurate, and more capable AI models
Payload Weight & Flight Dynamics	**Significantly increased**: The weight of an AI-capable compute module, cooling system, and dedicated power source reduces flight time and agility.	**Minimized**: The drone carries only essential components, keeping it lightweight and maximizing flight endurance and maneuverability.
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	**Simplified**: The ground station can employ large, active cooling solutions (fans, heatsinks) without weight limits, ensuring stable, sustained performance without overheating.
Power Consumption	**High onboard consumption**: AI processors draw significant power from the main flight battery, drastically shortening mission duration.	**Minimal onboard consumption**: The drone can dedicate its power budget primarily to propulsion; the ground station has its separate power source.
Development Agility	**Slow and cumbersome**: Debugging and software updates would require physical interaction with the drone, and managing onboard dependencies and libraries would be complex.	**Fast and efficient**: AI software can be developed, tested, and debugged on a standard workstation, facilitating rapid iteration and integration with simulation tools.

***Figure 4.3 – Architectural Models**: comparing drone-centric and GCS-centric models*

4.6.1 Computational Power

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, allowing language models to achieve 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.

4.6.2 Payload Weight & Flight Dynamics

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.

4.6.3 Thermal Management

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.

4.6.4 Power Consumption

In an onboard-centric model, the drone would require a significant amount of power consumption. The cumulative weight of housing an AI-capable compute module, a cooling system, and a dedicated power source would negatively impact the drone's flight time and agility. In contrast, power consumption for the drone in a GCS-centric model would be much more minimal, since the drone can dedicate its power budget primarily to propulsion; the ground control station is able to use a stable, external power source, fully independent of the battery on the drone.

4.6.5 Development Agility

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.

4.6.6 Final Selection

After a thorough analysis of the project's objectives and constraints, the GCS-centric model was selected as the optimal approach.

Consequently, we decided to treat the drone as a highly capable mobile sensor platform and actuator whose primary responsibilities would be limited to the reliable acquisition of high-fidelity sensor data and the precise execution of flight commands. We planned for the Raspberry Pi on the drone to focus 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. In effect, the drone would not perform any high-level reasoning or AI model inference onboard.

In contrast, the ground control station would function as the centralized intelligence hub of the entire system. All computationally intensive tasks would be offloaded to this ground-based computer, in particular an NVIDIA Jetson platform chosen for its powerful GPU acceleration. The ground control station would be responsible for running the core AI stack, including the large language model for command interpretation and planning, the visual language model for scene analysis, the generation of three-dimensional point clouds from the received depth data, and the execution of the main agentic framework that orchestrates the entire operation. This division of labor would allow the project to leverage processing power beyond what an aerial platform could handle by itself, which would enable the use of more sophisticated and capable AI models. These two halves of our system would be 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.

In conclusion, while a fully self-contained, onboard AI system may represent an ideal future, the GCS-centric architecture provided the most practical and powerful path forward for our project. It maximized AI capability, reduced airframe complexity, and created an agile development environment making it fundamental to the LION project's feasibility and success.

5. Diagram Design Documentation

5.1 Block Diagram

The block diagram on the following page (*Figure 5.1*) illustrates the high-level hardware architecture and data flow of the project. The system features a split-architecture paradigm, dividing responsibilities between the **Aerial Platform (S550 Hexacopter)** and the **Ground Control Station (GCS)**. This separation ensures that critical flight controls remain isolated on the drone, while agentic and dashboard components are housed on the ground.

The system consists of three primary interaction zones:

- **The Drone (Aerial Platform)**: built on an S550 Hexacopter frame and houses the avionics and immediate perception hardware
    - **Pixhawk Flight Controller (PX4)**: manages real-time flight stability, motor control, and safety failsafes, is powered by a dedicated module, and communicates via MAVLink
    - **NVIDIA Jetson Orin Nano (Onboard Companion)**: captures raw RGB and depth data from the **ZED 2i Stereo Camera** and streams this high-bandwidth data to the ground via a dedicated Wi-Fi module, and is powered by a separate battery elimination circuit (BEC) to ensure power isolation from the flight controller
    - **Power Distribution**: provided by a 4S LiPo battery, distributed via the Power Distribution Board (PDB) to the ESCs/Motors, the Pixhawk (via Power Module), and the Jetson (via BEC)
- **The Ground Control Station (GCS)**: the command center for the system

- **Main GCS Computer**: a laptop hosting the **React Dashboard** and the **LangGraph Agentic AI**, acting as the system “brain”, processing the video streams from the drone and high-level commands from the user
- **Communication Hardware**: comprises a **WiFi Router** to receive high-speed video/sensor data from the Jetson and a **SiK Telemetry Radio** to maintain a low-latency, long-range link with the Pixhawk

- **Data and Power Topology**
  - **Blue Arrows (Data Flow)**: shows the bidirectional flow of information, as high-bandwidth perception data flows from the drone to the GCS via WiFi, while telemetry and control commands flow via the SiK radio link
  - **Red Dotted Arrows (Power Flow):** illustrates the regulation and distribution of power from the main 4S LiPo battery to the various sub-modules, highlighting the use of specific regulators (BEC and Power Module) to voltage-match each component

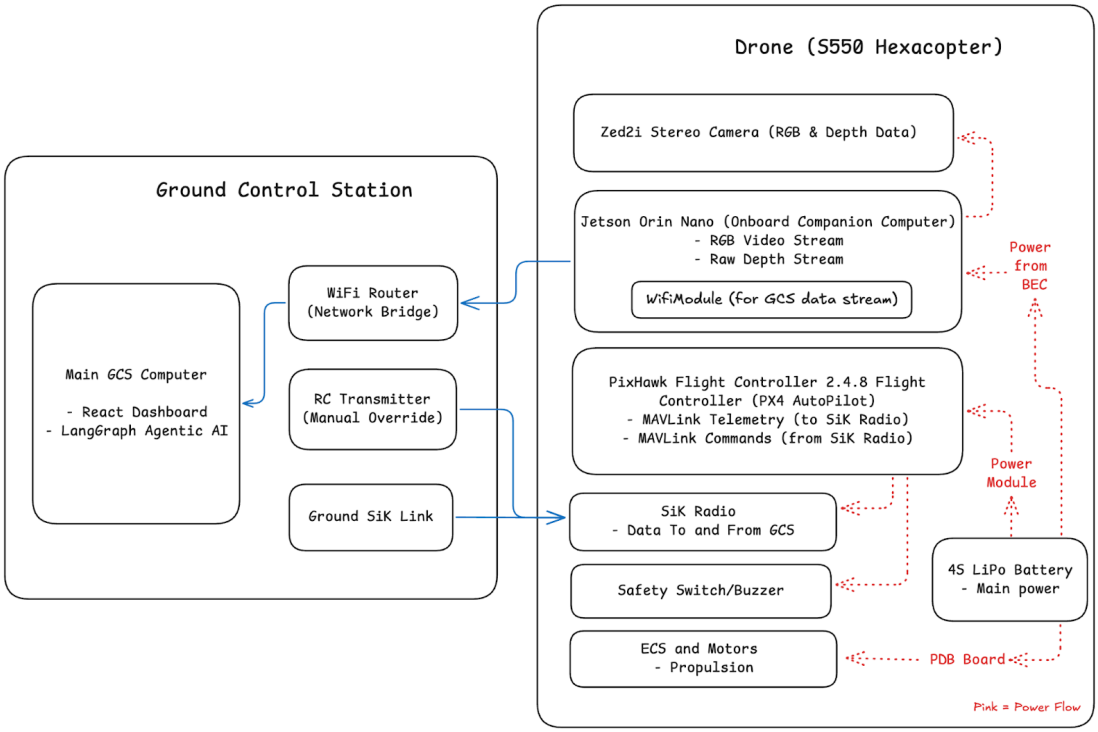

Figure 5.1 – Block Diagram *showing interactivity between the drone and the GCS*

5.2 Use-Case Diagram

The User Interaction Diagram (see *Figure 5.2* below) illustrates the system's modular nature, designed to handle concurrent execution paths for mission control, data visualization, and safety management. Users can issue prompts via a WebSocket connection to instruct the agent. In parallel, the telemetry and safety pipeline (highlighted in red) operates as a continuous, high-priority background process, ensuring that critical flight logs and the “Manual Override” capability remain accessible. Additionally, users can toggle on the live video feed and then swap the source between the left and right cameras on our stereo camera. Lastly, the digital twin pipeline allows users to view a live construction (digital twin) as the drone moves, which they can then save locally via IndexDB. Users can also review these models in later sessions.

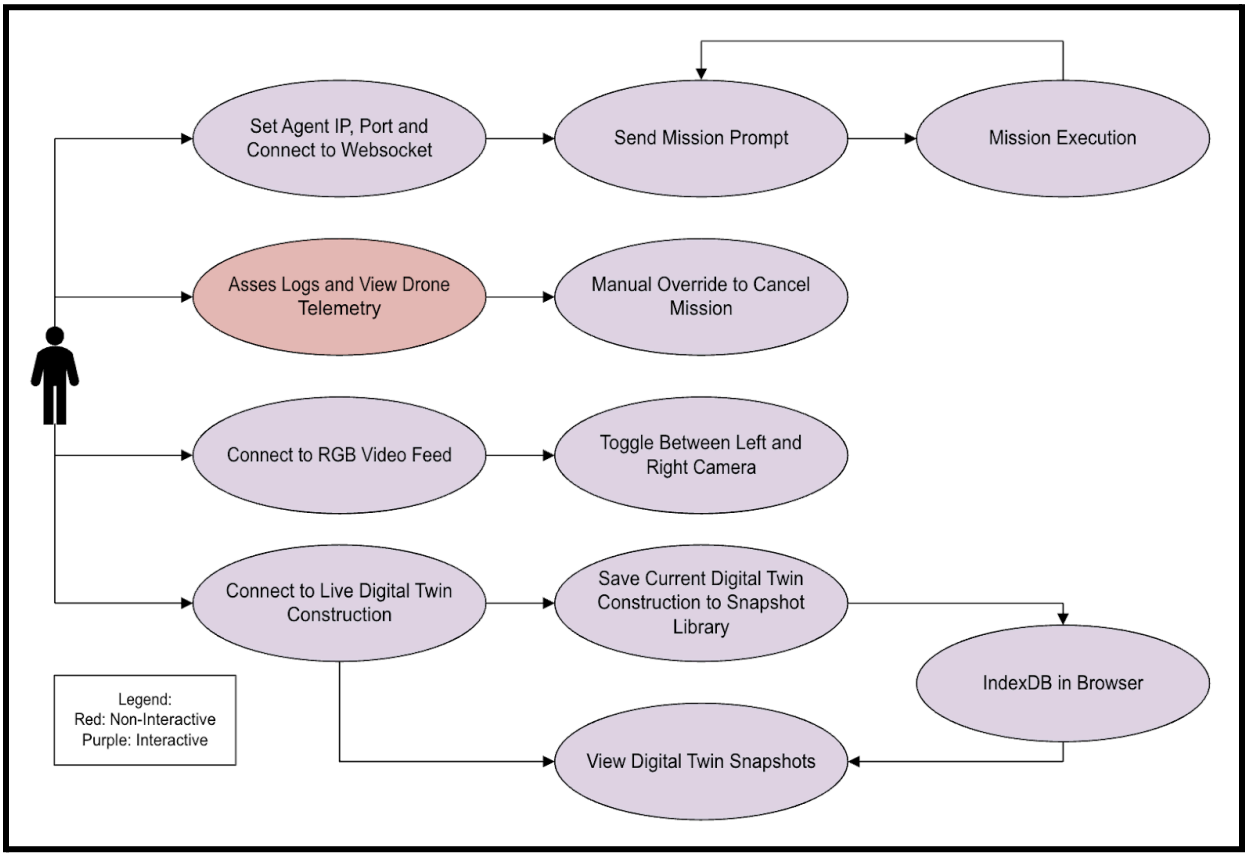

Figure 5.2 – Use-Case Diagram

5.3 Activity Diagram

This diagram (see *Figure 5.3* below) illustrates the system's primary loop. A user's natural language command is received by the ground control station, where a LangGraph agent leverages a large language model to interpret the mission. It plans steps with tools available to it and then In an iterative loop, the LLM decides whether to query the vision tools for perceptual data or to direct the mav-sdk tools to perform a flight maneuver. When a flight action is commanded, the agent translates the goal into MAVSDK instructions, which are sent via the MAVLink protocol to the drone's flight controller for execution.

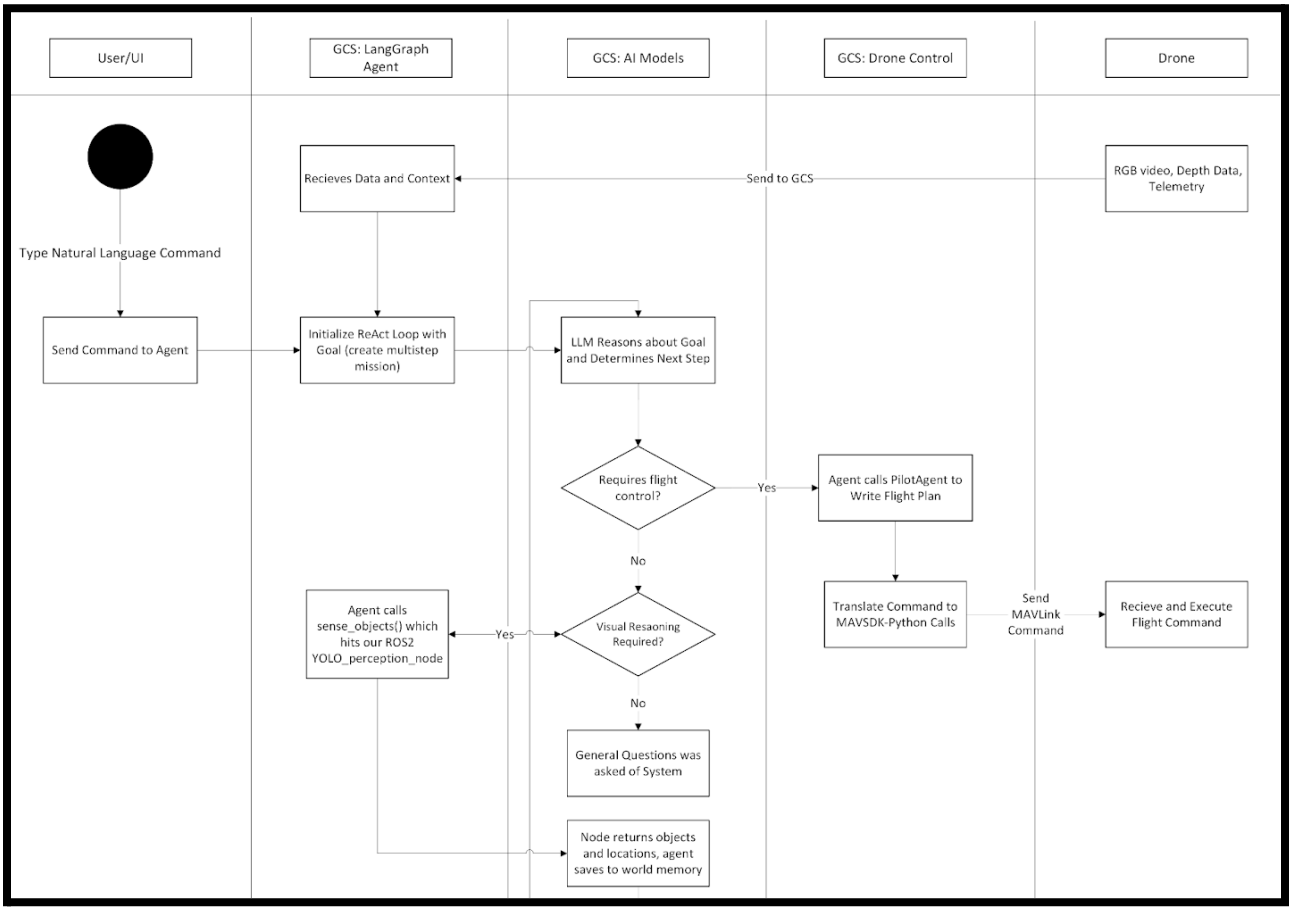

Figure 5.3 – Activity Diagram *with individual lanes*

6. Hardware Design Documentation

Our hardware architecture is twofold: the drone plus the ground control station.

6.1 Drone

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. Our drone components are as follows:

- **Airframe**: S550 Hexacopter Frame Kit
- **Propulsion System**: REC Technology 2212-920KV Motors
- **Motor Control**: REC Technology 30A Electronic Speed Controller
- **Flight Control**: Pixhawk 2.4.8
- **Navigation**: M8N GPS Module
- **Telemetry**: SiK Telemetry Radio
- **Manual Control**: RadioLink AT9S Pro
- **Companion Computer**: NVIDIA Jetson Orin Nano
- **Perception Sensor**: ZED 2i Stereo Camera

6.1.1 Airframe: S550 Hexacopter Frame Kit

The structural foundation of our autonomous drone system is the S550 Hexacopter Frame Kit, a platform selected to address the specific heavy-lift and redundancy requirements of the project's advanced AI payload. The S550 features a 550mm diagonal wheelbase, placing it in a size class that provides a substantial footprint for stability while remaining portable enough for single-person deployment in field testing environments.

The frame is constructed using a hybrid material approach designed to optimize the strength-to-weight ratio. The central hub consists of two high-strength PCB (Printed Circuit Board) plates, a top deck and a bottom deck, which function to sandwich the arm attachments. These plates also serve a dual purpose: they provide the structural rigidity to resist torsional flexing during high-yaw maneuvers, and they function as the drone's electrical backbone.

The six arms of the frame are manufactured from high-density composite materials reinforced with embedded carbon fiber rods. This composite construction is critical for vibration damping; unlike pure carbon fiber arms which can transmit high-frequency motor noise directly to the accelerometer, the composite material helps absorb the mechanical noise generated by the propulsion system, providing a cleaner mechanical environment for the flight controller’s sensitive Inertial Measurement Units (IMUs). The frame also features specialized “upswept” arms, which provide a slight dihedral angle. This geometric design feature naturally enhances the drone's self-leveling stability in hover, reducing the workload on the flight controller’s stabilization loops.

A key feature of the S550 frame is its integrated Power Distribution Board (PDB). The bottom plate of the central hub contains embedded high-current copper traces that route power from the main battery connector to the six arm terminals. This integration eliminates the need for a separate, standalone PDB, thereby reducing the total wiring mass and minimizing the number of failure points in the system, such as solder joints or cable crimps. For our build, an XT60 connector with 10CM 14AWG silicone wire functions as the main battery lead and is soldered directly to the central pads, providing a low-resistance path for the high amperage required by the six motors. The voltage regulation hardware, specifically the Power Module for the Pixhawk and the BEC for the Jetson, also taps directly into these central pads, ensuring that all subsystems share a common ground reference, which is vital for preventing ground loops in the signal lines.

The S550 is equipped with extended, high-profile landing gear attached to the frame's center. This design provides significant ground clearance, which is a critical requirement for our specific payload configuration. Since the ZED 2i stereo camera is mounted in an underslung position to maximize its field of view, the tall landing gear ensures the camera along with other expensive components are kept well above wet grass, dust, or uneven terrain during takeoff and landing operations. The landing gear's wide stance also minimizes the risk of tipping over during rough landings, protecting the top-mounted Jetson Orin Nano and GPS mast.

6.1.2 Propulsion System: REC Technology 2212-920KV Motors

The propulsion system serves as the drone's actuator, translating electrical energy from the battery into the aerodynamic thrust and torque required to manipulate the airframe's six degrees of freedom. This system has been specifically dimensioned to lift the total operational weight of the drone (approx. 2.5kg) while maintaining a hover throttle of roughly 50%, which leaves sufficient overhead for aggressive maneuvering and wind resistance.

The system utilizes six REC Technology 2212-920KV brushless DC outrunner motors. The 2212 designation refers to the stator size (22mm diameter, 12mm height), a standard form factor known for its reliability and thermal efficiency. The 920KV rating is the velocity constant, indicating that the unloaded motor will rotate at 920 RPM for every volt applied. This specific KV rating is engineering-optimized for 3S and 4S voltage architectures driving 9 to 10-inch propellers. In our configuration, powered by a 4S LiPo (with a nominal 14.8V), these motors can spin the suggested 10-inch propellers at upwards of 13,000 RPM at full throttle.

The six-motor architecture instead of the common four-motor version provides a critical layer of redundancy known as engine-out capability. In a standard quadcopter, the loss of a single motor or ESC results in an immediate, catastrophic loss of yaw and

roll control, leading to a crash. On a hexacopter like the S550, the flight controller can often compensate for a single motor failure by over-driving the remaining five motors, allowing the drone to maintain a stable, albeit degraded, hover and perform a controlled emergency landing. This redundancy is non-negotiable for an autonomous system carrying high-value computing hardware like the Jetson Orin Nano and high-value stereo video sensors like the ZED 2i camera.

6.1.3 Motor Control: REC 30A Electronic Speed Controller

Controlling each motor is a REC Technology 30A Electronic Speed Controller (ESC). The ESC acts as the bridge between the low-voltage digital logic of the flight controller and the high-voltage, high-current requirements of the motors. These units take the PWM (Pulse Width Modulation) signal updates from the Pixhawk and rapidly switch the MOSFET gates to energize the motor's electromagnetic coils in the correct sequence. We selected 30A rated ESCs to provide significant thermal headroom. While the 2212 motors typically draw 15-20A at maximum load, using 30A ESCs ensures that the components run cooler and are less likely to overheat during prolonged flights or in high-ambient-temperature environments. The ESCs are rated for 2S-4S LiPo input, matching our power system. They feature 2.5mm banana connectors, which allow for a solder-free connection to the motors. This modularity is crucial for the research workflow, since if a motor bearing wears out or a winding is damaged, the motor can be swapped in minutes without requiring a soldering iron in the field during live testing.

6.1.4 Flight Control: Pixhawk 2.4.8

The central nervous system of the drone is the Pixhawk 2.4.8 Flight Controller. While newer iterations of the Pixhawk standard exist (such as the 6C), the 2.4.8 was selected as the most cost-effective, battle-tested solution that still supports the full feature set of the PX4 Autopilot firmware. It creates a reliable, standardized interface for our custom software to interact with the physical world.

The Pixhawk 2.4.8 is powered by a 32-bit STM32F427 Cortex-M4F core running at 168 MHz. This processor is responsible for internal control tasks, including reading sensor data at high frequencies (400Hz+), running the Extended Kalman Filter (EKF) to estimate the drone's attitude, and calculating the PID (Proportional-Integral-Derivative) corrections needed to keep the drone stable. The unit contains a redundant suite of internal sensors to ensure reliability. It typically houses:

- **L3GD20H:** A 3-axis gyroscope for measuring angular velocity
- **LSM303D:** A combined 3-axis accelerometer and magnetometer for measuring linear acceleration and heading
- **MPU6000:** A 6-axis accelerometer/gyroscope connected via SPI for high-speed data access
- **MS5611:** A high-precision barometer for measuring altitude via atmospheric pressure changes

The PX4 firmware continuously compares data from these redundant sensors. If one sensor begins to drift or provide noisy data due to vibration, the EKF can reject that data source and rely on the others, preventing a “fly-away” event where one sensor leads the drone to take increasingly drastic or unpredictable maneuvers to try to “fix” its movement or position.

Vibration is the enemy of stable flight, as high-frequency noise from the motors can confuse the accelerometers. To mitigate this, the Pixhawk 2.4.8 is mounted on a dedicated Glass Fiber Shock Absorber. This component consists of two plates separated by soft rubber dampeners. The flight controller sits on the so-called “floating” top plate, mechanically decoupled from the frame's high-frequency vibrations. This hardware-level filtering is essential for achieving the stable hover required for our vision systems to operate effectively.

The Pixhawk provides a wealth of connectivity options fully utilized in our architecture:

- **TELEM1 & TELEM2:** Serial ports used for the radio telemetry and the link to the Jetson companion computer
- **GPS Port:** Dedicated interface for the external GNSS module
- **MAIN OUT:** 8 PWM outputs used to drive the 6 ESCs
- **POWER:** A dedicated port for the Power Module, which provides regulated 5.3V power to the flight controller while simultaneously sensing the main battery's voltage and current draw, allowing the Pixhawk to trigger a “Low Battery Return-to-Home” failsafe
- **Safety Switch/Buzzer:** An external module that provides a physical safety arming button and a loud piezoelectric buzzer to indicate status (e.g., arming errors, low battery warnings) to the pilot on the ground

6.1.5 Navigation: M8N GPS Module

While the IMUs handle stability, global navigation requires an external reference. Our system utilizes an M8N GPS Module equipped with a high-gain ceramic patch antenna.

The M8N module is a concurrent GNSS receiver, meaning it can simultaneously track satellites from multiple constellations (GPS, GLONASS, Galileo) to improve position accuracy and lock speed. Under open-sky conditions, this module typically secures a 3D lock with 12-20 satellites, giving an accurate horizontal position to approximately 2 meters. The module also houses an external HMC5883L digital compass.

The magnetic fields generated by the high currents flowing through the drone's PDB and motors can severely distort compass readings. To solve this, the GPS/Compass module is mounted on a collapsible mast, elevating it approximately 10cm above the frame. This physical separation moves the sensor out of the drone's electromagnetic interference noise floor, ensuring the heading data remains accurate. Without this

reliable heading, the drone would be unable to navigate to waypoints or hold its position in the wind.

6.1.6 Telemetry: SiK Telemetry Radio

To maintain a robust Command and Control (C2) link, the system employs a SiK Telemetry Radio set operating at 915 MHz. This system consists of two matched transceivers: an Air Module connected to the Pixhawk's TELEM1 port, and a Ground Module connected via USB to the GCS laptop. Unlike Wi-Fi, which is high-bandwidth but short-range and latency-prone, these radios use a lower frequency and Frequency Hopping Spread Spectrum (FHSS) technology to create a low-bandwidth (57600 baud), high-latency-tolerance link that can penetrate obstacles and reach distances of over a kilometer. This link is dedicated strictly to MAVLink protocol traffic. It allows the *PilotAgent* on the ground to send commands (e.g., COMMAND_LONG, SET_POSITION_TARGET) and receive vital health telemetry (Battery Voltage, Attitude, GPS Fix) even if the high-speed video link fails. This separation of the control plane (915 MHz) and the data plane (Wi-Fi) is a critical safety architecture choice.

6.1.7 Manual Control: RadioLink AT9S Pro

While the LION system is designed for high-level autonomous agency, safety regulations and engineering best practices mandate a robust, low-latency manual override mechanism. For this critical human-in-the-loop layer, the system employs the RadioLink AT9S Pro 12-channel transmitter paired with a compatible high-frequency receiver onboard the drone. This subsystem serves as the ultimate authority in the control hierarchy; regardless of the commands being issued by the AI agent or the GCS, inputs from this manual transmitter are prioritized by the flight controller to allow for immediate intervention during emergency scenarios.

The AT9S Pro was selected specifically for its hybrid communication protocol, which utilizes both DSSS (Direct Sequence Spread Spectrum) and FHSS (Frequency Hopping Spread Spectrum) simultaneously. Standard RC signals can suffer from packet loss or latency spikes in an environment saturated with 2.4GHz noise, including from the onboard Wi-Fi module, the high-speed switching of the Jetson's USB 3.0 peripherals, and external urban networks. The AT9S Pro's hybrid modulation actively combats this by hopping across 16 channels pseudo-randomly (FHSS) while spreading the signal power over a wider bandwidth (DSSS). This creates a highly resilient control link capable of maintaining signal integrity up to 3.4km (2.1 miles) in open air, providing a safety radius that far exceeds the drone's typical visual line-of-sight operational envelope.

Unlike traditional "blind" transmitters, the AT9S Pro supports bidirectional telemetry. When paired with the drone's Pixhawk flight controller, the radio does not just send stick commands, but rather receives vital health data back from the aircraft. The transmitter's 2.8-inch LCD screen is configured to display real-time metrics such as the RSSI (Received Signal Strength Indicator) and the main 4S battery voltage. Crucially, the transmitter features a programmable alarm system that utilizes both auditory alerts (DD sounds) and haptic feedback (vibration). This is a vital human-factors engineering feature. It allows the safety pilot to remain visually focused on the drone or the GCS dashboard without needing to look down at the controller screen. If the drone's battery reaches a critical voltage threshold or if the radio link quality degrades due to obstruction, the controller physically vibrates, prompting the pilot to initiate an immediate manual return-to-home or landing procedure.

The receiver unit connects to the RC IN port of the Pixhawk 2.4.8 via the SBUS (Serial Bus) protocol. SBUS was chosen over the older PPM standard because it is a digital, serial protocol that offers lower latency and higher resolution (2048 steps) for stick

movements. The 12-channel capability of the AT9S Pro allows for extensive switch mapping beyond the basic 4-axis flight controls (Throttle, Yaw, Pitch, Roll):

- **Channel 5 (Flight Mode Switch)**, mapped to a 3-position toggle switch, allowing the pilot to instantly transition the drone from “Offboard” (AI control) to “Position” (GPS-assisted hover) or “Stabilize” (Manual leveling) modes
- **Channel 7 (Emergency Kill/RTL)**, mapped to a guarded switch to trigger an immediate “Return to Land” or motor emergency stop in the event of a software runaway, with the hard-wired integration ensuring that even if the Jetson companion computer freezes or the Wi-Fi link is severed, the human pilot retains full physical authority over the airframe

6.1.8 Companion Computer: NVIDIA Jetson Orin Nano

In the initial designs, a Raspberry Pi 5 was considered. However, the final architecture necessitated a significant upgrade to the NVIDIA Jetson Orin Nano (8GB Developer Kit). While the Pixhawk manages the physics of flight, the Jetson serves as the cognitive brain of the agent, responsible for all high-level perception and decision-making tasks.

The Orin Nano is built on the NVIDIA Ampere GPU architecture, featuring 1024 CUDA cores and 32 Tensor cores. This hardware is specifically optimized for parallel processing and matrix operations, which are the fundamental math behind modern AI. It delivers up to 40 TOPS (Trillions of Operations Per Second) of AI performance, a magnitude of difference compared to the CPU-bound Raspberry Pi. This compute power allows us to locally run complex, uncompressed neural networks on the drone at real-time frame rates (30+ FPS), such as the YOLOv8n object detector and the ZED SDK’s depth engine. The 8GB of unified system memory is shared between the CPU and GPU, allowing for zero-copy data transfers. This is essential for our vision pipeline,

where large 1080p image buffers need to be processed by the GPU without the latency penalty of copying data back and forth between separate RAM pools.

The Jetson is physically mounted on the top plate of the S550 frame to ensure adequate airflow for its passive heatsink and active cooling fan. It connects to the system in three key ways:

1. **Power**: The Jeston receives a regulated 12-19V DC input from a dedicated Battery Elimination Circuit (BEC) soldered to the PDB. This isolates the sensitive computer from the voltage sags that occur when the motors draw high current.
2. **Data Link (Pixhawk)**: The Jetson connects to the Pixhawk's TELEM2 port via a UART-to-USB adapter or direct GPIO serial. This link runs a high-speed MAVLink instance (921600 baud), allowing the Jetson to send "Offboard" velocity control commands directly to the flight controller.
3. **Network**: An Intel 8265 M.2 Wi-Fi module installed in the Jetson's carrier board provides the high-bandwidth data link to the ground control station. This link carries the heavy payloads: the compressed video stream from the ZED camera and the debug visualizations from the AI agents.

6.1.9 Perception Sensor: ZED 2i Stereo Camera

The "eyes" of the system are the Stereolabs ZED 2i stereo camera. This sensor was selected for its industrial-grade ruggedness and its ability to provide human-like depth perception without the active emissions of a LiDAR system.

The ZED 2i functions on the principle of triangulation. It uses two separate 4-megapixel lenses spaced 120mm apart (the baseline) to capture two slightly offset images of the world. The Jetson Orin Nano processes these images to calculate the disparity between the pixel position of objects in the left and right cameras, and then uses this data to generate a real-time depth map. This depth map provides a distance value for

every pixel in the image, allowing the drone to understand the 3D structure of its environment, detecting obstacles from 0.2 meters out to 20 meters.

Beyond just simple imaging, the ZED 2i contains its own high-frequency internal IMU, barometer, and magnetometer. The ZED SDK performs sensor fusion, combining the visual tracking data (how features move across the frame) with the inertial data (acceleration and rotation) to perform Visual-Inertial Odometry (VIO). This gives the drone a highly accurate awareness of its own motion and position in space, independent of the GPS. This capability is crucial for the project, as it allows the drone to maintain stable flight and accurate navigation even in GPS-denied environments (like indoors or under tree canopies) where the external M8N GPS module might lose its satellite lock. The camera connects to the Jetson via a USB 3.0 Type-C cable, which carries both the high-bandwidth video data and the power required to run the camera's internal ISPs.

6.1.10 Power Distribution and Regulation Architecture

Reliable power delivery is the single most critical factor in drone reliability. The electrical architecture for the LION project is designed around a bifurcated power topology, utilizing a high-amperage parallel Y-splitter to segregate the noisy propulsion loads from the sensitive computational payloads. This design strategy minimizes electrical noise coupling and ensures that voltage sags caused by aggressive motor maneuvers do not trigger brownouts in the onboard AI computer.

The system is energized by a high-discharge four-cell (4S) Lithium Polymer battery. This battery has a nominal voltage of 14.8V and a fully charged peak of 16.8V. The 4S voltage class was selected specifically to optimize the efficiency of the propulsion system, as higher voltage allows for lower current draw at equivalent wattage, reducing resistive heating in the wiring and ESCs. The battery connects to the system via a

robust XT60 connector, an industry-standard high-amperage interconnect capable of sustaining the continuous current draw of the six motors.

Immediately downstream from the battery, the electrical path divides into two distinct parallel circuits using an XT60 Y-Splitter. This passive component features one female XT60 connector (battery input) splitting into two male XT60 connectors (output) for:

1. **Domain A (Avionics & Propulsion)**: feeds the flight controller and motors
2. **Domain B (AI Compute)**: feeds the Jetson Orin Nano

Domain A is the first leg of the Y-splitter connected to the Power Module. This module performs two critical functions. First, it serves as a sensing element, containing a shunt resistor to measure real-time battery voltage and current draw, which is reported back to the Pixhawk via a 6-pin cable. Second, it contains a specific 5.3V battery elimination circuit (BEC) that powers the Pixhawk flight controller itself. The high-current output of the power module then continues directly to the S550 frame's integrated PDB. Here, the raw battery voltage is distributed to the six ESCs. By placing the propulsion system on this specific branch, we ensure that the flight controller gets the most direct reading of the battery status, allowing for accurate battery life gauging and failsafe triggering.

Domain B is the second leg of the Y-splitter dedicated to the NVIDIA Jetson Orin Nano. While the Jetson can technically accept a voltage range of 9V-19V, direct connection to the fluctuating battery voltage of a drone is poor engineering practice. Rapid throttling can cause voltage spikes and drops that may destabilize the GPU. To mitigate this, this branch connects to a DC-DC Step-Up/Step-Down Converter (UBEC). We soldered to connect the input of this regulator to the second male XT60 connector. The converter actively conditions the power, transforming the variable input voltage into a locked, clean ~12V output. This regulation provides a firewall against voltage ripple. The output of this UBEC is terminated with a 5.5mm OD x 2.1mm ID Barrel Jack, the specific power interface required by the Jetson carrier board. This ensures that the AI computer

receives a constant, noise-free power supply regardless of whether the drone is hovering calmly or executing high-throttle maneuvers.

6.2 Ground Control Station

The ground control station (GCS) serves as the primary interface between the human operator and the autonomous agent.

Although previous iterations saw the GCS functioning as the primary computational node, our current architecture leverages the GCS as a supervisory command center and visualization hub. While the S550 Hexacopter conducts real-time perception and decision-making via its onboard avionics, the GCS monitors system health, issues high-level mission directives, runs the agentic AI, and provides the human operator with a real-time telemetry and video dashboard. This separation of concerns ensures that critical flight operations are handled locally on the airframe itself, while the GCS remains focused on targeting situational awareness and human-in-the-loop safety protocols.

Our primary ground control station components are as follows:

- **Host computer**: Acer Predator Helios 300
- **Network Core**: TP-Link AC1900 Wi-Fi Router
- **Network Extension**: TP-Link EAP225 Omada Access Point
- **Command and Control Link**: SiK Telemetry Radio

6.2.1 Host Computer

The core of the ground control station is a high-performance laptop responsible for rendering the user interface and managing communication with the drone's onboard network. This host machine runs the client-side dashboard application, which aggregates data streams transmitted from the drone, including the RGB video feed, depth map visualizations, and agent reasoning logs, while also facilitating dialogue between the human-in-the-loop and the agentic AI.

By offloading the AI processing of vision perception and point-cloud calculation to the airborne Jetson Orin Nano, the GCS hardware requirements are optimized for reliable data throughput and responsive user interaction rather than raw inference speed. The host machine also maintains the development environment, allowing for the deployment of software updates to the drone over the local network.

6.2.2 Network Core: TP-Link AC1900 Wi-Fi Router

To maintain a robust link between the operator and the S550 Hexacopter, the GCS employs a multi-layered communication architecture designed for redundancy and bandwidth optimization.

A local, private wireless network forms the communication backbone of our ground control station architecture, establishing the means by which the GCS can interact with many of the components on the drone. This network functions to handle the transmission of high-volume data, such as video streams and ROS messages, between the onboard Jetson and the GCS laptop. The core of this network is a TP-Link AC1900 (Archer A8) Wi-Fi Router, selected for its MU-MIMO technology and gigabit throughput capabilities. To ensure field portability independent of grid power, the router is energized by a high-capacity portable battery bank located at the ground station.

6.2.3 Network Extension: TP-Link EAP225 Omada Access Point

To accommodate the flight range of the S550 Hexacopter, the network range is augmented further by a TP-Link EAP225-Outdoor Omada AC1350 Access Point. The extender is connected directly to the primary router, and this extender also projects the 5GHz signal significantly further than standard consumer hardware, extending the effective high-bandwidth communication range to approximately 200 meters. This dedicated infrastructure isolates the mission-critical data streams from public network interference and ensures low-latency video transmission.

6.2.4 Command and Control Link: SiK Telemetry Radio

Parallel to the Wi-Fi network, a SiK Telemetry Radio establishes a dedicated 915MHz command and control (C2) link. This radio connects via USB to the GCS laptop and communicates directly with the Pixhawk flight controller on the S550. This channel is reserved exclusively for MAVLink flight commands and vital telemetry data. By decoupling flight control from the high-bandwidth data link, the system ensures that the operator maintains control over the aircraft even if the video feed or Wi-Fi connection is saturated or lost.

The primary safety mechanism for the system is the Radiolink AT9S Pro transmitter. This 10-channel handheld remote communicates independently with an R9DS receiver mounted on the drone. This subsystem operates completely outside the GCS software loop, so that in the event of an autonomous navigation failure or an emergency, the pilot can utilize the transmitter to instantly override the onboard agent and assume manual control of the hexacopter.

6.2.5 Power Management and Support Equipment

To sustain operations in the field, the GCS includes a specialized charging infrastructure for the S550's propulsion system. A Thunder Power TP610HVC LiPo battery charger is utilized to manage the 4S 4000mAh flight batteries. The charger is critical for maintaining cell equilibrium, as it performs precise voltage balancing across all four cells during the charge cycle. This maintenance is essential for preventing thermal runaway and ensuring that the high-current demands of the hexacopter motors are met safely and reliably throughout the battery's lifespan.

7. Software Design Documentation

Our four main software components are our agentic AI system, our React dashboard, our simulations, and ROS 2.

7.1 Agentic AI

This section details the software architecture, frameworks, and custom code that power our cohesive, intelligent, and autonomous system. We will break down how commands are understood, how the drone is controlled, and how data is managed.

7.1.1 Cognitive State Graph

The core of our architecture has evolved from a static script-generating approach to a dynamic, stateful reasoning engine built on LangGraph. Instead of writing a single Python script to execute a mission blindly, our agent operates within a ReAct (Reasoning + Acting) loop. This allows the AI to think step-by-step, execute a single action, observe the result (or error), and adapt its plan in real-time.

The system is defined as a state graph (StateGraph) with nodes labelled with the names “Agent” and “Tools”. The Agent node utilizes a large language model (LLM) via OpenRouter, configured with a custom system prompt that explains to the agent that it is a “drone pilot with computer vision”. The Tool node executes the actions requested by the agent. The flow of information is managed by a persistent AgentState, which tracks the conversation history (messages), the current status of the drone (drone_state), and a semantic map of the environment (world_memory). This state flows through the graph and grows after every tool call. This architecture allows the agent to maintain context over long missions, remember objects it has seen, and recover from failures intelligently.

Here is a simplified example of how the agent's state would update during a mission:

- **Initial state**
    - User Message: "Take off and move forward 2 meters."
    - System Message: Telemetry shows that the drone is on the ground.
- **State after calling "take_off" tool**
    - Agent Message: "I have successfully taken off."
    - System Message: Telemetry shows that the drone has taken off.
- **State after calling "move" tool**
    - Agent Message: "I have successfully taken off."
    - Agent Message: "I moved forward 2m."
    - Agent Message: "The mission is now complete."
    - System Message: Telemetry shows that the drone has moved forward.
- **Final state**
    - At this point, the agent would provide a mission summary which we would show to the user in the react dashboard.

7.1.2 Tool Abstraction Layer

The agent interacts with the physical world through a defined set of tools, implemented as subclasses of LangChain's BaseTool. These tools serve as the Application Programming Interface (API) for the agent, providing a clean abstraction over the complex hardware logic.

We adhere to the Facade design pattern through a DroneService class. This class encapsulates the low-level MAVSDK-Python logic, handling connection, arming, and telemetry. Agent tools such as MoveRelativeToBody, SenseObjects, or Takeoff are lightweight wrappers that call methods on this service. This separation enables strict typing and Pydantic validation of agent inputs, such as ensuring movement commands receive valid distances and directions, before any hardware commands are issued.

7.1.3 Native Asynchronous Architecture

A significant advancement in our current iteration is the move to a fully asynchronous, event-driven architecture. Unlike other versions that may require complex threading to bridge synchronous agents with asynchronous hardware libraries, our entire stack is built on Python's asyncio.

The LangGraph application runs via app.astream(), allowing the mission execution to yield events in real-time such as token streams, tool inputs, and tool outputs. The tools themselves define _arun (async run) methods, allowing them to await MAVSDK commands like drone_service.goto_location() without blocking the main event loop. This non-blocking design enables the system to maintain a responsive WebSocket connection with our React dashboard user interface even while the drone is in the middle of a long-duration flight maneuver.

7.1.4 Integrated Perception and World Memory

Perception is no longer a separate, disjointed process but an integrated capability of the pilot agent. We implemented a SenseObjectsTool that the agent can call whenever it needs to observe its surrounding environment.

When triggered, this tool queries our YOLO perception node to detect objects and calculate their precise 3D coordinates. The critical innovation here is the World Memory system. Instead of just returning raw detections, the system processes the data to maintain a persistent state of the environment. It assigns unique IDs to objects (e.g., "chair_1", "person_2") and stores their 3D locations in the global world_memory. This memory is injected back into the agent's state, allowing the LLM to perform complex semantic navigation commands like "Fly to the red chair I saw earlier" by referencing the stored coordinates in its memory.

7.2 React Dashboard

To facilitate real-time human interaction, control, and monitoring of our autonomous drone system, we developed a custom React-based dashboard. For implementation of the dashboard, we utilized Next.js since it is a robust React framework that supports server-side rendering and API routing. For styling, we adopted Tailwind CSS for its utility-first approach. For components, we leveraged 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 7.1* on the next page), with each part designed to present critical information or points of interaction in real time to the human operator. This structure provides a clear and intuitive display for the user experience, functioning to ensure the system remains transparent, responsive, and adaptable across a range of mission complexities. Additionally, this structure allowed for modularity and future scalability during our development process, with each component able to make progress independent of the others, and it also enabled rapid debugging, override control, and operator awareness through a cohesive and informative visual interface.

The four quadrants of the dashboard are:

- A **live drone video feed** in the top-left quadrant, showing the ZED 2i camera (live) or the Gazebo camera (virtual), with the option to switch cameras where applicable,
- A **digital twin display** in the top-right quadrant, showing the SLAM-generated three-dimensional point cloud, rendered visible and interactable through Three.js
- A **chat interface** in the bottom-left quadrant, where the human-in-the-loop operator can send commands to and receive updates from the agentic AI, and

- **System messages and telemetry** in the bottom-right quadrant, showing status updates, live drone details, and a manual override button

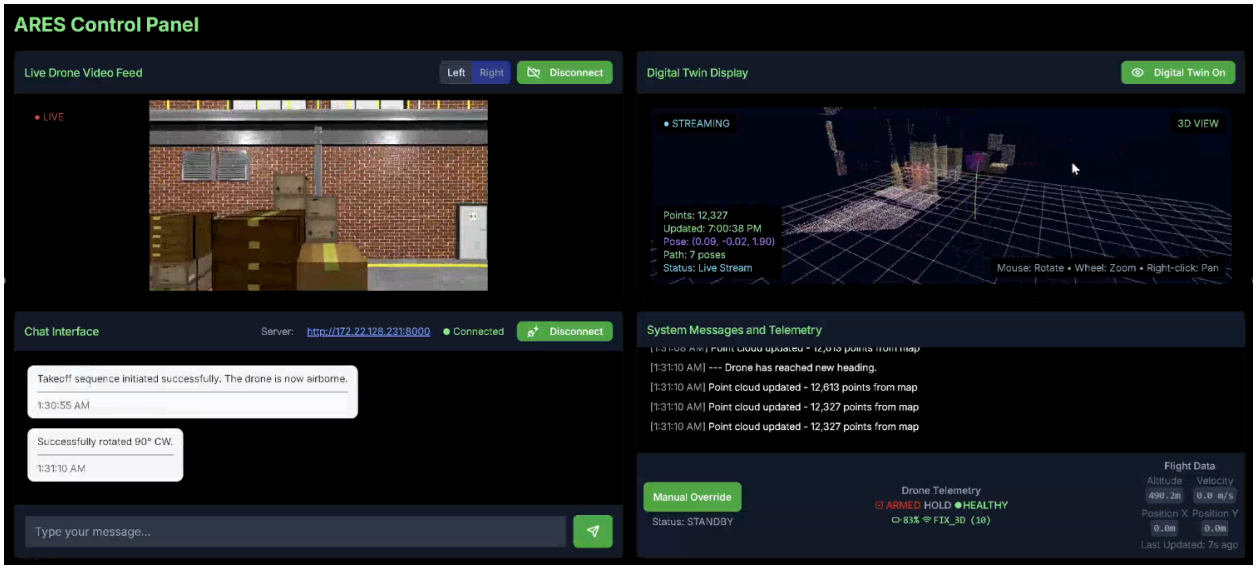

Figure 7.1 – React Dashboard *screenshot showing the four-quadrant layout*

7.2.1 Live Drone Video Feed

The top-left quadrant of the dashboard is dedicated to displaying a live video feed from the drone's onboard camera, which is 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. In the corner of this quadrant lies a connect/disconnect button that will initiate or terminate the connection to the drone. Additionally, there is the option to switch between the two halves of the ZED 2i stereo camera, allowing the human operator the choice of the left or right camera.

In the background, the video feed is implemented by ensuring the component connects to a video stream, which in turn is itself created by connecting to a ROS video server that is connected to the ZED 2i stereo camera.

7.2.2 Digital Twin Display

The top-right quadrant is reserved for displaying a digital twin of the drone's surrounding environment, which is dynamically constructed via a simultaneous localization and mapping (SLAM) algorithm. The digital twin in our dashboard is a three-dimensional point cloud obtained from ROS, akin to the video feed data. However, this spatial data requires additional parsing in order for it to be sent to Three.js to be interpreted correctly. Subsequently, Three.js renders the received data in the dashboard in order to be viewed and manipulated accordingly by the human operator, including camera rotation and zoom capabilities.

The dashboard is programmed to continuously poll and listen for new point cloud updates, allowing for near real-time updates to the virtual environment as the drone with its SLAM algorithm incrementally improves upon the digital twin.

Like the video feed component, the digital twin component will also include a toggle for initiating and terminating point-cloud generation.

7.2.3 Agentic AI Chat Interface

The bottom-left quadrant serves as the primary communication interface between the human operator and the agentic AI system, which is powered by a localized large language model running in a Python environment. The interface is designed as a chatbox, allowing for natural language dialogue between the human operator and the agentic AI.

When the human user enters a command in the chatbox, it will be sent to the agentic AI. The agent will then interpret the command, breaking it down into a series of actionable steps. The agent will then perform each of these steps in order, providing feedback to the user via the chatbox as to what action is being taken and what the result of that action is.

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. The connection is implemented via a WebSockets library, where the dashboard acts as a client and the agentic AI acts as a server.

7.2.4 System Messages and Telemetry

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

- Initialization of the dashboard
- Drone camera connection and disconnection events
- 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
- Changes and updates to drone telemetry

In addition to this message log, this quadrant also displays live telemetry data from the drone, such as current altitude and velocity, at the bottom of the quadrant.

Lastly, this quadrant features the manual override button, which allows the operator to instantly disable any ongoing missions from the agentic AI and prevent the agentic AI from issuing any further commands until the override is disengaged. The user will subsequently be able to take manual control of the drone via the physical controller. This safety mechanism serves as another mode of protection surrounding the agentic AI, providing human oversight over the autonomous system.

7.2.5 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**: The operator will connect and turn on the live video feed, the digital twin display, and the agentic AI chatbox.
2. **Human-AI Interactivity**: In the bottom-left quadrant, the operator can type in "Rotate around, and if you see a chair, orbit it" and submit the message to the agentic AI system.
3. **Autonomous Execution**: The agentic AI then interprets the command, translating it into the following actionable plan: "I will arm and take off. I will then check if my current field of vision contains a chair. If not, I will rotate 90 degrees and check for a chair again. I will continue rotating until I am back in the starting orientation." The drone will then begin to execute this plan in sequence as a series of steps.
4. **Manual Override**: The operator clicks the "Manual Override" button in the bottom-right quadrant while the drone is turning. The drone then halts in place and continues to hover, while the agentic AI system is paused and disabled from drone control.
5. **Mission Completion**: The operator disengages manual override by clicking the button again, then sends the command "Return home" to the agentic AI. The agentic AI then comes up with the plan to return to the starting position and land, and it executes the plan on the drone. The drone returns to the starting location and lands safely, upon which the operator exits the programming.

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

7.3 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 a 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 de facto 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, or sends, the simulated sensor data onto ROS 2 so-called “topics”, and the rest of our codebase subscribes to these topics for processing and decision making.

The drone is controlled via MAVSDK, a protocol for sending commands to drones. Gazebo is launched using PX4 which initializes a drone that is capable of following MAVSDK commands. Since our physical drone also uses MAVSDK, this allows us to use the same logic and code to control the virtual drone as the physical drone.

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, examine the drone’s planned trajectory, or view the output of our computer vision algorithms, which proved invaluable for debugging and verification.

7.3.1 Streaming Data to the Dashboard

The software running on the Jetson is lean and specialized. It acts as a high-performance data pipeline by 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 digital twin data.

Porting the live video feed to the dashboard proved facile.

Getting the live video feed to the dashboard was the easier of the two. Our code uses an existing ROS library called web_video_server which exposes all image data topics via HTTP streaming. On the ROS side, we added the web_video_server as a node to our startup script. In React, we created a component that would display the video streams depending on which topic (left/right) was selected, using an image tag with the IP of our linux machine, port of the web_video_server (8080) and the path was the ROS topic (left/right).

On the other hand, porting the digital twin data to our React dashboard required a more customized approach, requiring more code than any of our other major features. The digital twin we show on the dashboard is created from multiple topics in ROS. It uses the point cloud from SLAM, the odometry history from the robot, the drones current pose, and object coordinates from the YOLO perception node. We created a custom ROS node to efficiently relay these to the React dashboard via WebSockets, and it intelligently updates by using a debounce and only sending new data when changes to any of the topics occurred. The dashboard receives this data by connecting to the WebSocket exposed by this ROS node. Finally, the dashboard renders the data using Three.js components after transforming it into the data types Three.js requires to operate.

7.3.2 Perception in the Digital Twin

The drone is equipped with a SenseObjects tool to determine the types and locations of objects currently within its range of view. This tool used our YOLO perception node to get the three-dimensional coordinates of objects YOLO recognized in the video feed. The object names, IDs, and coordinates are then injected into the agent's prompts, which let the model act based on the world around it. An example of the data the drone has access to:

- *Your Position: (1, 0, 3)*
- *Your Yaw: 12 degrees*
- *Objects In World*
    - *Id: 1, Type: Chair, Position: (3, 0, 2)*
    - *Id: 2, Type: Person, Position: (-3, 2, 5)*
    - *Id: 3, Type: Person, Position: (5, 3, 4)*

The YOLO perception node operates by fusing data from the video feed and the point cloud from SLAM. First, it sends the video feed to our YOLO vision model to get the bounding boxes of objects it identified within the video feed. Then, it computes the centroid of the object and calculates the angle it would need to project a ray to hit the object in the digital twin. The centroid/angle algorithm works by shooting out multiple rays towards the digital twin version of the object and taking the point closest to the center out of the points of contact between the rays and the objects. This gives us a coordinate relative to the center of the map, which is useful because it does not change throughout the mission.

7.3.3 Gazebo Environment

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

This workflow was crucial for de-risking the project, allowing 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.

Gazebo features strong integration with ROS and ROS 2. As a result, drones in Gazebo exist as fully virtualized entities that are able to communicate with standard ROS 2 topics, allowing direct connection between our virtualized environment and the software on our ground control station, such as our React dashboard and our agentic AI. Our simulated drone in Gazebo is a multicopter equipped with two cameras, akin to our real-life hexacopter equipped with the ZED 2i stereo camera.

Furthermore, we created several virtual worlds in Gazebo for our simulated end-to-end testing and validation. These worlds included digital warehouse objects, chairs, people, shapes, and other items to test the robustness of our digital twin point cloud modeling and our YOLO perception node for object identification.

7.3.4 Limitations To Gazebo

The transition from a simulated environment 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 how a drone is able to move and how its internal systems function are simplified representations of reality.
- **Sensor Discrepancies**: Real-world sensors are susceptible to noise, calibration errors, and environmental factors like changing light conditions, which are often idealized in simulations. Even simulators that can mimic some variances cannot fully replicate all of these potential discrepancies.
- **Environmental Dynamics**: Factors such as wind, weather, and GPS signal dropouts introduce variability that is challenging to model with complete accuracy in a simulated environment.
- **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.

As a result of these factors, part of our design had to account for how we would migrate appropriately from our Gazebo simulation to real-world field testing, as even validated end-to-end virtual tests were susceptible to not replicating as effectively in live conditions.

7.3.5 Staged Transition to Real-World Flight

To safely bridge this gap between simulation and reality, our project adopted a methodical, multi-stage approach. This process involves progressively introducing real-world components and complexities while validating system performance at each step.

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. **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.

In this project, the majority of our work was done on pure simulation. Towards the end of the project we skipped HIL simulation and moved directly to Controlled Real-World Testing at a park nearby. We were not successful enough with our controlled simulations to move to more realistic operating environments like near buildings on farms.

7.3.6 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 a collision is imminent.
- **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

When we moved to our controlled environment, we implemented many of these, including pre-flight checks and loss of communication protocols. However, we incorrectly assumed that at any moment if the joysticks were moved, control would be handed off from the agent to the radio controller; this caused a scary moment where the drone was in the air and we were unable to send it commands. It eventually crashed on its own, only damaging a few propellers. This served as a lesson to not make assumptions when it comes to safety.

7.4 ROS 2

7.4.1 ROS Nodes

Our architecture leverages a distributed network of ROS 2 nodes, with each node responsible for a specific, discrete task. This separation of concerns is critical for building a complex, maintainable system.

There are many nodes:

- **Camera Node**: A node utilizing the zed_ros2_wrapper package interfaces directly with the ZED 2i camera. It publishes synchronized RGB images, depth maps, and camera calibration data to dedicated ROS 2 topics (/zed/rgb/image_raw, /zed/depth/depth_registered)
- **web_video_server**: This is an existing library built for relaying image topics over HTTP, which we use to transmit our video feed to the React dashboard.
- **Ros_gz_bridge**: This node only runs during simulation. This node is a bridge that turns Gazebo topics into ROS topics. We use it in our virtualized drone-agentic system for the Gazebo clock, odometry data, and image data.
- **SLAM Node**: This node subscribes to the image and depth topics to generate a real-time point cloud of the environment, publishing the resulting map to a /cloud_map topic.
- **Digital Twin Bridge**: This is a custom-built node that, via WebSockets, relays the digital twin from RTAB map, the drone's position, any identified world objects, and the drone's historical movements to the React dashboard so that it can be visualized by the user.
- **YOLO Perception Node**: This subscribes to the video feed, cloud map, and tf tree to allow the agent to sense objects within the drone's view.

This ROS-2-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 ROS 2 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, as adding new sensors or computational nodes is extremely facile.

7.4.2 RTAB-Map

Although it was difficult to set up, RTAB-Map was eventually implemented into our project. RTAB-Map is our project's choice of algorithm for simultaneous localization and mapping (SLAM), and it powers the digital twin that is visible in the React dashboard. In order to render RTAB-Map operational, our system publishes a multitude of TFs that tells the algorithm how the different sensors are physically connected to the drone.

In addition, our system bridges between differing coordinate systems for Gazebo and for ROS 2, which we were able to accomplish with additional image processing nodes. We also ensure that the launch file works cooperatively with our ZED 2i stereo camera, since the ZED 2i natively publishes the video feed of both cameras, a depth topic, and the TFs needed for SLAM.

Our system predominantly uses the default options for RTAB-Map. However, there are two notable configurations we have implemented:

- "**approx_sync**" to allow some of the sensor data to be used even with slightly different timestamps
- "**delete_db_on_start**" so that RTAB-Map begins with an empty map every time

8. Logistics

8.1 High-Level Planning

8.1.1 Build and Design

The first phase of our project consisted of designing the build for our project in addition to researching the numerous individual components that would eventually comprise our overall autonomous drone system. In our initial architecture drafts, we planned to have the following three main subsystems (see *Figure 8.2* on the following page for our old version of a block diagram):

- A Jetson Orin Nano would be located on the ground and perform the majority of computational lifting, including our large language model, our vision language model, and our simultaneous localization and mapping module. In addition, we planned to use Ollama to host both language models locally.
- Our first choice of drone platform was the Holybro X500 V2 (see *Figure 8.1* on the right), along with a Pixhawk, a stereo camera, a SiK radio, a LiPo battery, and motors. Importantly, there was also an onboard Raspberry Pi for calculating camera depth data as well as point cloud computation.

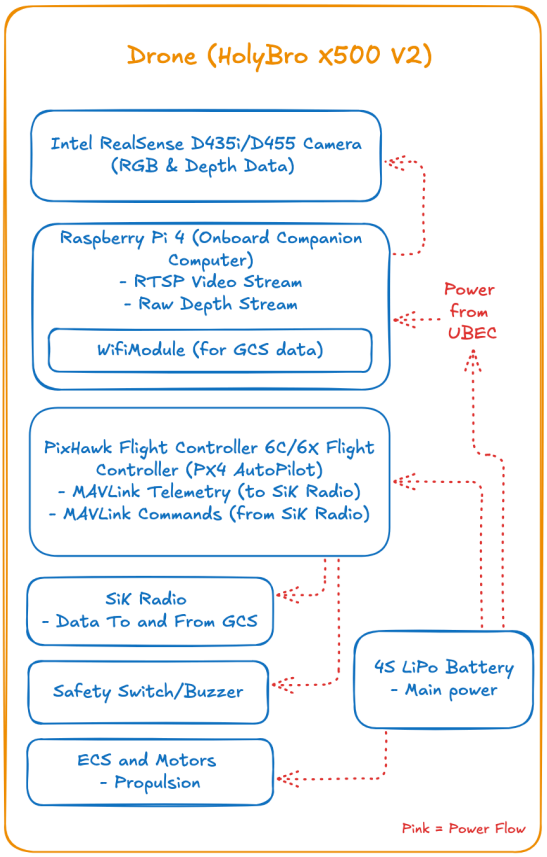

Figure 8.1 – Initial Drone Setup

- A laptop would house our React dashboard and our agentic AI framework. Also, the dashboard would receive video feed, telemetry, and point cloud data from the drone, displayed in a four-quadrant layout.

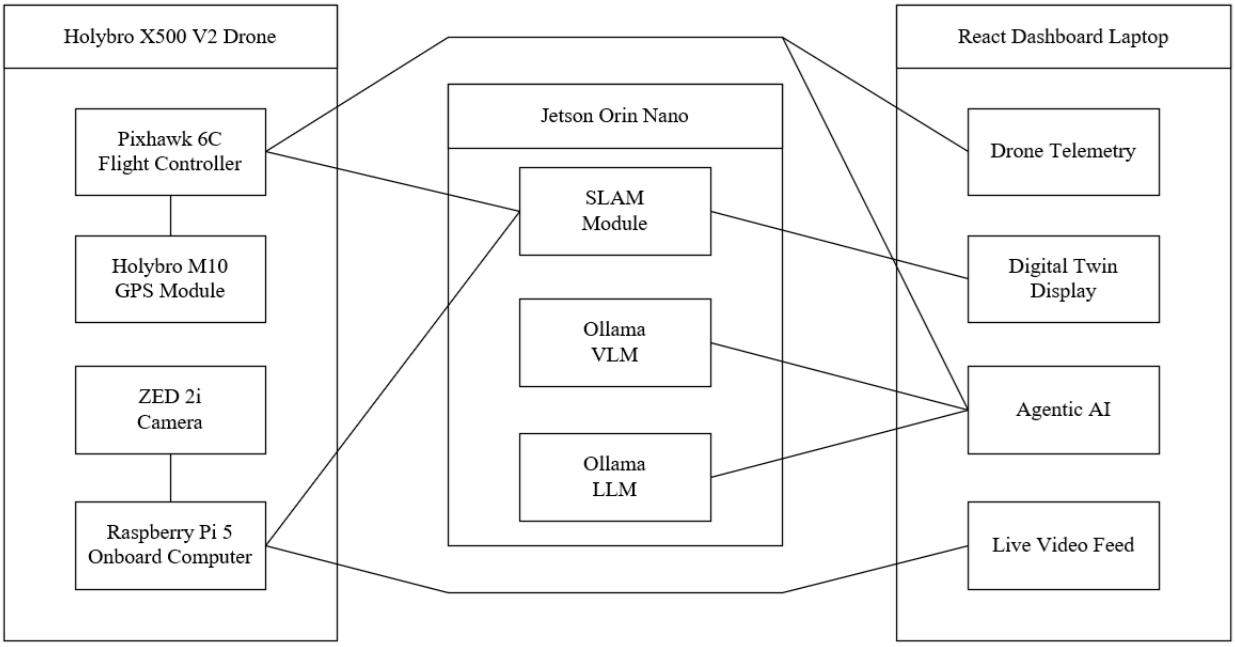

Figure 8.2 – Initial Block Diagram *showing our old three-part architecture*

During this build and design phase, we spent a significant amount of time researching each of the components in these subsystems, picking our presumed ideal candidates after individual research, group discussion, and final analyses.

In addition, we settled on creating a virtualized drone environment in order to accelerate software prototyping in parallel to waiting for the purchase and construction of our actual physical drone. We decided to use Gazebo, the industry standard for drone virtualization, in conjunction with the Robot Operating System (ROS) for data communication.

8.1.2 Prototyping Components

After synthesizing our build plan, we proceeded to begin to create prototypes of the various components of our project. Below is a brief summary of the prototypes we made that led to direct changes to our build design:

- Our agentic AI framework began with SmolAgents, but due to limitations in scope and robustness, we swapped to LangGraph.
- We discovered that the Raspberry Pi was unable to process the ZED 2i depth data due to it lacking a GPU, so we had to replace it with a Jetson.
- Subsequently, the Jetson Orin Nano on the ground was shifted to being an aerial Jetson on board the drone.
- Ollama VLMs proved too slow and inaccurate for real-time use, so we pivoted to using the YOLO vision model instead.
- Ollama LLMs also proved to be slow and less than ideal, so we utilized the online Gemini 2.5 Flash-Lite model instead.
- We initially selected the Intel RealSense D455 camera to pair with our drone, but after additional research, we switched over to the ZED 2i stereo camera due to superior range and depth processing results.
- We placed an order for the Holybro X500 V2, but due to supply shortages and elongated order processes, we were unable to acquire it. As a result, we went with a S550 hexacopter for our drone platform instead.
- For similar reasons, we had to move from the M10 GPS module to the M8N version.

Nonetheless, many prototypes and acquisitions remained unchanged throughout this process:

- The React dashboard retained its original four-quadrant layout, with the prototype version being very similar in appearance to the final version; changes

over time were primarily in regards to increasing dashboard functionality and debugging communication protocols.

- The Gazebo simulation environment proved highly suitable for agentic AI prototyping, and we continued to use the same virtualization throughout the entirety of the project.
- The Pixhawk flight controller was the same between our initial and final drone platforms, albeit with a minor difference in software version.
- The SiK radio that we initially selected was acquired and stayed the same.
- The LiPo battery that we initially chose was purchased and remained the same as well.

Our final architecture and block diagram can be found in section 5.1 of this document.

Throughout this prototyping stage, we continuously sought feedback from our sponsor and made the requisite changes based on that feedback.

8.1.3 Field Testing

As we progressed with our integration testing and our drone construction, we transitioned into our final phase of live drone testing. We validated our full end-to-end testing in the virtual environment, ensuring that our workflows operated correctly between the simulated Gazebo drone and our ground control station.

At this point, we began to 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 worked together in concert. We worked our way through testing drone subsystems with the React dashboard one at a time – the drone video feed, the point cloud generation, and the agentic AI – all initially without propellers on the drone, for safety purposes. Once we completed these integration tests with the actual drone, we moved to field testing with the full system.

8.2 Jira Artifacts

Throughout the project, we used Jira as our primary Agile planning and tracking tool. All major tasks, experiments, and integration efforts were represented as epics, stories, and tasks, grouped into thematic releases such as navigation, modeling, autonomy, dashboard, hardware, and deliverables. This structure allowed us to trace work from high-level goals down to individual implementation tasks and to coordinate cross-cutting efforts across hardware, software, and research domains. The subsections below summarize our milestones, sprint progress, and representative story themes based on these Jira artifacts.

8.2.1 Milestones

Our development timeline spanned from early July through late November and was organized around a series of monthly milestones aligned with Serco and course deliverables. Each milestone aggregated Jira stories across multiple epics and represented a major integration step toward the Minimum Viable Product (MVP).

- **July – Foundational Architecture, Simulation, and Hardware Planning**
  Early work focused on validating basic feasibility. Milestones included selecting the drone platform and airframe, evaluating potential point-cloud and mapping technologies, experimenting with Gazebo/Unity for simulation, and designing the initial system architecture. Our team worked on conducting airframe selection, flight-controller research, and the first navigation stack experiments.

- **August – Hardware Acquisition and Early Perception Prototypes**
  In August, milestones centered on acquiring and assembling key hardware: the S550 frame, Pixhawk flight controller, ZED 2i stereo camera, Jetson Orin Nano, telemetry radios, and GCS network components. In parallel, we began

photogrammetry and visualization research, testing RViz versus Unity, and building preliminary pipelines for converting image data into point clouds.

- **September – Agentic AI and Tooling Integration**
  September milestones shifted toward autonomy and intelligent control. Work focused on building the first SmolAgent and LangGraph agents, defining a tool abstraction for drone control, and integrating vision outputs with the agent's reasoning loop. We also brought up initial code-generating tools, reactive safety checks, and first prototypes of simulated agent-driven missions.

- **October – Real-Time Dashboard and End-to-End Data Flow**
  In October, the emphasis moved to the React dashboard and real-time data flows. Milestones included initial integration of the video feed, telemetry overlays, digital twin rendering, and the LangGraph agent chat interface. By the end of this phase, we could stream data from the drone or simulation through ROS 2 into the GCS and visualize it live on the dashboard, closing the loop between perception, planning, and operator feedback.

- **November – Outdoor Testing, Snapshot System, and Final Deliverables**
  The final milestones focused on real-world tests, reliability improvements, and polishing deliverables. We assembled and flew the physical drone, validated basic autonomous behaviors outdoors, and deployed the snapshot-saving pipeline for point-cloud reconstructions within the dashboard. Concurrently, our team worked on documentation, presentations to Serco and the university, final design report preparation, slide decks, and STEM showcase materials. These efforts culminated in a fully-integrated final demonstration that highlighted both the system's capabilities and its remaining challenges.

8.2.2 Epic Summary

At a high level, Jira epics captured the concrete tasks necessary to realize the system. While there were many individual items, they fell into several recurring themes aligned with the LAD releases:

- **LAD-1 Drone Navigation & Simulation**
  Stories in this group covered simulation setup, navigation stack configuration, and early control experiments. Examples included configuring PX4 with Gazebo, evaluating basic waypoints, implementing a “move forward one meter” test from the dashboard, and building the foundations for a reusable navigation stack.

- **LAD-2 Intelligent Object Detection and Modeling**
  These stories focused on photogrammetry research, RViz versus Unity comparison, and tools to compute object positions using depth. Tasks included experimenting with different point-cloud workflows, researching how to stream ROS 2 topics into visualization tools, and prototyping methods to compute the 3D location of objects detected in the camera’s field of view.

- **LAD-3 Agentic AI & Autonomy**
  This release grouped stories related to the AI planner and tools. Representative tasks included building SmolAgent prototypes, acquiring and setting up the Jetson, writing the VLM agent/tool, integrating wireless telemetry into the agent context, implementing a LangGraph-based DronePilot, constructing a multi-step mission graph, and developing a reactive safety tool to prevent unsafe movements.

- **LAD-4 Real-Time Monitoring Dashboard**
  Stories here targeted the React UI and its integration with backend services.

Items included initial dashboard integration with the agentic AI, wiring up the video feed, displaying telemetry overlays, rendering the digital twin, and ensuring that LangGraph messages and system logs appeared correctly in the frontend.

- **LAD-5 Core System Infrastructure & Hardware Integration**
  LAD-5 encompassed physical and infrastructure tasks: exploring point-cloud technology options, selecting and ordering the airframe, choosing compute and flight-control components, purchasing and assembling drone and GCS hardware, configuring telemetry links, powering the Jetson on the drone, creating custom 3D-printed mounts, and setting up the GCS network and router.

- **LAD-42 Senior Design and Serco Deliverables**
  These stories tracked all major academic and sponsor deliverables, including planning and writing the Preliminary Design Document, Critical Design Review, Final Design Document, Serco presentations, summary submissions to CECS, and producing the STEM showcase video. Together, they ensured that the technical work was communicated clearly and met all course and sponsor expectations.

Overall, the Jira artifacts provided a complete trace from high-level objectives and milestones down to specific implementation tasks. This structure helped the team stay organized across hardware, software, and AI research, while providing clear evidence of progress throughout the project lifecycle.

8.2.3 Sprint Summary

Our work was organized into a series of sprints running roughly every two weeks from early July through late November. Below is a lower-level summary of ten representative sprints, capturing how the project evolved from research and planning into full hardware integration and outdoor testing.

- **Sprint 1 – System Vision and Technology Survey**
  This sprint focused on finalizing the problem statement, identifying core subsystems, and surveying candidate technologies. We compared simulation tools (Gazebo vs. Unity), evaluated mapping options such as RTAB-Map and photogrammetry tools, and drafted the earliest block diagrams. The outcome was a shared understanding of the planned architecture and a concrete list of hardware and software components to prototype.

- **Sprint 2 – Navigation & Simulation Prototypes**
  Sprint 2 built on the initial research by targeting navigation and simulation validation. We brought up a PX4-based simulation and explored the navigation stack. These efforts established a baseline for later autonomous mission execution and helped expose early issues in motion control.

- **Sprint 3 – Photogrammetry and Mapping Research**
  During Sprint 3, the team focused heavily on mapping and visualization. We compared RViz with Unity for rendering point clouds, tested COLMAP-like workflows for offline reconstruction, and explored how RViz could be used for live data rendering. Although RViz was ultimately deprioritized in favor of a web-based pipeline, this sprint clarified trade-offs in tooling and informed the design of our eventual React-based digital twin.

- **Sprint 4 – Hardware Acquisition and Core Infrastructure**

  Sprint 4 was dominated by hardware tasks. We finalized and ordered the drone airframe, flight controller, ZED 2i camera, telemetry radios, batteries, and GCS components. The TI lab and Senior Design Lab were used extensively for soldering power components, designing and 3D printing mounts, and assembling the hexacopter frame, nearly completing our hardware stack.

- **Sprint 5 – Jetson, ZED 2i, and ROS 2 Integration**

  In Sprint 5, the team focused on bringing the Jetson and ZED 2i to a ROS 2 environment. We installed drivers, configured the ZED ROS 2 wrapper, and validated that the camera could stream RGB and depth topics into the GCS. Initial tests of SLAM and point-cloud generation were conducted, verifying that the data pathway from sensor to GCS was viable for real-time processing.

- **Sprint 6 – Agentic AI & Tooling**

  Sprint 6 concentrated on the autonomy pipeline. We developed the first working agent tools, integrated them with ROS 2 topics and MAVSDK commands, and experimented with multi-step plans. Stories in this sprint included building a LangGraph-based DronePilot agent and ensuring the agent could reason using telemetry and vision inputs before issuing movement commands.

- **Sprint 7 – React Dashboard: Video, Telemetry, and Digital Twin**

  In Sprint 7, we matured the React dashboard into a fully interactive control surface. We integrated the live video feed, overlaid telemetry, and linked a 3D digital twin viewer to the incoming point cloud data. We also connected the agent chat interface so that natural-language commands issued through the UI could trigger agent workflows and reflect reasoning and status back to the user.

- **Sprint 8 – End-to-End Integration and Simulation Runs**
  Sprint 8 focused on connecting all major components in simulation. The team ran full end-to-end tests where a user command flowed from the dashboard to the agent, into ROS 2 and MAVLink, and ultimately into simulated drone motion, while sensor topics streamed back into the digital twin. This sprint surfaced synchronization issues, timing glitches, and data-format inconsistencies, which were addressed through tighter API contracts and monitoring tools.

- **Sprint 9 – Outdoor Testing, Tuning, and Snapshot Feature**
  Sprint 9 represented the transition from simulation to live flights at Blanchard Park. We tuned PID parameters, validated basic autonomous behaviors outdoors, and instrumented the system for real-world latency and packet-loss conditions. In parallel, we implemented the IndexedDB-backed snapshot feature, allowing operators to save and reload point-cloud reconstructions directly from the dashboard. Partial autonomous runs were achieved, though instability in the vision pipeline and hardware wear began to appear toward the end of testing.

- **Sprint 10 – Stabilization, Documentation, and Final Deliverables**
  The final sprint focused on refining reliability, documenting results, and preparing external deliverables. We consolidated Jira issues, performed regression tests on the latest integrated build, and captured remaining bugs or technical debt as future-work items. The team also completed the Final Design Document, Serco leadership presentation, committee slides, and the STEM showcase video, ensuring the project's outcomes were clearly communicated to both technical and non-technical audiences.

8.3 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 expenditure was $1,560.16 (not including shipping), keeping us comfortably under budget. We achieved this through a combination of online purchasing for specific components and securing critical, high-value hardware like the NVIDIA Jetson, Raspberry Pi 5, and a professional-grade battery charger through labs at the University of Central Florida as well as 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. (see *Figure 8.1* below).

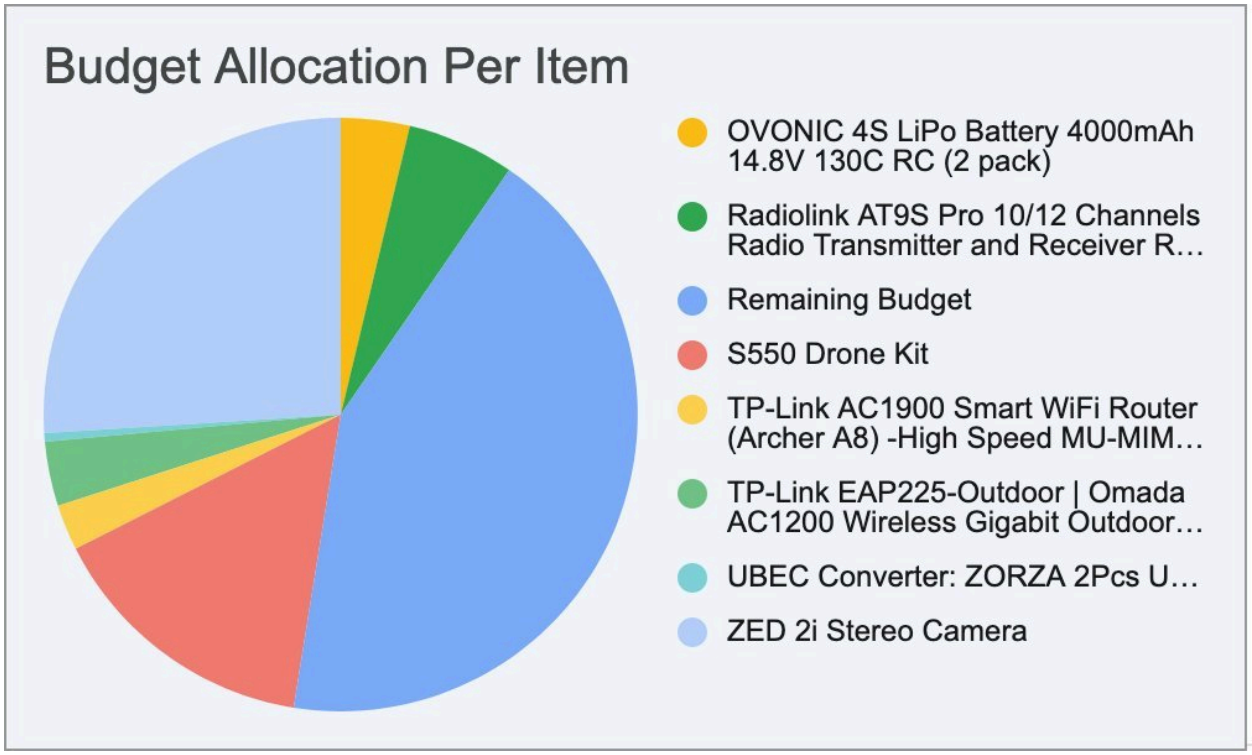

Figure 8.1 – Budget Allocation shown in a pie chart format

The following table (*Figure 8.2*) details all of the expenditures for our project in itemized form:

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 8.2 – Purchasing List Table

9. Conclusions

9.1 Summary of Results

The ARES Autonomous Drone project successfully demonstrates how agentic AI can translate high-level human intent into precise robotic action. Our advanced robotics system successfully increases drone accessibility for the everyday user and provides an extensible foundation for future human-robot interaction research and real-world autonomous deployment.

Our agentic AI combines perception from a YOLOv8n vision model, reasoning from a Gemini 2.5 Flash-Lite large language model, and orchestration from a LangGraph framework, and our system proves capable of executing complex missions expressed through natural language.

Our React dashboard facilitates human-in-the-loop control of our system, allowing the user to view live drone camera feed, manipulate a SLAM-generated three-dimensional point cloud rendered via Three.js, communicate to the AI agent through a chat window, and monitor system updates with drone telemetry.

Our S550 hexacopter is well-endowed with a ZED 2i stereo camera for visual perception, a SiK Radio for telemetry, a Pixhawk for flight control, an M8N GPS module for navigation alongside simultaneous localization and mapping, and an onboard Jetson Orin Nano for the computation, analysis, and streaming of RGB video and depth data.

Critically, our sponsor is highly satisfied with our Project LION-ARES product, stating “I could not be happier” at our final presentation in front of a committee of several University of Central Florida computer science professors.

9.2 Current Status

9.2.1 Performance Analysis

The performance of the LION Autonomous Drone System was evaluated through a series of integrated tests that ranged from simulation to outdoor field trials at Blanchard Park. Early simulation runs demonstrated that the end-to-end architecture functioned as intended: natural-language commands issued from the React dashboard were interpreted by the agentic AI, converted into MAVLink commands, and executed by the virtual drone while telemetry and simulated sensor data streamed back into the digital twin. These tests validated the core control loop, including command parsing, agent reasoning, and feedback visualization.

During physical testing, we achieved several partial successes. The assembled S550 hexacopter, equipped with the ZED 2i stereo camera and Jetson Orin Nano, successfully executed basic autonomous maneuvers under agent supervision. The system was able to stream live RGB and depth data from the ZED through ROS 2 to the GCS, generate point-cloud reconstructions, and display them in the dashboard. The IndexedDB-backed snapshot feature worked as designed, allowing us to save reconstructions mid-mission and reload them later for analysis. In these initial flights, the point clouds were accurate enough to capture key structures in the environment, and the agent's control commands mapped correctly to real-world motion.

However, the outdoor tests also revealed important limitations. During later trials, the drone experienced a crash that physically damaged the camera and likely affected the IMU calibration and mounting alignment. After this event, point-cloud quality degraded noticeably: reconstructions became distorted, drift increased, and SLAM outputs exhibited inconsistent alignment with the true environment. The live video feed also became less stable and more prone to lag, reducing the operator's confidence in the

visual and spatial information being displayed. These issues highlighted the sensitivity of the perception stack to mechanical integrity and sensor calibration, especially in an outdoor environment with variable lighting and wind.

From a reliability perspective, the system demonstrated that the architecture is sound but still fragile under real-world conditions. The software stack could recover from minor communication hiccups, but hardware failures and sensor degradation had a direct and substantial impact on autonomy quality. These results underscore the importance of robust mounting, vibration isolation, redundancy, and post-impact recalibration procedures for future teams. Despite the setbacks, the tests confirm that the system can perform the intended end-to-end workflow; improving robustness and repeatability will be the main focus of future work.

9.2.2 Lessons Learned and Insights Gained

Throughout the lifecycle of the LION Autonomous Drone project, our team encountered numerous technical and logistical challenges that provided valuable engineering insights. These lessons extend beyond mere code and hardware. They included safety protocols, architectural trade-offs, and the realities of deploying AI in the physical world.

- **The Reality of the Sim-to-Real Gap**: The most critical lesson learned was that success in a simulation does not guarantee success in the real world. Our reliance on Gazebo for developing the agentic control loop led to implicit assumptions about hardware responsiveness. We assumed that calling the same MAVSDK commands for the Gazebo drone and our physical drone would cause the exact same behavior, which proved false. For example, the take off command in Gazebo functions perfectly; the drone takes off and hovers. But for our physical drone, due to our overweight drone and sensor data, the drone took off and then slowly descended and crashed. Additionally, we assumed that

moving the manual control sticks would instantly override the autonomous agent, a logic that functioned perfectly in software but failed during a specific high-latency event in the field. This resulted in a loss of control and a subsequent crash.

- **Hardware Insights**: We learned the importance of adaptability regarding supply chain constraints. Our initial design relied on the Holybro X500 V2, a "gold standard" research platform. When global shortages made this unattainable, the team was forced to pivot to the S550 Hexacopter. While this required significantly more manual labor, including soldering power distribution boards and fabricating custom mounts, it ultimately taught us more about the electrical and mechanical intricacies of drone construction than a pre-assembled kit would have. We learned that engineering is often about achieving specifications within constraints, and the cheaper, older S550 platform successfully met all our payload and flight requirements while saving budget. We also learned that things break, so it's best to plan for it by purchasing extra parts ahead of time; we ran out of propellers and caps for the propellers during several of our flight tests, postponing our testing schedule. If we had purchased those parts ahead of time, we could have saved valuable time and iterated faster.
- **Latency vs. Privacy in AI Models**: A major technical insight was the trade-off between data sovereignty and system latency. Our initial objective was to run all large language models (LLMs) and vision language models (VLMs) locally using Ollama to ensure total offline privacy. However, we discovered that local inference on consumer hardware introduced unacceptable latency (up to 8 seconds per query), which paralyzed the drone’s decision-making loop. Pivoting to the online Gemini 2.5 Flash-Lite model and the optimized, local YOLOv8n network proved that real-time responsiveness is paramount for autonomous robotics. This taught us that while total localization is an ideal goal, hybrid architectures are often necessary to make a system operationally viable.

9.2.3 Currently Operational Components

As of the conclusion of this project, the following LION-ARES components are working:

Aerial Platform: The S550 Hexacopter is fully assembled and flight-worthy. It is integrated with the Pixhawk flight controller, REC propulsion system, and the dedicated power distribution rail for an onboard computer. Despite previous impact damage, the airframe remains structurally sound for flight. However, it will be partially deconstructed due to having to return some of the parts, requiring some future reassembly.

Perception & Connectivity: The system successfully streams high-bandwidth RGB video and depth data via Wi-Fi to the Ground Control Station using ROS 2. The YOLOv8n object detection pipeline is active and correctly identifies target objects (e.g., chairs, people) in real-time. Unfortunately, the IMU in the ZED 2i is likely damaged, necessitating a probable replacement.

Ground Control Station (Dashboard): The React-based dashboard is fully functional. It successfully renders the live video feed, displays real-time telemetry (battery, altitude, mode), and visualizes the 3D digital twin generated by the SLAM algorithm. The "Snapshot" feature allows users to save point clouds to IndexedDB for later review.

Agentic Autonomy: The LangGraph-based AI agent is capable of receiving natural language commands via the dashboard chat interface. It can successfully parse instructions, generate multi-step flight plans, and execute basic maneuvers.

ROS 2 Workspace: The unified ROS 2 workspace serves as the system's communication backbone and is fully operational. The build system correctly compiles and links all custom packages, including the perception_node for object detection, the digital_twin_bridge for websocket communication, and the ros_gz_bridge for simulation handling. The workspace successfully streams high-bandwidth video from the drone to the dashboard without significant packet loss.

9.3 Future Directions

There were several features and stretch goals that our team looked into implementing for our project but were unable to accomplish at the current time. We are hopeful that future teams may be able to complete some of these items in subsequent continuations of our autonomous drone system.

9.3.1 Agentic Plan Approval Before Implementation

The current human-agentic-drone process involves the human sending a prompt to the AI agent, then the agent formulating a plan, and lastly the agent executing that plan stepwise through controlling drone movement. In an improved workflow, after the agent formulates its plan (or updates its plan at each step), the agent would then respond to the human via the React dashboard chatbox with its proposed plan *before* proceeding to execute that plan. At this point, the operator would be able to 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.

In addition, there would be a toggle on the dashboard that allows for automatic plan approval, in which case the process will operate identically to as it does at the current time. This approval-before-implementation mechanism would thus provide support for both supervised and autonomous operation modes for the agentic drone system.

9.3.2 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 some vision model or vision language model.
3. The vision module would then use object detection to tag objects in either the visual feed or the point cloud data.
4. There would be some mechanism that finally filters or removes those objects from the data, either pre-processing (by ignoring object RGB and depth data, before the point cloud is built) or post-processing (by removing the point cloud data points for the object, after the point cloud generation 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.

9.3.3 Model Context Protocol

Implementing tools through the Model Context Protocol has the potential to expand the range of applications and use cases that the agentic AI system can perform. Foremost, this could simplify multi-agent behavior by centralizing tools to a single server source that can be called by various different agents. Additionally, dividing tools from the LangGraph Python code allows for modularity in development and separation of concerns. Lastly, new tools can be created to interface with other data sources and APIs.

9.3.4 Temporal Change Detection

One of our original drone use case ideas was the implementation of a difference algorithm that would detect environmental changes across multiple scans of the same area. Below is a sample pipeline that future teams could potentially use for inspiration:

1. **First pass**: The drone scans the environment and creates the digital twin, as per our minimum viable product.
2. **Second pass**: The drone later repeats a scan of the same area.
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, increasing the potential range of applications and use cases for autonomous drones.

9.3.5 Safety Measures

One area we hope future teams can implement effectively is geofencing, the restriction of a drone to a predefined area or zone. Providing hard boundaries for drone movement would reduce the risk of unanticipated drone behavior or loss of flight control.

Additionally, we hope future teams are able to implement collision detection and avoidance. Our digital twin and point cloud data accurately models the boundaries of objects detected within the drone's field of vision, and so preventing collisions would require additional code that ports over our point cloud data to another agent or node within our LangGraph framework to terminate drone movement.

9.3.6 Dockerization

Although we were able to successfully keep track of version control and required dependencies without utilizing Docker in the course of our project, Dockerization would increase portability and accessibility for the project to other users and future developers.

9.3.7 Exploration Algorithms

Although our agentic AI system is able to use tools to explore the surrounding environment autonomously with new, independent thought processes, exploration algorithms remain strong contenders for helping navigate unknown terrain, and they could even be implemented as tool calls for the AI agent to use and manipulate on its own accord. Implementing an exploration algorithm would help unlock true autonomy and open the door to applications such as long-duration mapping, multi-flight environmental monitoring, and fully automated search-and-rescue operations.

Future teams can benchmark the various approaches to exploration algorithms within the virtual Gazebo environment to determine which best supports the current digital twin three-dimensional mapping pipeline. Additionally, future teams will need to evaluate these algorithms against hardware constraints and ROS 2 compatibility.

We anticipate that a 3D planner such as Fast-Planner or EGO-Planner will work well with our project. These planners use volumetric data structures like OctoMaps to represent the world, and they integrate exploration directly into their motion planning, generating dynamically feasible, smooth trajectories that are safe for the drone to execute. However, their primary drawback is complexity, since 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 available hardware components.

10. Acknowledgments

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

10.1 Sponsors

This project was generously sponsored by Serco, whose support enabled our team to explore advanced drone autonomy, intelligent agent architectures, and real-time sensing technologies in a real-world engineering context. Serco provided approximately $2,000 in funding, which made it possible to acquire essential hardware, including the S550 hexacopter frame, Pixhawk flight controller, ZED 2i stereo camera, Jetson Orin Nano, telemetry radios, batteries, and related components necessary for assembly, testing, and mission execution.

We extend our sincere gratitude to Maddy Braganca, Brian Babukovic, Blake Bergstrom, and Chris Griffith for their consistent support throughout the semester. Their willingness to meet with us, review our system architecture, and provide technical and practical feedback helped shape the project's direction and ensured our goals remained grounded in realistic operational requirements. Although much of the development relied on experimentation and self-guided problem solving, the expertise and encouragement provided by Serco mentors offered invaluable perspective and kept the team motivated through complex integration challenges. Their involvement greatly enriched our experience and contributed directly to the scope and ambition of this project.

10.2 Coordinators

We would also like to thank our IDesign coordinators, Matthew Gerber and Richard Leinecker, for their guidance, support, and feedback throughout Senior Design I and II. Their expectations for documentation, design reviews, and system clarity helped us maintain high standards in both technical execution and communication. We further appreciate the CECS faculty and staff who supported the course infrastructure, maintained the laboratories used for development, and ensured we had access to the tools and resources necessary to complete a multidisciplinary project of this scale.

10.3 Facilities and Equipment

This project relied heavily on a diverse set of facilities and equipment provided by the University of Central Florida and our sponsor. Blanchard Park served as our primary outdoor flight-testing environment, offering the open space required for autonomous trials while remaining accessible to the team. The Texas Instruments (TI) Innovation Lab was essential for hardware development, providing soldering equipment, power-testing tools, and 3D printers used to fabricate custom mounts for the Jetson, ZED 2i, and power components. The Senior Design Lab provided a collaborative workspace for long integration sessions, debugging, and team coordination, especially during the final phases of system assembly and testing. Lastly, Serco's office spaces hosted our regular weekly in-person meetings where we reviewed milestones, aligned expectations, and prepared deliverables such as sponsor presentations and STEM Showcase materials.

The combination of these facilities, along with access to specialized hardware platforms, allowed us to build and test a sophisticated autonomous drone system that would not have been possible without such support.