Preliminary Design Document

LION Autonomous Drone

G10a

Rayyan Jamil, Abraham Ng, Nathaniel D'Alfonso, Connor Hallman

COP4934-25Summer C001

Prof. Matthew Gerber

7/7/2025

Table of Contents:

1. Project Introduction
    1.1. Brief Overview
    1.2. MVP Description
2. Student Sections
    2.1. Rayyan Jamil
        2.1.1. Core Architecture: GCS-Centric AI
        2.1.2. Drone Platform Selection: Holybro X500 V2 PX4 Development Kit
        2.1.3. Hardware Component Analysis: Onboard the Drone
        2.1.4. Hardware Component Analysis: At the Ground Control Station (GCS)
        2.1.5. Software Stack & AI Process Integration
        2.1.6. Communication & Data Flows: The System's Nervous System
        2.1.7. Example Use Case: Breaking Down a Natural Language Command
        2.1.8. Conclusion: An Optimal Path to AI Autonomy
    2.2. Abraham Ng
        2.2.1. Software Implementation of Agentic AI
        2.2.2. Stretch Goals Regarding Agentic AI
        2.2.3. Research and Project Design: Individual Contributions
        2.2.4. Team Dynamics and Deliverables: Individual Contributions
    2.3. Nathan D’Alfonso
        2.3.1. Foundational Research and Individual Contribution
        2.3.2. Drone Platform Evaluation and Selection
        2.3.3. Photogrammetry and Spatial Mapping Tools
        2.3.4. 3D Rendering and Visualization Research
        2.3.5. Real-Time Visualization and Data Processing
        2.3.6. Multi-Drone Operation and Swarm Coordination
        2.3.7. AI Integration Research and Strategy
        2.3.8. Planning for Future AI Integration
    2.4. Connor Hallman
        2.4.1. Individual Contributions

2.4.2. Picking The Best Photogrammetry
2.4.3. Existing SLAM Methods
2.4.4. The Agent
2.4.5. Robotic Simulation
2.4.6. Effective Library Building

1. Project Introduction

1.1 Senior Design Project Overview

Our senior design project is focused on developing an advanced autonomous drone system that bridges the gap between human intent and robotic action through sophisticated Artificial Intelligence. The overarching goal is to create a drone capable of understanding and executing complex commands given in natural language, enabling intuitive interaction for tasks in diverse environments. This ambitious objective is powered by a multi-faceted AI architecture that includes a Visual Language Model (VLM) for real-time visual perception, a Large Language Model (LLM) for high-level reasoning and decision-making, and an Agentic AI Framework (like SmolAgents) to orchestrate these intelligent components.

Our chosen system employs a GCS-centric AI architecture, where the drone itself functions as a highly capable sensor platform and actuator, while the heavy computational lifting for AI processing occurs on a powerful Ground Control Station (GCS). The drone, built around the Holybro X500 V2 PX4 Dev Kit with a Raspberry Pi and Intel RealSense stereo camera onboard, is responsible for acquiring high-fidelity visual and depth data and executing precise flight commands. This strategic division of labor allows us to maximize AI performance by leveraging GCS computing power, simplify onboard drone systems, and dedicate our primary efforts to innovative software development rather than complex hardware fabrication.

1.2 Minimum Viable Product (MVP)

The Minimum Viable Product for our autonomous drone system will demonstrate core capabilities in intelligent perception, spatial understanding, and autonomous navigation. Our MVP will feature a drone utilizing an agentic and 3D spatial system for autonomous flight within a defined indoor environment. The drone, equipped with its Intel RealSense camera, will actively create detailed 3D maps of its environments, specifically by generating comprehensive point cloud data from its live depth stream.

This captured 3D point cloud data will then be used by our GCS-based AI for further modeling and classification, allowing the system to identify and categorize objects within the generated

map (for example, distinguishing a chair from a table). Crucially, the system will demonstrate wayfinding capabilities, where the agentic AI can leverage this classified 3D map to autonomously navigate the drone to specified object locations or perform maneuvers like orbiting identified objects, all initiated through natural language commands issued from our GCS interface.

2. Student Sections

2.1 Rayyan Jamil

2.1.1 Core Architecture: GCS-Centric AI

After looking into different system setups, I decided to use a **Ground Control Station (GCS)-Centric AI Architecture**. This design principle sets up a clear division of work between the drone and the ground computer, a decision that has a big impact on all of my other hardware and software choices. This approach comes from a practical look at the project's goals, letting me focus on the most important parts of my contribution.

The main ideas of this architecture are:

- **The Drone as a Sensor and Actuator:** In the model I have outlined, the drone is set up to be an agile, sensor-rich platform and a precise actuator. Its main jobs are to reliably capture good sensor data—specifically, live video and depth information—and to accurately follow the flight commands it gets from the GCS. By moving the heavy computing to the ground, we keep the drone lightweight, which can help with flight time, and makes the onboard electronics simpler. This approach treats the drone as a capable "body" that is controlled by an outside "brain."
- **The GCS as the AI Brain:** The Ground Control Station is the main processing hub for the whole system. All computationally heavy tasks are handled here. This includes running the Visual Language Model (VLM), the Large Language Model (LLM), the agentic framework, and processing 3D point clouds from the drone's depth sensor. This setup lets the team use the power of dedicated AI hardware (like an NVIDIA Jetson or a good desktop GPU), which would be too heavy and power-hungry to put on a drone of

this size. Also, having the AI on the ground makes development easier, since I can debug and test AI code on the GCS without needing to mess with the drone hardware for every small change.

This GCS-centric architecture gives the project a good balance between advanced AI capabilities, hardware simplicity, and making sure the project is doable within a semester. It allows the project to achieve high performance with a cost-effective drone by putting the heavy processing on the ground.

2.1.2 Drone Platform Selection: Holybro X500 V2 PX4 Development Kit

My research led me to choose the **Holybro X500 V2 PX4 Development Kit** as the best drone platform for this project. This kit is a solid, open-source, and highly programmable platform that fits perfectly with the GCS-centric architecture and my focus on software. It is more than just a drone; it is a complete platform for development.

2.1.2.1 Minimizing Hardware Hurdles for Software-Focused Innovation

A key factor in my choice was finding a kit that was easy to assemble. The X500 V2 is an "Almost Ready to Fly" (ARF) kit. This is a big plus because important parts that can be tricky to install, like the motors and Electronic Speed Controllers (ESCs), are already installed on the arms. The main frame assembly is mostly plug-and-play and doesn't require much, if any, soldering for the main drone systems. This design greatly reduces the build time and chances for error, which lets my team and I spend more time on our main focus: the software and AI.

2.1.2.2 Full Programmable Control via PX4 and MAVSDK

At the center of the kit is the **Holybro Pixhawk 6C Flight Controller**. This unit runs the well-known **PX4 Autopilot firmware**. PX4 is a key part of the open-source drone community, known for being reliable and having many features for autonomous flight. It uses the **MAVLink protocol** for communication, which is an open standard. This protocol is how I can get full programmatic control from our GCS. By using the high-level **MAVSDK-Python library**, the GCS-based AI can send clear commands—like goto_location(), set_velocity(), or

orbit()—without having to worry about the low-level flight stabilization, which the Pixhawk handles.

2.1.2.3 Payload Capacity & Future Expandability

The X500 V2 has a 500mm wheelbase, which makes it a stable platform with plenty of space and a good payload capacity. I figure it can carry between 1kg and 1.5kg, not including the main battery. This is important because it can easily carry the required onboard electronics: the Raspberry Pi 4 companion computer and the Intel RealSense Stereo Camera. The frame was designed with mounting rails for this kind of extra equipment, which shows it was built for advanced projects like ours.

2.1.2.4 Cost-Effectiveness and Budget Allocation

At around $750 for the kit, the X500 V2 is a great value. It is a professional-level development platform for much less than bigger, pre-built commercial drones. This cost-effectiveness is a key advantage, as it lets the team spend more of its budget on the important AI hardware for the GCS, like an NVIDIA Jetson developer kit, which is needed to run the VLM and LLM.

2.1.2.5 A Robust and Reliable Open-Source Foundation

The Pixhawk/PX4 ecosystem is known for being stable and reliable for autonomous flight. It's a mature platform with a large community and a lot of documentation. By building the project on this foundation, we are using years of work from a global community, which gives the project a solid base for its advanced AI control systems.

2.1.3 Hardware Component Analysis: Onboard the Drone

The parts mounted on the Holybro X500 V2 are chosen to let the drone do its job as a mobile sensor and actuator.

2.1.3.1 Core Airframe and Propulsion System

- **Holybro X500 V2 Frame Kit:** The carbon fiber chassis, including plates, arms, and landing gear. It provides the structure and mounting points for everything else.

- **Brushless Motors (x4) & Propellers:** High-performance motors, pre-installed on the arms, that spin the propellers to create thrust for flight.
- **Electronic Speed Controllers (ESCs) (x4):** Modules that take signals from the flight controller and control the speed of each motor.
- **Power Distribution Board (PDB):** A circuit board that sends power from the main battery to the four ESCs and other parts, simplifying the wiring.

2.1.3.2 The Flight Control Unit: Holybro Pixhawk 6C

The **Holybro Pixhawk 6C** is the main controller of the aircraft. This microcontroller contains sensors like IMUs and a barometer. Running the PX4 Autopilot firmware, it handles the core flight control. It processes sensor data to keep the drone stable, manages flight modes, and most importantly for this project, turns the high-level MAVLink commands from the GCS into the low-level motor outputs needed for controlled flight.

2.1.3.3 Navigation and Communication Systems

- **Holybro M10 GPS Module:** Contains a GPS receiver and a compass. It is usually mounted on a mast for a clear sky view and provides location and heading data, which is needed for outdoor navigation and holding a position.
- **Holybro SiK Telemetry Radio (Drone Side):** A small radio for the main command and control (C2) link. It creates a reliable, low-bandwidth connection with the GCS to send critical MAVLink telemetry and receive MAVLink commands.
- **RC Receiver:** A radio receiver connected to the Pixhawk. It receives signals from the human pilot's RC Transmitter for manual flight control. This is a key safety feature, providing a reliable override.

2.1.3.4 The Onboard Companion Computer: Raspberry Pi 4

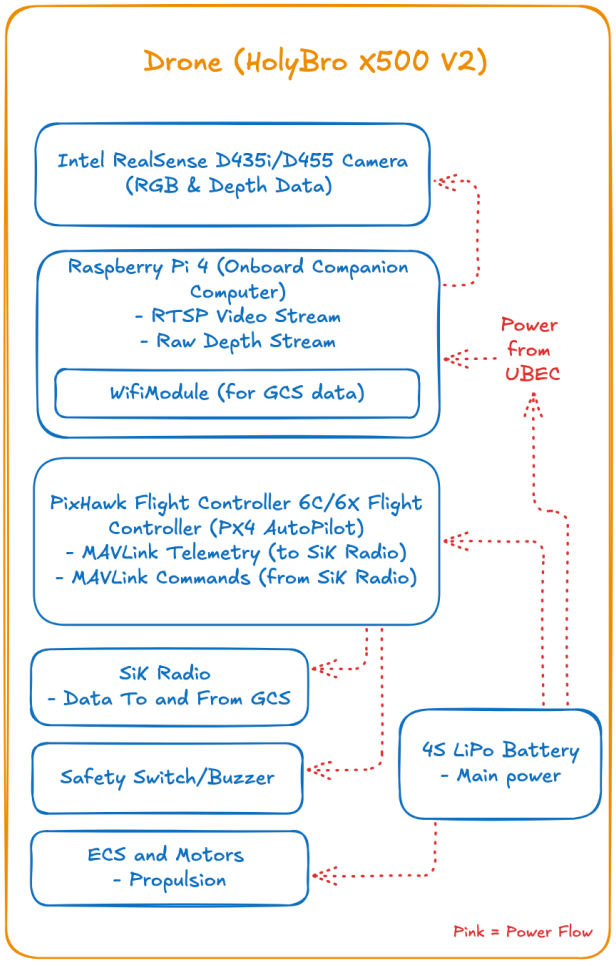
A **Raspberry Pi 4 (8GB RAM)** is the onboard data hub. It is a small, single-board computer running Linux. Its job in this system is to handle all the high-bandwidth data. It connects to the stereo camera, captures video and depth data, compresses the depth info, and streams both to the GCS over Wi-Fi.

2.1.3.5 Primary Perception Sensor: Intel RealSense D435i/D455 Stereo Camera

The **Intel RealSense D435i/D455** is the main sensor for the project's AI. It is a 3D camera with a standard RGB camera and infrared cameras for depth sensing. It provides the real-time RGB video feed for the VLM and the raw depth data that the GCS will use to create 3D point clouds of the area.

2.1.3.6 Power Distribution and Safety Systems

- **Main LiPo Flight Battery (OVONIC 4S 4000mAh 130C):** The drone's main rechargeable power source, supplying power to the motors and all electronics.
- **Dedicated UBEC/Buck Converter (5V @ 5A-6A):** A small but important module that steps down the main battery's voltage (e.g., 14.8V) to a stable 5V. This provides clean power for the Raspberry Pi 4 and the RealSense camera, preventing power issues that could cause them to fail.
- **Safety Switch & Buzzer:** A physical button and an alarm. The safety switch prevents the motors from accidentally starting, and the buzzer gives audio feedback about the drone's status.



2.1.4 Hardware Component Analysis: At the Ground Control Station (GCS)

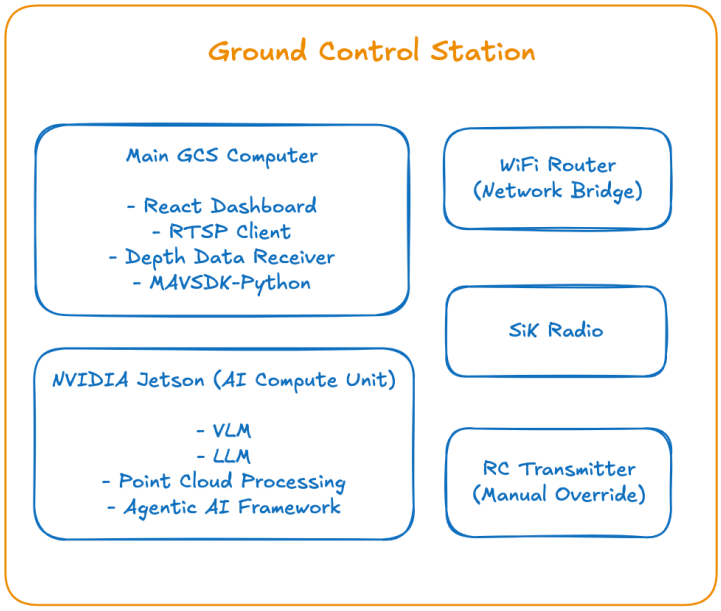
The GCS is the central command and intelligence hub, with the main AI processing hardware.

2.1.4.1 The AI Brain: GPU-Accelerated Compute Platform

- **NVIDIA Jetson:** This is the dedicated, GPU-accelerated AI computer and the main workhorse of the system. While a powerful desktop PC with a good GPU could also work, the Jetson is a compact, power-efficient option designed for AI. Its job is to run the most intensive tasks:
    - **VLM (Visual Language Model):** Processing the live video stream.
    - **LLM (Large Language Model):** Acting as the core reasoning engine.
    - **3D Point Cloud Generation:** Turning the raw depth data into a 3D model.
- **Main GCS Computer / Workstation:** A standard desktop or laptop. It hosts the OS, development tools, and the project's custom React Dashboard UI. It acts as the main user interface and coordinates the software.

2.1.4.2 Command, Control, and Communication Interfaces

- **Wi-Fi Router:** A standard Wi-Fi router that creates a local network for all high-bandwidth communication between the GCS, the Jetson, and the drone's Raspberry Pi.
- **Holybro SiK Telemetry Radio (GCS Side):** The other half of the drone's radio, this connects to the GCS computer via USB. It creates the reliable MAVLink link for sending commands and receiving telemetry.
- **RC Transmitter (Remote Control):** The physical remote control held by the operator. It provides the direct manual override for safety.
- **LiPo Battery Charger:** A special charger for safely recharging the drone's LiPo flight batteries.



2.1.5 Software Stack & AI Process Integration

The system I have designed uses a software stack that is split across both the drone's companion computer and the ground station.

2.1.5.1 Onboard Software (Raspberry Pi 4)

- **Operating System:** Raspberry Pi OS Lite (64-bit). A minimal Linux version without a graphical interface to save resources for data handling.
- **RealSense SDK (librealsense & pyrealsense2):** The official Intel SDK with drivers and APIs to use the RealSense camera.
- **RTSP Video Streamer (Python/GStreamer):** A Python script using GStreamer or FFmpeg to encode the RGB video and stream it over the Wi-Fi network.
- **Raw Depth Data Streamer (Python):** A Python script that grabs depth frames, compresses them (using something like zstandard or LZ4), and streams the data to the GCS.

2.1.5.2 Ground Control Station Software

- **Operating System:** Ubuntu Linux is preferred for the GCS, especially on the Jetson, because it works well with AI and robotics software.
- **React Dashboard:** The project's custom web UI with a chat window for commands, a telemetry display, and a live video feed.
- **Agentic AI Framework (SmolAgents - Python):** The core Python code that orchestrates the AI. It takes user input, asks the LLM for a plan, and calls "tool" functions to carry out the plan.
- **AI Models (VLM & LLM):** The pre-trained neural networks running on the Jetson, which are used by the Agentic Framework.
- **PointCloudProcessor (Python):** A process on the Jetson that receives the depth data, decompresses it, and turns it into a 3D point cloud.
- **Agentic "Tools" (Python):** Python functions that the Agentic Framework can call:

    - **VLMAgent:** A tool to ask the VLM a question about what it sees.
    - **PilotAgent / DroneAgent:** A tool that turns the AI's high-level decisions (like "move forward 2 meters") into MAVSDK-Python commands.
- **Drone Control Logic (MAVSDK-Python):** The Python library that talks directly to the drone's flight controller. It creates and sends MAVLink messages and receives all incoming telemetry.

2.1.6 Communication & Data Flows: The System's Nervous System

Good communication between parts is key to making this architecture work. I have planned for different channels, each suited for its specific type of data.

2.1.6.1 High-Bandwidth Data Stream (Drone → GCS)

This link uses Wi-Fi and carries sensor data that needs a lot of bandwidth.

- **RGB Video Stream:** The Raspberry Pi streams video from the RealSense camera to the GCS, where it is fed to the VLM and shown on the dashboard.
- **Raw Depth Data Stream:** The Raspberry Pi compresses and streams depth data to the GCS, where the PointCloudProcessor uses it.

2.1.6.2 Robust Command & Telemetry Link (Drone ↔ GCS)

This link uses the dedicated SiK telemetry radios and the MAVLink protocol. It is for reliable, low-latency communication of small data packets. This is the drone's main control link.

- **MAVLink Telemetry (Drone → GCS):** The Pixhawk sends real-time flight data to the GCS, which updates the dashboard and gives the AI situational awareness.
- **MAVLink Commands (GCS → Drone):** The GCS AI sends flight commands to the drone's Pixhawk, which carries them out.

2.1.6.3 Internal GCS AI Logic Flow

This data flow happens on the GCS local network.

1. A user command from the React Dashboard goes to the Agentic AI Framework.

2. The Framework sends the text to the LLM for planning.
3. The Framework might call the VLMAgent to check the video feed or the point cloud.
4. Once a decision is made, the Framework calls the PilotAgent, which creates a command for the Drone Control Logic.

2.1.6.4 Manual Override Safety Link

This is a direct radio link. The pilot's RC Transmitter sends signals straight to the RC Receiver on the drone. The Pixhawk is set up to always listen to these signals over any computer commands, providing an instant manual override.

2.1.7 Example Use Case: Breaking Down a Natural Language Command

To show how the system works, here is how it would handle the command: **"Fly up 5 feet and then go in a circle around the red chair."**

1. **User Input (React Dashboard):** The operator types the command. The React app sends this text to the Agentic AI Framework.
2. **AI Interpretation (GCS - LLM):** The Framework sends the command to the LLM. The LLM understands it is a two-step plan: (1) go up, and (2) circle an object, which is "the red chair." The LLM knows it needs to find this object visually.
3. **Perception & Spatial Understanding (GCS - VLM & PointCloud):**
    - The RealSense camera is already streaming video and depth data to the GCS.
    - The Framework uses the VLMAgent tool to find the "red chair" in the video. The VLM returns the location of the chair in the video frame.
    - The PointCloudProcessor has been making a 3D point cloud from the depth data.
    - The AI combines the VLM's 2D location with the 3D point cloud to find the chair's actual 3D coordinates.
4. **AI Planning & Command Generation (GCS - LLM & MAVSDK):** The LLM now has all the info it needs: the drone's current location (from telemetry) and the chair's 3D coordinates. It creates a two-part flight plan:
    - **Part 1:** It tells the PilotAgent to arm() and takeoff() to an altitude of 5 feet.

    - **Part 2:** Once the drone is at the right height, it tells the PilotAgent to orbit() around the chair's coordinates.
5. **Drone Execution (Pixhawk):** The MAVSDK-Python library turns these commands into MAVLink messages and sends them over the SiK radio to the Pixhawk. The Pixhawk's firmware then controls the motors to make the drone arm, take off, and fly in a circle around the chair.
6. **Real-time Feedback:** The whole time, the drone sends back telemetry and live video to the dashboard, so the operator can see what is happening.

2.1.8 Conclusion: An Optimal Path to AI Autonomy

The research and system architecture in this report lay out a complete and practical plan for my area of the project. The proposed setup, using the Holybro X500 V2 kit with a Raspberry Pi and a powerful GCS for AI processing, is the best path forward for this part of the senior design project.

This GCS-centric design is a specific choice that allows the team to:

- **Maximize Focus on Software Innovation:** By using a kit that is mostly pre-built and moving the heavy computing to the ground, I can spend my time on my part of this project: the AI algorithms.
- **Achieve High-End AI Performance:** By using a dedicated ground-based GPU, the system can run complex VLM and LLM models that would be impossible to run on an affordable drone.
- **Maintain Cost-Effectiveness:** The architecture lets the team hit its goals while staying within budget.
- **Build Upon a Robust Foundation:** Using the well-supported, open-source Pixhawk/PX4 ecosystem ensures the platform is reliable.
- **Ensure Project Feasibility:** With a clear division of tasks and a focus on software, I am confident that this architecture will contribute to a working autonomous AI drone system in the project timeline.

2.2 Abraham Ng

2.2.1 Software Implementation of Agentic AI

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

To accomplish this, our software system will thus be composed of several key architectural components:

2.2.1.1 Model Context Protocol

The Model Context Protocol (MCP) was recently introduced by Anthropic less than a year ago. The MCP has seen widespread adoption across the AI ecosystem for standardizing how tools, data, and other environmental contexts are communicated to large language models and other AI agents. The MCP enables consistent interfacing and interoperability without having to create custom connectors for every combination of agent and tool.

In our implementation, the MCP will serve as the backbone of inter-agent communication in our system, allowing each agent to access a shared context, including:

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

2.2.1.2 Ollama and Large Language Models

In alignment with our sponsor’s requirements for data localization and security, it is necessary that we plan to use Ollama to run large language models on our NVIDIA Jetson hardware. Ollama is a platform that is optimized for the deployment of lightweight language models. By

using Ollama, our system gains:

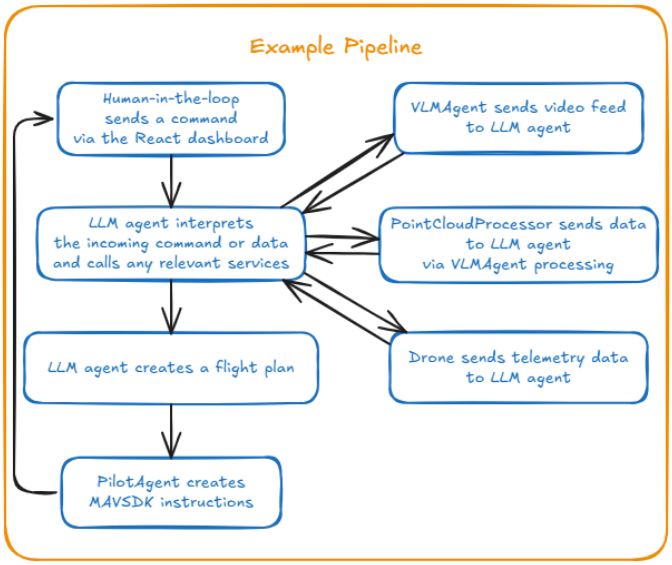
- **Offline operation**, without the need for internet connectivity,
- **Data privacy**, as all data and human prompting remains local,
- **Model flexibility**, since Ollama allows for the facile swapping of LLMs, and
- **Low-latency inference**, operating with real-time responsiveness even on limited hardware

We currently plan to use Llama or Mistral as our large language models, serving as command interpreters and parsing human instructions into structured goals. For visual language model tasks such as interpreting image or video data, we are currently evaluating several candidates, including multimodal versions of the above large language models.

2.2.1.3 LangChain and LangGraph

LangChain and LangGraph will allow us to establish a controlled flow involving multiple agents, and both are compatible with the MCP framework. Below is one potential pipeline for our agentic system:

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.



LangChain and LangGraph will facilitate conditional branching, such as the PointCloudProcessor sending its point cloud data to the command interpreter

agent if the VLMAgent does not immediately identify or find an object. In addition, LangChain and LangGraph allow for cycle or task repetition, such as in having the DroneAgent continue proceeding with autonomous navigation or having the VLMAgent continuously send image data until an object is discovered, depending on the tools that we create and the tuning of various language models.

2.2.1.4 Docker and Containerization

Each major component in our system will operate within its own Docker container. This approach offers several advantages:

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

2.2.2 Stretch Goals Regarding Agentic AI

Our minimum viable product centers on achieving drone autonomous navigation. However, we have identified several other stretch goals involving agentic AI that would enhance the functionality and intelligence of our system.

2.2.2.1 Semantic Filtering During SLAM

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

1. An operator could type into the React dashboard, “Scan this field but ignore trees and chairs.”
2. The agentic system would interpret this command and pass the relevant filtering targets (trees and chairs) to the VLMAgent.

3. The VLM agent then uses object detection to tag trees and chairs in its visual feed or in the point cloud data.
4. The PointCloudProcessor finally filters or removes those objects during the SLAM pipeline, either:
    - Pre-processing, before the point cloud is built, or
    - Post-processing, after the digital twin is complete.

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

2.2.2.2 Temporal Change Detection

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

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

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

2.2.3.3 Context-Aware Task Prioritization

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

1. The operator initially issues a command, “Scan this area and identify all construction equipment.”

2. Partway through the scan, the drone encounters an object not related to construction equipment.
3. The VLMAgent classifies the object as anomalous and sends a flag to the React dashboard.
4. The general autonomous scanning is interrupted, and the drone examines the unexpected object in detail.
5. The LLM then reports the results to the operator, upon which the scan will pause and can be resumed based on the operator's input.

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

2.2.3 Research and Project Design: Individual Contributions

The following items constitute several aspects related to the project and overall design on which I worked or for which I did research.

- **Project MVP Proposals:** I drafted two proposals involving multi-agent drones for a minimum viable product for our project, although we eventually ended up picking a different proposal. Furthermore, I also helped assess the various team proposals and assisted in the vote tallying. Lastly, I worked on clarifying the role of agentic AI in the project proposal we selected.
    - **"Space Invaders"**: One drone would simulate firing virtual lasers at varying speeds and patterns against another drone that would attempt evasive maneuvers, but both drones would have potential restrictions on vision range and sensor usage frequency.
    - **Package Delivery Service**: The first drone would conduct a larger-scale geographical scan of multiple roads and current traffic, performing general pathfinding analysis, whereas the second drone would perform smaller-scale scans of its immediate vicinity while performing the actual route navigation and delivery.

- **Photogrammetry**: I looked into the different techniques and methodologies for obtaining a three-dimensional digital model of the surrounding environment of a drone, comparing and contrasting approaches.
    - Traditional photogrammetry algorithms can offer higher-resolution models with better accuracy, but they demand longer processing times and do not work incrementally.
    - Simultaneous localization and mapping (SLAM) techniques can create real-time models at the same time as data is acquired but at the cost of somewhat lower accuracy.
    - Since we want our user dashboard to include a live rendition of the virtual model in real-time as the drone performs its scanning, we decided to pursue SLAM over other possibilities.
- **Machine Learning Research**: Due to my lack of prior background in machine learning, neural networks, and large language models, I had to spend considerable time getting up to speed in these domains, perusing a number of online video tutorials as well as enrolling in and completing several introductory courses.

2.2.4 Team Dynamics and Deliverables: Individual Contributions

The following items constitute several aspects related to group deliverables and team dynamics to which I made major contributions.

- **Notional Sprint Map**: I helped set up our notional sprint map for our early status report, planning out the months below.
    - **June**: Our main focus was finalizing our minimum viable product as well as performing foundational research into the various technologies and hardware necessary for our project.
    - **July**: Our main focus has been selecting and purchasing our candidate drone, with additional efforts toward selecting our various software components and working on initial builds.

    - **August**: Our main focus will be unit testing and preliminary integration testing as we continue building hardware and software in preparation for our first full end-to-end test the following month.

- **Team Dynamics and Communication**: I worked on multiple sections of our team contract assignment, aided in gathering contact information from our team members, set up our group Discord and organized it appropriately for our project, and helped brainstorm several names for our group project.

- **Proofreading and Formatting**: For our three group deliverables thus far – the design proposal and team contract, the early status report, and the preliminary design document – I dedicated time to proofread, reorganize, and format our drafts before final submissions.

2.3 Nathan D'Alfonso

2.3.1 Foundational Research and Individual Contribution

To prepare for building our system, I spent the past few weeks researching the key technologies needed for the project. Since we did not have access to physical hardware yet, I focused on evaluating drone platforms, photogrammetry tools, 3D rendering software, data workflows, and AI/ML technologies. My goal was to understand how these tools could work together in an open-source setup to create a 3D digital twin using either a single drone or a multi-drone approach.

I researched several drone kits on my own, focusing on how programmable, expandable, and compatible they were with onboard computers. I made comparison charts that highlighted key trade-offs like cost, sensor support, and flexibility. At the same time, I looked into open-source photogrammetry and point cloud tools, checking how well they worked in real-time or near real-time settings. I organized all my findings into a shared document to help guide our system integration planning.

I helped define what our Minimum Viable Product (MVP) should include by evaluating what we could realistically build within our time and budget limits. During planning meetings, I

asked technical questions and gave input on which features to prioritize. I also assessed the risks of potential stretch goals like real-time rendering and AI-assisted navigation, and shared my thoughts on hardware and processing tool trade-offs to help shape our overall implementation strategy.

To help avoid problems later on, I pushed for a structured research phase early in the project. I focused on long-term scalability, system reliability, and how complex the build would be. I also created planning materials like roadmaps and diagrams to show how our hardware and software choices connect to specific project goals. This early preparation helped reduce the chances of delays during integration and development.

I plan to help build and test the core parts of the system in both simulations and real-world settings. The research I have done so far will help solve technical problems more efficiently and reduce time spent fixing issues later.

Since this project covers many technical areas, I organized my research into clear categories with specific goals:

- Choosing the best drone platform
- Comparing tools for converting between mesh and point cloud formats
- Evaluating 3D rendering tools for spatial data
- Reviewing integration workflows and toolchains

To stay efficient, I used filters like open source availability, budget fit, and compatibility with our tools. This helped avoid distractions and kept the research focused.

2.3.2 Drone Platform Evaluation and Selection

I focused my research on five key areas that were essential to building a working and scalable system. Each area supported a major part of the project:

- **Drone Platforms:** I compared several kits based on computing power, payload space, ease of setup, and how well they worked with third-party tools.

- **Photogrammetry Tools:** I evaluated open-source and commercial options for turning images into 3D data like point clouds or meshes.
- **Unity for 3D Visualization:** I tested Unity's ability to render 3D scan data, including support for standard formats like OBJ and PLY.
- **ROS Integration:** I looked into ROS-compatible tools that help drone components communicate and work together reliably.
- **AI and ML Tools:** I began early research into AI tools for tasks like object detection or improving scan coverage.

All of this research was chosen to fit into one streamlined system that could grow over time.

To make sure my research was thorough and accurate, I used a wide range of sources, including:

- Technical documentation and whitepapers
- GitHub projects and wikis
- Academic papers and preprints
- Online tutorials, forums, and case studies
- Direct feedback from professionals

I organized everything into comparison charts showing trade-offs in integration, usability, cost, and performance. These charts were shared with the team on Google Drive, Jira, and Discord to support informed decision-making.

A key part of my research was a meeting with Dr. Mohsen Rakshan, a professor at the University of Central Florida. He shared real-world experience from lab projects using LiDAR for spatial mapping, which helped me better understand the practical challenges of LiDAR systems. His insights helped me compare LiDAR and stereo vision in terms of cost, environment flexibility, and ease of integration. I plan to use this knowledge when selecting sensors in future phases of the project.

Choosing the right drone was one of the most technically demanding parts of my research. I compared both commercial and DIY options, including the DJI Mavic 3, Skydio 2+, Parrot Anafi, Holy Stone HS720E, and kits based on Raspberry Pi or Jetson Nano.

I created comparison charts based on key factors like:

- Autonomous flight capability
- Imaging quality
- Support for photogrammetry or LiDAR
- SDK/API flexibility
- Compatibility with open source tools
- Modularity and expandability
- Support for ROS/MQTT communication
- Cost and availability

This helped highlight the strengths and limitations of each platform, playing a direct role in our final selection process.

My research revealed several important trade-offs across different drone options:

- **DJI drones** had great camera quality and long flight time but limited flexibility due to restricted SDK access.
- **Skydio drones** offered advanced autonomy but were locked into a closed software ecosystem.
- **Parrot Anafi** featured open SDKs and decent performance but lacked support for advanced sensors.
- **DIY kits** provided flexibility but required too much engineering effort for our limited time frame.

This analysis helped the team understand what was realistic to implement and guided our final platform decision.

During my evaluation process, I also considered how each platform could scale with future project needs. For instance, while some commercial drones provided great out-of-the-box performance, they lacked modularity for sensor upgrades or custom payloads. In contrast, open-source-friendly platforms, though more difficult to configure, offered long-term flexibility that aligned better with our vision for a scalable, research-grade system. I documented these trade-offs to help the team weigh short-term ease against long-term adaptability.

Although I was not the one to make the final drone selection, I played an important role by providing comparison charts that outlined each platform's flexibility, modular design, and support for communication protocols like ROS and MQTT. I also researched how each drone's SDK connects with these systems by reviewing technical docs and examples. This helped confirm that our communication setup would work and supported the team's final decision.

2.3.3 Photogrammetry and Spatial Mapping Tools

To help generate accurate 3D spatial data from drone images, I researched open-source photogrammetry tools that could fit into our workflow. I focused on tools that were reliable, well-documented, and could support automation for future scaling.

The main tools I evaluated were:

- **Meshroom** – User-friendly with a visual interface, great for high-quality images and simple workflows.
- **COLMAP** – A more advanced tool used in research, supporting Structure from Motion (SfM) and Multi-View Stereo (MVS). It also works in headless mode for automated processing.
- **OpenMVS** – Often used with COLMAP to generate dense reconstructions and export mesh or point cloud data.

I compared them based on community support, ease of use, automation support, output quality, and how well they integrate with our visualization tools. The results were documented for future use when drone images become available.

After speaking with Dr. Mohsen Rakhshan, I gained a clearer understanding of the trade-offs between LiDAR and photogrammetry. LiDAR is effective at generating direct point clouds and works well in complex or low-light environments. However, its high cost makes it better suited as a future upgrade rather than a core part of our MVP. This insight helped guide my continued research into sensing options and tool compatibility.

To help speed up processing and reduce manual effort, I researched ways to automate our photogrammetry workflow. The goal was to make it easier to go from drone imagery to a visualized 3D model with minimal human input. I focused on tools and techniques that could be connected into a streamlined pipeline, including:

- **Headless COLMAP**: Runs from the command line and supports GPU acceleration, making it suitable for automation and batch processing.
- **CloudCompare or Blender**: Used for cleaning up, filtering, and converting point cloud or mesh data after reconstruction.
- **Unity Import Plugins**: Allow photogrammetry outputs to be brought into Unity with minimal manual setup, supporting our dashboard visualization goals.

This setup would support quick iterations during testing and help reduce turnaround time when working with large or repetitive scan data.

2.3.4 3D Rendering and Visualization Research

Although not required for the MVP, I researched 3D rendering tools to support future project phases. I focused on the Unity Game Engine due to its strong 3D capabilities, plugin flexibility, and support for WebGL, which enables browser-based deployment. Unity also supports formats like OBJ and PLY, making it compatible with outputs from tools like Meshroom and COLMAP. This confirmed Unity as a solid option for both static mesh rendering and potential real-time point cloud visualization.

As part of my research, I explored **Pcx**, a Unity plugin built to handle large-scale point cloud data efficiently. It supports billions of points and offers customization through shaders and material settings. I reviewed its documentation, GitHub resources, and user demos to assess performance and limitations. Based on this, I found Pcx to be a strong option for future dashboard development if we choose to implement point cloud visualization later in the project.

I researched methods used in the industry to improve how dense 3D data is rendered, especially on lower end computers or in web browsers. These included:

- **Occlusion Culling:** Avoids rendering objects that are not currently visible on screen.
- **Shader-Based Filtering:** Simplifies how particles are drawn to lower processing demands.
- **Level of Detail (LOD):** Reduces detail for far-away objects and increases it for close-up ones.

These techniques will help us create a smoother and more stable experience if we decide to render large scans or use real-time visualization in future versions.

To prepare 3D data for use in Unity, I studied common preprocessing steps that make rendering faster and cleaner. These included:

- **Downsampling:** Reducing the total number of points in the data to speed up rendering.
- **Noise Filtering:** Removing bad or inaccurate points that can clutter the final output.
- **Normalization:** Aligning all data to a shared coordinate system so it is easier to import and view.

These steps are important for turning raw scans into optimized 3D models that load quickly and look clean in our dashboard.

2.3.5 Real-Time Visualization and Data Processing

While our MVP follows a simple capture process render workflow, I researched ways to upgrade the system for near-real-time streaming. This involved exploring how video or depth

data could be sent directly to a Unity-based frontend using tools like WebSockets or ROS bridges. Investigating this stretch goal helped us understand key trade-offs, such as latency, bandwidth use, and system load—important factors for future scalability.

I looked into challenges that may arise when converting photogrammetry outputs like point clouds into formats Unity can handle without losing performance or visual quality. Tools like CloudCompare stood out for their ability to filter, downsample, and convert point cloud data. Although it has a steep learning curve, its advanced features will be useful for preserving detail and preparing data for smooth integration into our Unity-based visualization system.

To speed up processing and reduce manual steps, I researched ways to automate our photogrammetry workflow. I drafted a script concept that monitors a folder for new image data and automatically starts the processing pipeline. This approach would enable quicker testing, especially when working with small or frequent scan updates.

2.3.6 Multi-Drone Operation and Swarm Coordination

I explored a manual approach to coordinating multiple drones by dividing the scan area into predefined grid sections. Each drone would be assigned a specific sector, helping to avoid scan overlap and making post-processing easier. This simple method fits well within the practical limits of our MVP.

To better understand how multiple drones could coordinate effectively, I reviewed case studies from academic research and real-world applications in search and rescue and agricultural surveying. These examples highlighted the importance of synchronized flight planning and real-time communication between agents. I considered how sector-based scanning could later evolve into fully autonomous swarm behavior, using ROS-based task assignment and inter-drone messaging to dynamically manage scan coverage without human input.

I explored future ideas for making our drones work together as a smart group, or “swarm,” that could adapt while scanning. Topics I looked into included:

- **Scan Coverage Monitoring:** Drones would track what areas have or have not been scanned.
- **Obstacle Navigation:** They could detect and move around barriers in real time.

- **Adaptive Pathfinding:** Flight paths could adjust based on conditions or missing data.

To support this, I studied control methods like:

- **Boids-Based Models:** Simulates flocking behavior using simple rules.
- **Potential Fields:** Guides movement based on attraction or repulsion zones.
- **Behavior Trees:** Breaks complex actions into manageable decision steps for coordination.

While not part of our MVP, this research could guide future versions of our system that aim for more autonomy.

To support coordinated multi-drone operation, I researched how the Robot Operating System (ROS) can manage communication between agents. I focused on using separate ROS topics for each drone to help reduce issues like GPS drift. This setup also supports future features such as collision avoidance, task handoff, and scan coordination.

I explored how ROS message types can be used for real-time data sharing between drones. This included publishing position updates, scan completion signals, and task reassignment messages. Understanding these communication protocols prepares our system for future support of autonomous task coordination across multiple drones.

2.3.7 AI Integration Research and Strategy

I began researching how AI and machine learning could improve scanning efficiency and system intelligence. Key focus areas included detecting coverage gaps, prioritizing complex regions, tracking environmental changes, and enabling onboard object detection. I focused on tools designed for edge computing to ensure they could realistically run on lightweight onboard systems in future phases.

I researched AI tools that could run on low-power, compact hardware for possible use directly on drones. This included the Jetson Nano for GPU-accelerated edge computing, TensorFlow Lite and PyTorch Mobile for running lightweight models, and pre-trained object detection models like MobileNet and YOLOv5-Nano. I confirmed their practicality by reviewing

how these tools have been successfully used in similar robotics projects, making them strong candidates for future integration.

2.3.8 Planning for Future AI Integration

I identified several challenges that make early AI integration difficult. These include the lack of labeled drone data for training, limited memory and power on small onboard devices, and the complexity of connecting real-time AI outputs with Unity-based visualization. These findings supported our decision to delay AI features until the core system is stable and working reliably.

I helped develop a modular roadmap that supports future AI features without affecting the MVP structure. This plan focuses on building a stable base system first, developing AI components separately, and ensuring that future additions will not require major changes to the overall architecture.

To prepare for future AI features that could run directly on the drone, I researched lightweight tools that work well on small, resource-limited computers. I focused on options that have been successfully used in other robotics or embedded AI projects:

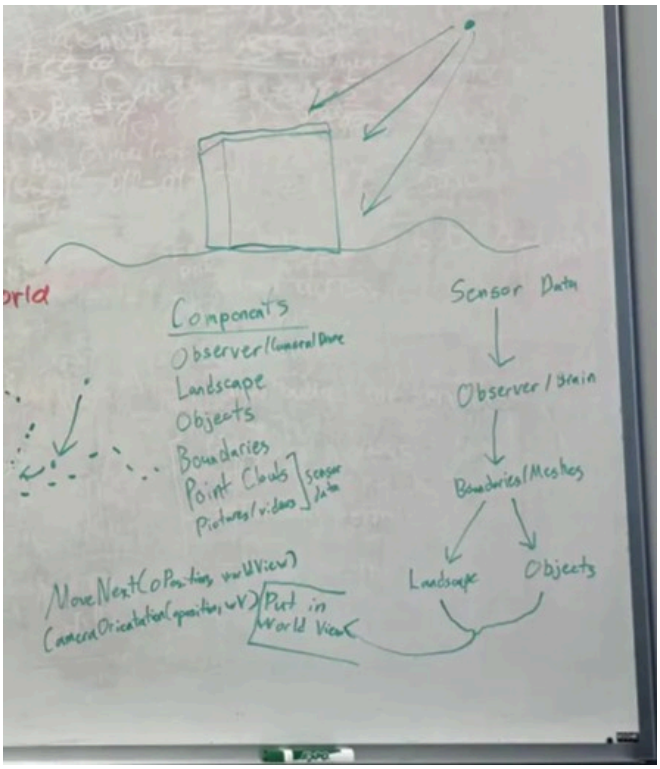
- **Jetson Nano**: A compact, GPU-enabled board made for edge computing. It supports real-time vision processing and deep learning at a small power cost.
- **TensorFlow Lite & PyTorch Mobile**: Lighter versions of the most popular machine learning frameworks, built specifically for mobile and embedded devices.
- **MobileNet & YOLOv5-Nano**: Pre-trained object detection models designed to run efficiently on small systems while still detecting people, vehicles, or other objects of interest.

I confirmed the practicality of these tools by reviewing open-source robotics projects and tutorials, and each one showed potential for future integration into our system.

2.4 Connor Hallman

2.4.1: Individual Contributions

My contributions throughout this project have been primarily brainstorming and researching. I helped come up with the initial design and idea. In one of our first meetings I drew up the initial concept for the exploration portion of the project, which was initially the entire project. After that, it became a lot of research for existing algorithms, potential drones, robotic simulation, and more.



2.4.1a: Meeting With Robotics Professor

I leveraged my connection with my previous robotics professors to get advice on our project since we are all computer science majors. Nate also attended the meeting. There were a few major takeaways from that meeting:

- Absolutely do not build a DIY drone
- Pay strong consideration to the specs of parts to make sure they can do everything you needed
- Leverage existing SLAM projects, do not DIY
- Be unique, don't let the project be building the drone and using existing libraries. Add functionality to stand out
- He recommended using a WiFi router to act coordinate communications

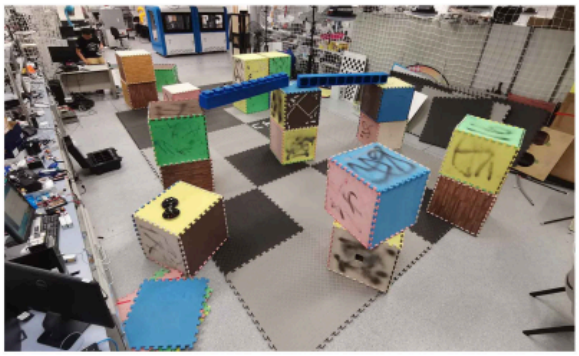
2.4.2: Choosing The Best Photogrammetry

Our drone will be heavily dependent on its sensors for accurate point cloud and rgb data. Nearly every one of our algorithms and tools will use it in some fashion. So making sure that data is accurate and optimized for our use case was very important to me. I tackled the problem the same way I tackle every problem, breaking down our use case to simplify the problem. I first identified where the sensor would be used and how much precision it needs. Here was everything

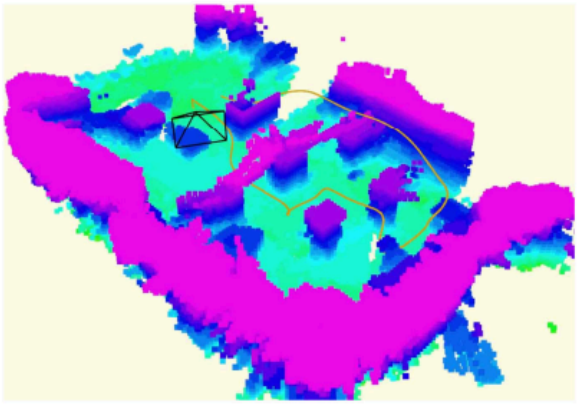
I took into account: operate outside, limited budget, weight, precision, data needed (rgb and point cloud data). Based on the data needed, we were brought down to LiDar or Stereogrammetry. LiDar is better but doesn't really fit our budget so it got ruled out. That left stereogrammetry (using two cameras for depth information). There are two types of stereogrammetry, passive and active. Active project infrared sensors are great indoors but are significantly affected outdoors. Which leaves the winner, passive stereo. We have many options and are yet to decide on the final one.

2.4.3: SLAM Algorithms

A lot of our project revolves around Synchronous Localization And Mapping (SLAM). The idea for how the drone operates is that it should always be doing slam, even when performing a tool like Move, Follow, and Orbit. However, we can also instruct the model to go into an exploration mode which will try to optimize for information gain; letting the drone map out an area as fast as possible. I did a lot of research on the different SLAM algorithms because using existing libraries will be a significant time saver for us. The exact SLAM algorithm we use will be determined by the sensor we used since some companies provide proprietary SLAM algorithms that only work with their sensors. We will likely use RTAB-Map or Zed SDK SLAM which both support RGB + Depth fusion.



(a) A cluttered environment for the exploration tests.



(b) The online built map and executed trajectory.

Regardless of what we pick, the SLAM algorithm will be paired with a Fast UAV ExpLoration (FUEL) library algorithm–pictured on the right–that will do the motion planning and exploration. The FUEL library runs on ROS which makes it compatible with most drone systems. It has a few hard requirements which should be easy to fulfill with our system:

- Odometry - 6 Degrees of freedom, tells where the drone is
- Point Cloud/Depth Map - to continuously build voxel map
- Motion Data - from IMU to help with pose
- Rgb Image Data - for visual slam

Based on all the data, FUEL will continuously plan minimum-time trajectories that will guide the drone to new positions that maximize information gain which effectively prioritizes the most valuable unexplored regions. This allows the drone to autonomously and intelligently cover unknown areas with high efficiency.

2.4.4: The Agent

One of the core requirements of our project was that it must be multi-drone and *agentic.* For a few reasons, we were able to drop the requirement for a multi-drone system. Primarily due to the fact that our budget ($2,000) was not enough to multiple drones. Additionally, our MVP would not benefit much from an additional agent, a true multi-agent system would be too complex to complete in our time, and a partially multi-agent system–think 2 agents splitting up work amongst themselves–was not worth it. So we decided to devote all our energy into a getting a single agent to work very well.

There was a lot of debate at the beginning about what agentic really meant. For a while, I, and the rest of us other than Rayyan, conflated agentic with autonomous. I thought that it would be enough for our agent to autonomously perform a task. But with some convincing from Rayyan and Chris, we realized agentic was more than that. We settled that agentic means that the drones agent must be capable of calling tools. Which meant we must create tools for the agent to call, define the parameters, and prompt the agent to teach it how to use the correct ones.

The way we imagine the agentic system working is a *joint human-robot system.* In this system, the human interacts with the drone through voice commands. They should ideally be able to issue commands as if the drone were operated by a coworker: “launch up 20 meters”, “scan that platform”, “follow that human”. These commands would be transcribed by a text-to-speech model like whispr and then put through an LLM which would determine the humans intentions and call the appropriate tools, which are defined below.

2.4.4.1 - The Tools

The following tools are ones we would like to expose to the agentic system. I have ordered them based on the priority for getting them implemented. The ones towards the bottom can be thought of as stretch goals. The sub bullet points define parameters to the tools.

- **Status -** return a value indicating what the drone is currently doing
- **Launch -** Starts the drone and then will likely call *Move*
    - Position - Where the drone should go (likely directly above its current position)
    - Pose - Which way the drone should face one launched
- **Land -** Similar to launch but to land the drone
    - Position - where to land
- **Hover -** Maintain position and orientation
    - Duration - how long to maintain pose
- **Look At -** Rotate camera to point at POI
    - Target Position - where to look
    - KeepTracking - boolean to instruct drone to update pose if target moves
- **Capture Photo/Video -** Save visual data
    - Resolution
    - Duration (if video)
    - Name
    - Tags - for grouping photos/videos
- **Move (Absolute) -** Move the drone to a specified position based on its SLAM interpretation of the environment
    - Position - Where the drone should go
    - Pose - Orientation of the drone
- **Move (Relative)** - Move the drone relative to its current position
    - Delta Position - How the far the drone should move from its current position, and in which direction
    - Delta Rotation - How much the drone should rotate, and in what direction
- **Orbit -** Move in a continuous circle about a point
    - Pose - face camera in or out

    - Radius - how wide to orbit
    - Speed - how fast to orbit object
- **Explore** - Perform SLAM algorithm on specified area to build virtual digital twin
    - Area - Can be defined as a circle (center and radius) or other shape, square (center and side length)
    - Time Limit - max time to allow exploration algorithm to go
- **Scan -** Perform SLAM algorithm on specific object in the scene (digital twin)
    - Object Id - The id that the SLAM algorithm has assigned an object
- **Follow -** Point the camera at an object and keep a consistent distance from object
    - Object Id - The id that the SLAM algorithm has assigned an object
    - Follow distance - how far to stay from object
- **Tag/Name Object -** Allow agent to give names/tags to objects in the SLAM algorithm
    - Name/Tags
    - Object Id

2.4.5: Robotic Simulation

We decided pretty early on that this project was a very good candidate for robotic simulation. We plan on using Unity since both Nate and I are already familiar with it. There are many reasons:

- Easier To Train And Test
    - Can be done from home instead of having to go to a field
    - Accelerates iteration cycle - Can just restart the scene instead of having to recall drone and start over
    - Multiple people can work on different features concurrently thanks to Git
- Safe Environment For Debugging
    - Can try new algorithms without worrying about destroying a $2,000 drone
    - Identify bugs quicker
- Repeatability
    - A virtual game environment will provide consistent data which will make it easier to test out algorithms

    - We do not have to worry about the sun, wind, and other environmental factors
        - Though they can be simulated if needed
- Data Generation
    - We can generate synthetic data to test certain edge cases
- Algorithms are environment agnostic / simulatable
    - Pretty much all our tools and algorithms are agnostic to whether the agent is in the physical world or a virtual world

2.4.5.1: Considerations with robotic simulation

At the end of the day, simulation is great, but not perfect. So I did research to determine some of the drawbacks and considerations to make robotic simulation:

- Data isn't actually perfect
    - Games have changing frame rates based on GPU and CPU usage which can cause inconsistent outcomes from algorithms
- Overfitting to simulation
    - The algorithms may become to used to running in a game environment and fail to generalize when deployed to a real environment
- Too Perfect Sensor Data - real world sensor data has quirks
    - Lighting simulation isn't perfect
    - GPS dropout/interference

2.4.6: Safety Considerations & Robustness

To account for all of these discrepancies, we will have to do two things. First, we will have to make our algorithms robust to account for environmental differences. Some of these differences should be automatically handled by the drone itself, but others we will have to account for ourselves. Second, and most importantly, we need to implement safety procedures and failsafes to prevent the drone from going haywall. There are two parts to this. For one, we will always allow a human to take over and stop the drone from whatever task it is currently performing. Secondly, we will have fail safes built directly into the drone to prevent unwanted data. For example, we will want to ensure the drone never leaves the operating zone. If it does,

we can program it to return to home. We will also implement a collision detection system that always runs and can overrun any behaviour if it detects a collision.

These safety systems can be thought of like the kernel of an operating system. They are very low level and capable of taking full control over the system. By default, they shouldn't do anything but if the situation calls for it, they leap into action to prevent unwanted behaviour.

2.4.7: Effective Library Building

We will have at least 3 projects going on: frontend, robotic simulation, and drone control. To handle all of them without all our codebases digressing into chaos, we will need to implement a lot of best practices for distributed systems: interfaces, shared libraries, and shared databases/applications.

A lot of the logic from the robot simulation and drone control code will be duplicated; for example, identical signature functions that will move the observer (drone or game camera). A good code practice is to define interfaces. As an example, the I*Observer* interface would have a *Move(newPosition)* function. This interface would be implemented by the *DroneObserver* and *CameraObserver*. By structuring the code like this, the FUEL algorithm doesn't need to know if it's dealing with a Camera or Drone. If you write the code to work with an *IObserver*, it should automatically work with both the drone and camera because the code is agnostic to what type of observer it is dealing with.

Shared libraries will also be very useful. We will need a place to put all these interfaces, dtos, data structures, and utility functions. By putting these in shared projects, we can be certain that we never deal with code drift; a model in 1 project changing but not another. Everything is consistent across projects which prevents issues.

We will also work with shared databases and applications. For example, our react frontend will expose an api that allows the observer to send a message to the frontend. It's fine if there is a parameter to specify where the message comes from, a drone or the game, but the API code itself is mostly agnostic to where the message is coming from. The same idea applies for the database and ROS code. ROS doesn't care where the sensor data is coming from. As long as you can provide it with the necessary data it can perform SLAM/FUEL.

The main concept with everything I've described is decoupling our code. Decoupling is the practice of designing software components so they can operate independently of each other, which improves flexibility, scalability, and maintainability. By enforcing clear interfaces and shared contracts (like IObserver), we ensure that each part of the system—whether it's the frontend, drone control, or simulation—can evolve without breaking the others. This makes the entire codebase easier to reason about and dramatically reduces the risk of inconsistent behavior across projects.