# Dedication

Anologment

Abstract

Cantent table

Figure table

Table list

# 1. Project Scope and Overview of Autonomous Drones Based on LiDAR

## 1.1 Introduction

Keeping public spaces like parks, college campuses, and city streets clean and safe is no small task. Manual inspections—where people walk around checking for litter or safety issues—are slow, expensive, and often miss hard-to-reach spots like dense bushes or rooftops. This is where drones, especially quadrotors, come in handy. They can zip over wide areas quickly, giving a bird’s-eye view perfect for tasks like spotting trash or patrolling. Our project dives into building a low-cost, customizable drone that uses LiDAR for navigation and mapping, aimed at making these tasks easier and more affordable. We’re focusing on a quadrotor with a Pixhawk 2.4.8 flight controller, tested in ROS and Gazebo simulations, with plans to add AI for waste detection later. This chapter lays out why we’re doing this, the tech we’re using, and what we hope to achieve, guiding you through the rest of the report.

## 1.2 Host Company

## 1.3 Review of Key Technologies for LiDAR-Based Autonomous Drones

Our surveillance drone is designed to patrol places like parks or campus buildings, spotting litter with a quadrotor that’s cheap and easy to tweak. LiDAR is the star here, helping the drone map its surroundings and dodge obstacles. This section breaks down the tech behind LiDAR-based navigation, giving you a sense of what makes our drone tick.

### 1.3.1 Principles of LiDAR Operation

LiDAR, or Light Detection and Ranging, works by firing laser pulses and timing how long they take to bounce back, building a 3D point cloud of the environment. It’s like a super-accurate radar with parts like a laser, scanner, and receiver. For our drone, LiDAR maps spaces in real-time, letting it weave around trees or desks without crashing. It’s quick and precise, making it perfect for autonomous systems [1].

### 1.3.2 LiDAR Technologies for Indoor Environments

Indoor LiDAR tech is tailored for tight spaces like labs or hallways, where our drone needs to navigate around furniture or walls without bumping into anything. These systems often use shorter-wavelength lasers—around 850 nm or 905 nm—because they’re great for short-range accuracy, usually up to 20–30 meters. They create detailed 2D maps by sweeping a laser in a single plane, which is enough for dodging obstacles in a room. They’re small, light, and don’t need much power, so they’re ideal for a lightweight quadrotor like ours, especially for indoor testing in a controlled lab.

A solid example of a 2D indoor LiDAR is the **RPLIDAR A1**. It’s a compact, spinning laser scanner that pumps out 2D point clouds at about 5.5 Hz, perfect for mapping small spaces. It’s cheap—great for budget projects like ours—and works well in dim or stable lighting, which we get in our lab. But it’s not built for outdoors. Sunlight can swamp the 905 nm laser, making it hard to read reflections, and it doesn’t handle weather like rain or fog, which scatter the beam. Glass walls or mirrors indoors can also throw it off if not calibrated. For our project, this kind of LiDAR is awesome for indoor sims and tests, but we wouldn’t trust it flying over a sunny campus lawn [2].



Figure 1 :RPLIDAR A1

### 1.3.3 LiDAR Technologies for Outdoor and Indoor Environments

Outdoor/indoor LiDAR tech is more versatile, built to handle both open fields and enclosed spaces, which fits our long-term goal of patrolling parks or campuses. These systems often use longer-wavelength lasers—around 1550 nm—to cut through sunlight and light weather like fog or drizzle. They can map out longer ranges making them great for big areas like a park. They’re typically 2D or 3D scanners, but even 2D versions pack more processing power to filter out noise from bright light or reflective surfaces. They’re a bit heavier and thirstier for power, but their flexibility makes them worth it for drones that need to work anywhere.

A good example of a 2D outdoor/indoor LiDAR is the **SICK LMS511**. This beast of a scanner uses a 905 nm laser but cranks up the signal processing to handle sunlight and weather better than indoor-only models. It’s got a range of up to 80 meters and can churn out 2D scans fast enough for real-time navigation. Indoors, it works just fine, mapping tight spaces with solid accuracy thanks to its noise-filtering tricks. Outdoors, it holds up against moderate sunlight and light rain, making it a fit for our eventual outdoor tests. [3].



Figure 2 : SICK LMS511

## 1.4 Study of Existing Autonomous Drone Solutions

To understand the landscape of LiDAR-based drones, we explored two well-known commercial options—the Skydio X10 and DJI Mavic 3E—to see how they measure up against our goal of crafting an affordable, customizable drone for monitoring public spaces like parks or campus grounds. These drones pack impressive features, but their limitations highlight the need for a low-cost, open-source alternative like ours. Below, we outline each drone briefly and evaluate their drawbacks in relation to our project’s priorities.

*Image Title: Commercial LiDAR-Based Drones in Action (Placeholder for Figure 1.4.1)*

### 1.4.1 Skydio X10

The Skydio X10 is a standout in autonomous drones, built for tough surveillance tasks. It uses LiDAR and multiple cameras to navigate complex environments, dodging obstacles like trees or benches with ease. Its software makes mission planning a breeze, letting users set flight paths or track objects, and it boasts a 45-minute flight time. This makes it a strong pick for professional operations, like patrolling busy public spaces[4] .



Figure 3 : Skydio X10 drone

### 1.4.2 DJI Mavic 3E

The DJI Mavic 3E is a more accessible commercial drone, tailored for mapping and inspections. It combines LiDAR with high-resolution cameras to create detailed maps and navigate reliably, especially outdoors. DJI’s user-friendly app simplifies flight planning, and its 45-minute battery life supports extended patrols. It’s a favorite for professionals needing a dependable, ready-to-fly solution [5] .

****

Figure 4 : DJI Mavic 3E

### 1.4.3 Comparative Evaluation

The Skydio X10 and DJI Mavic 3E excel in LiDAR-based autonomy, but their steep costs and closed-off software are dealbreakers for our project. The Skydio X10’s $10,000–$12,000 price is rough for small groups like city councils or researchers on tight budgets, and the DJI Mavic 3E’s $4,000 tag is still too high for scaling up public space monitoring. Worse, their proprietary software blocks customization—we can’t tweak it for our 2D LiDAR or ROS setup, or add AI for waste detection later. Our open-source platform offers the flexibility these drones lack, making it a better fit for our affordable, adaptable vision.

## 1.5 Problem Statement and Project Objectives

### 1.5.1 Problem Statement

Right now, there’s no affordable, flexible quadrotor platform for environmental monitoring that supports autonomy and easy upgrades. Commercial drones like the Skydio or DJI are either too pricey or locked down, while open-source options are often clunky to adapt for specific tasks like waste detection. This gap leaves municipalities, researchers, and small organizations without a practical solution for scalable surveillance.

### 1.5.2 Project Objectives

Our project aims to tackle this by:

* Building a low-cost quadrotor with a Pixhawk 2.4.8 for stable, autonomous flight, tested indoors with limited outdoor trials.
* Designing a modular platform that supports new sensors and future AI for waste detection (developed separately).
* Validating performance through ROS/Gazebo simulations and manual flight tests.

## 1.6 Project Methodology

Our goal was to build a Minimum Viable Product (MVP) for a surveillance drone that’s affordable, reliable, and ready for future tweaks like AI waste detection. We kicked things off by designing a lightweight quadrotor frame—sturdy enough to handle light outdoor tests but optimized for zipping around indoors, like in a lab or campus hall. The frame was rigged with DC motors, electronic speed controllers, a GPS module, telemetry for real-time data, and a Pixhawk 2.4.8 flight controller running ArduPilot firmware. Getting this setup dialed in took some elbow grease, but we ran manual flight tests to make sure the drone was stable and responsive before diving into anything fancy.

For autonomous navigation, we knew real-world testing was tricky—time constraints, safety concerns, and keeping costs low pushed us toward simulation. We used ROS and Gazebo to create a virtual quadcopter model equipped with our Hokuyo UTM-30LX 2D LiDAR, an IMU, and a barometer. This setup let us mimic real-world scenarios without risking crashes. We developed a layered control system with PID controllers for position, velocity, and attitude, fine-tuning them in simulation to get the drone’s behavior just right. The LiDAR, paired with the IMU and barometer, fed data into a pseudo-3D mapping approach, stacking 2D scans at different heights using Hector SLAM to map environments like a park or lab. We validated the autonomous navigation stack—waypoint.

## 1.7 Conclusion

This chapter kicks off our project to build a budget-friendly drone for keeping parks and campuses clean and safe. Using ROS 1 and Gazebo simulations was pretty handy, letting us test autonomous navigation safely and cheaply while fine-tuning our 2D LiDAR setup for pseudo-3D mapping. The next chapters dive into the drone’s design, embedded systems .

*References*: [1] Pomerleau, F., et al., “A Review of Point Cloud Registration Algorithms for Mobile Robotics,” Foundations and Trends in Robotics, 2015. [2] Liu, J., et al., “Low-Cost LiDAR Systems for Indoor Robotics,” IEEE Robotics and Automation Letters, 2020. [3] Thakur, R., “Outdoor LiDAR Performance in Adverse Weather Conditions,” Journal of Field Robotics, 2022., 2018. [4] Skydio, “X10 Technical Specifications,” Skydio Official Documentation, 2024. [5] DJI, “Mavic 3 Enterprise Series User Manual,” DJI Official Website, 2024.

## 2. Embedded System Design

Crafting a surveillance drone to monitor public spaces like parks and campuses for litter detection calls for a robust, adaptable platform capable of supporting mission-critical equipment. We’ve selected components based on performance, cost-effectiveness, modularity, compatibility with ArduPilot and ROS Noetic, and reliability in the field. To assure our supervisors, we’ve honed the calculations, refined tables, included key resources, and added illustrative figures. Let’s break it down!

**2.1 Design Requirements for the Minimum Viable Product (MVP)**

To ensure the drone meets surveillance objectives, we established a structured set of engineering requirements, detailed in the table below. Table 2.1: MVP Design Requirements

|  |  |
| --- | --- |
| **Requirement** | **Specification** |
| Payload Capacity | Sufficient to accommodate mission-specific sensors and processing units |
| Flight Endurance | Minimum of 10 minutes at nominal load, outdoor hover at 70% throttle |
| Flight Stability | Thrust-to-weight ratio exceeding 2:1; hover accuracy within ±10 cm using GPS and IMU |
| Modularity | Design supports replaceable parts and adaptable sensor mounting |
| System Compatibility | Full integration with ArduPilot and ROS Noetic (MAVROS, SLAM, waypoint navigation) |

**2.2 Software Tools**

The development and operation of the surveillance drone rely on a comprehensive suite of software tools to enable flight control, autonomous navigation, simulation, and system integration. Each tool was selected for its robustness, compatibility with open-source platforms, and ability to support the project’s goals of affordability and modularity. Below, we provide a detailed description of each tool, highlighting its purpose, functionality, and critical role in the project.

**2.2.1 ArduPilot**

ArduPilot is an open-source autopilot software designed to control autonomous vehicles, including quadrotors, fixed-wing aircraft, and rovers. In this project, it serves as the core flight control software, managing low-level operations such as motor control, sensor fusion, and flight stabilization. ArduPilot supports multiple flight modes, including manual (Stabilize), GPS-assisted (Loiter), and fully autonomous (Auto), which are essential for transitioning from manual testing to autonomous surveillance missions. Its flexibility allows integration with various sensors (e.g., IMU, GPS, barometer) and communication protocols, enabling precise navigation and control.

The software’s open-source nature provides access to a global community for support, extensive documentation, and regular updates, reducing development time and costs. ArduPilot’s compatibility with the Robot Operating System (ROS) through the MAVROS package facilitates high-level autonomy, such as simultaneous localization and mapping (SLAM) and waypoint navigation, critical for litter detection tasks. Its ability to run on resource-constrained flight controllers ensures efficient performance, making it indispensable for achieving stable and reliable flight in both indoor and outdoor environments.

**2.2.2 Mission Planner**

Mission Planner is a ground control station (GCS) software used for configuring, monitoring, and testing autonomous vehicles running ArduPilot. It provides a graphical user interface for critical tasks, including firmware installation, sensor calibration, flight mode configuration, and mission planning. In this project, Mission Planner is essential for setting up the flight controller, calibrating electronic speed controllers (ESCs), radio transmitters, and onboard sensors (e.g., accelerometer, gyroscope), ensuring accurate flight behavior.

The software allows real-time monitoring of telemetry data, such as altitude, attitude, and battery status, during manual flight tests, enabling rapid identification of issues. Its mission planning feature supports the creation of waypoint-based flight paths, which are validated in simulation before real-world deployment. Mission Planner’s logging capabilities record flight data for post-flight analysis, aiding in performance optimization. Its user-friendly interface and compatibility with ArduPilot make it a vital tool for both development and operational phases, ensuring the drone meets stability and reliability requirements.

**2.2.3 Robot Operating System (ROS) Noetic**

The Robot Operating System (ROS) Noetic is a flexible, open-source middleware framework that provides tools and libraries for developing robotic applications. In this project, ROS Noetic serves as the backbone for high-level autonomy, managing tasks such as real-time mapping, path planning, and navigation. It operates on an onboard computer, enabling modular software development through a publish/subscribe communication model, where nodes (independent programs) exchange data via topics.

ROS Noetic’s extensive ecosystem includes packages for SLAM (e.g., Hector SLAM), which processes 2D LiDAR data to generate occupancy grid maps, and navigation stacks (e.g., move\_base) for global and local path planning. The MAVROS package bridges ROS with ArduPilot, allowing seamless communication between the flight controller and onboard computer for tasks like waypoint navigation and sensor data integration. ROS’s compatibility with simulation tools and visualization software enhances development efficiency by enabling virtual testing and debugging. Its stability and community support make it ideal for implementing complex autonomous behaviors required for public space monitoring.

**2.2.4 Gazebo**

Gazebo is a 3D robotics simulator that provides a realistic environment for testing robotic systems, integrating seamlessly with ROS. In this project, Gazebo is used to simulate the drone’s dynamics, sensors, and interactions with virtual environments, reducing the risks and costs associated with real-world testing. It models physical properties such as gravity, inertia, and drag, ensuring accurate representation of the quadrotor’s flight behavior.

Gazebo supports plugins for simulating sensors like LiDAR, IMU, GPS, and barometers, with configurable noise models to mimic real-world conditions. Through the gazebo\_ros package, simulated sensor data is published to ROS topics, allowing the navigation stack to process it as if from physical hardware. This enables end-to-end testing of autonomous navigation algorithms, including mapping, obstacle avoidance, and waypoint following, in scenarios like indoor labs or outdoor parks. Gazebo’s ability to replicate complex environments and its integration with ROS make it a critical tool for validating the drone’s performance before physical deployment.

**2.2.5 RViz**

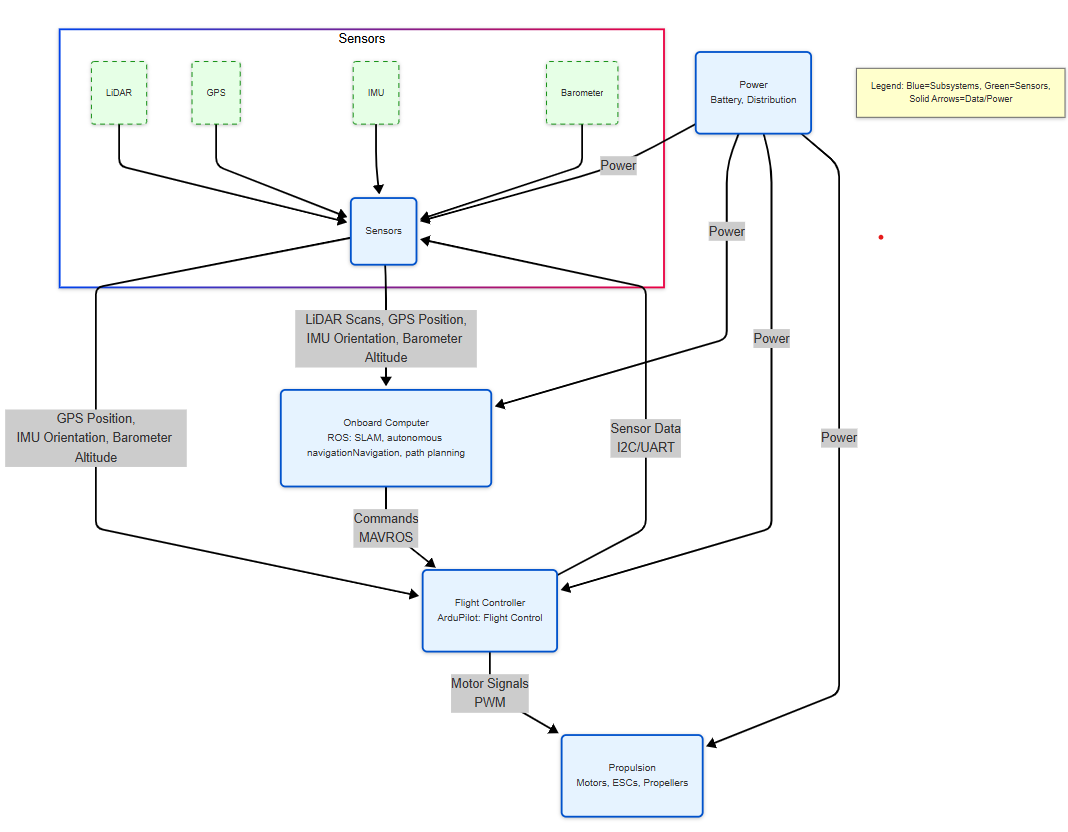
RViz is a 3D visualization tool within the ROS ecosystem, designed to display sensor data, robot states, and navigation outputs in real time. In this project, RViz is used for debugging and tuning the autonomous navigation stack by visualizing data such as LiDAR point clouds, occupancy grids, robot pose, and planned paths. It subscribes to ROS topics published by the drone’s sensors and navigation nodes, providing a graphical interface to monitor system behavior.

RViz’s ability to overlay multiple data streams (e.g., LiDAR scans, costmaps, trajectories) helps identify issues like mapping errors or path planning failures during simulations or real-world tests. Its interactive features allow developers to adjust visualization parameters, such as map resolution or sensor range, to optimize debugging. RViz’s integration with ROS and its role in providing actionable insights into the drone’s perception and navigation systems make it essential for ensuring robust autonomous performance.

**2.2.6 Summary of Software Tools**

The combination of ArduPilot, Mission Planner, ROS Noetic, Gazebo, and RViz forms a cohesive software ecosystem that supports the drone’s development from initial setup to autonomous operation. ArduPilot and Mission Planner handle low-level flight control and configuration, while ROS Noetic enables high-level autonomy. Gazebo and RViz facilitate safe, efficient testing and debugging, ensuring the system meets the project’s requirements for reliability, modularity, and performance in public space monitoring tasks.

**2.3 System Design**



**Figure 2.1: System Architecture Block Diagram**

The surveillance drone’s system design, shown in Figure 2.1, integrates five subsystems—Onboard Computer, Flight Controller, Sensors, Propulsion, and Power—to enable drone control and autonomous navigation . The Onboard Computer, running ROS, processes LiDAR scans, GPS position, IMU orientation, and Barometer altitude for SLAM, navigation, and data processing, sending commands to the Flight Controller. The Flight Controller, using ArduPilot, fuses sensor data to stabilize flight and controls the Propulsion subsystem’s motors, ESCs, and propellers for thrust. The Sensors subsystem, including LiDAR, GPS, IMU, and Barometer, provides data to both the Onboard Computer and Flight Controller for robust perception. The Power subsystem distributes battery energy to all components, ensuring reliable operation.

**2.4 Hardware Selection**

The hardware selection for the surveillance drone balances durability, performance, modularity, and cost to meet the requirements of sufficient payload capacity, minimum 10-minute flight endurance, thrust-to-weight ratio exceeding 2:1, hover accuracy within ±10 cm, and full integration with ArduPilot and ROS Noetic for autonomous navigation and litter detection. The following subsections detail the selection process for each subsystem—frame, propulsion, flight controller, GPS, LiDAR, onboard computer, and power—with comparisons and justifications tied to the project’s goals. The final component summary consolidates the selections, ensuring compatibility with the system design (Section 2.3).

**2.2.1 Frame Selection and Structural Evaluation**

The frame must balance durability, weight, and modularity for a 450 mm quadrotor. We compared three options in Table 2.2.

**Table 2.2: Frame Comparison**

| **Frame** | **Material** | **Weight (g)** | **Cost (DTN)** | **Pros** | **Cons** |
| --- | --- | --- | --- | --- | --- |
| DJI F450 | GFRP arms, polycarbonate | 282 | 98 | Durable, modular, cost-effective, integrated PDB | Less rigid than carbon fiber |
| S500 | Carbon fiber | 250 | 409 | Stiffer, low vibration | No PDB, less modular |
| 3D-Printed PLA | PLA plastic | 300–350 | - | Highly customizable | Weak tensile strength (50 MPa) |

We selected the DJI F450 frame for its optimal balance of durability, modularity, and cost. Its glass fiber-reinforced polymer (GFRP) arms (90 MPa tensile strength) withstand 1 m drops, unlike PLA’s fragile 50 MPa [6]. The integrated power distribution board (PDB) which reduces wiring complexity by it is significantly cheaper than the S500, aligning with cost-effectiveness while ensuring reliability in 0–40°C and 5 m/s winds.

**Figure 5: DJI F450 Frame with Modular Design and Integrated PDB**

*Resource*: [6] DJI, “F450 Frame Specifications,” DJI Official Website, 2023.

**2.2.2 Propulsion System: Motors and Propellers**

The propulsion system must deliver sufficient thrust, ensure energy efficiency, and maintain stability on the DJI F450 frame. We evaluated three motor-propeller combinations in Table 4.3.

**Table 4.3: Propulsion System Comparison**

| **Setup** | **Specs** | **Thrust (kg)** | **Weight (g)** | **Cost (DTN)** | **Pros** | **Cons** |
| --- | --- | --- | --- | --- | --- | --- |
| 1000 KV + 10×4.5” Props | 2212 motor, 30A ESC, 10×4.5 prop | 4.04 (4×1.01) | 280 (4×70) | 232 | Stable thrust, efficient (0.91 g/W) | Moderate current draw |
| 800 KV + 12×5.5” Props | 2216 motor, 40A ESC, 12×5.5 prop | 4.64 (4×1.16) | 320 (4×80) | 346 | High thrust | High current draw, low endurance |
| 1400 KV + 8×4” Props | 2208 motor, 20A ESC, 8×4 prop | 2.64 (4×0.66) | 240 (4×60) | 156 | Lightweight, cost-effective | Insufficient thrust |

We chose the 1000 KV 2212 motors with 10×4.5-inch propellers. Each motor delivers 1.01 kg thrust, yielding 4.04 kg total thrust for a 1.857 kg drone(section2.4.8), achieving a 2.18:1 thrust-to-weight ratio, exceeding the 2:1 requirement [7]. At 50% hover thrust (2.02 kg), power consumption is 757.6 W, supporting a 10-minute flight with the 4S 3500 mAh battery. The 800 KV setup’s high current draw reduces endurance, while the 1400 KV setup lacks sufficient thrust. The 1000 KV configuration ensures stability and efficiency, supporting the payload and modularity.

**Figure 7: 1000 KV 2212 Motor with 10×4.5-Inch Propeller**

*Resource*: [7] T-Motor, “2212 1000 KV Motor Datasheet,” T-Motor Official Website, 2023.

**2.2.3 Flight Controller Selection**

The flight controller must ensure precise navigation and integration with ArduPilot and ROS Noetic. We chose the Pixhawk 2.4.8, powered by an STM32F427 microcontroller, MPU6000 IMU, and MS5611 barometer, for its reliability and compatibility. It achieves ±10 cm hover accuracy, supports SLAM and waypoint navigation via ROS, and integrates with the F450 frame, 1000 KV motors, and battery [9]. At 70 g and $70, it offers cost-effective performance, with ArduPilot’s open-source community ensuring modularity and updates.

**Figure 8: Pixhawk 2.4.8 Flight Controller**

*Resource*: [9] ArduPilot, “Pixhawk 2.4.8 Setup Guide,” ArduPilot Documentation, 2023.

**2.2.4 GPS Module Selection**

The GPS module must provide accurate positioning for navigation. We compared three options in Table 4.7.

**Table 4.7: GPS Module Comparison**

| **Module** | **Accuracy (m)** | **Update Rate (Hz)** | **Cost ($)** | **Pros** | **Cons** |
| --- | --- | --- | --- | --- | --- |
| Neo-M8N | 1.5–2.0 | 10 | 30 | GPS+GLONASS, 10 Hz, compass | Not RTK-level accuracy |
| Neo-6M | 2.5–3.0 | 5 | 20 | Lightweight | Single constellation, low rate |
| Here+ RTK | <0.5 | 10 | 300 | High precision | Expensive, needs base station |

We selected the u-blox Neo-M8N for its 1.5–2 m accuracy and 10 Hz update rate, supporting ROS-based navigation and SLAM [10]. At 30 g and $30, it integrates with the Pixhawk 2.4.8 and F450 frame, ensuring modularity and cost-effectiveness. The Neo-6M’s lower rate and accuracy are insufficient, while the Here+ RTK is cost-prohibitive.

*Resource*: [10] u-blox, “NEO-M8N Datasheet,” u-blox Official Website, 2023.

**2.2.5 LiDAR Selection**

The LiDAR sensor must provide high-resolution 2D scans for SLAM and obstacle avoidance, supporting autonomous navigation. We compared three options in Table 2.3.

**Table 2.3: LiDAR Comparison**

| **Model** | **Range (m)** | **FOV (°)** | **Weight (g)** | **Cost ($)** | **Pros** | **Cons** |
| --- | --- | --- | --- | --- | --- | --- |
| RPLIDAR A2 | 12 | 360 | 190 | 100 | Lightweight, ROS-compatible, affordable | Limited range for outdoors |
| Hokuyo UTM-30LX | 30 | 270 | 370 | 1200 | Long range, high accuracy | Heavy, expensive |
| Slamtec RPLIDAR S1 | 40 | 360 | 220 | 600 | Long range, lightweight | Higher cost than A2 |

We selected the RPLIDAR A2 for its balance of performance, weight, and cost. With a 12 m range, 360° field of view, and 10 Hz scan rate, it supports ROS-based SLAM (e.g., Hector SLAM), meeting system compatibility requirements [11]. At 190 g, it fits within the payload capacity, and its $100 cost aligns with the low-cost goal. The Hokuyo UTM-30LX’s 370 g weight strains the thrust-to-weight ratio, and the RPLIDAR S1 is less cost-effective.

*Resource*: [11] Slamtec, “RPLIDAR A2 Datasheet,” Slamtec Official Website, 2023.

**2.2.6 Onboard Computer Selection**

The onboard computer must process sensor data for SLAM, navigation, and data analysis, running ROS Noetic. We compared three options in Table 2.4.

**Table 2.4: Onboard Computer Comparison**

| **Model** | **Specs** | **Weight (g)** | **Cost ($)** | **Pros** | **Cons** |
| --- | --- | --- | --- | --- | --- |
| Raspberry Pi 4 | 4 GB RAM, Quad-core 1.5 GHz | 46 | 55 | Lightweight, ROS-compatible, affordable | Limited GPU for AI tasks |
| NVIDIA Jetson Nano | 4 GB RAM, Quad-core 1.4 GHz, 128-core GPU | 140 | 100 | GPU for AI, ROS-compatible | Heavier, higher power draw |
| Intel NUC Mini PC | 8 GB RAM, Dual-core 2.3 GHz | 400 | 300 | High performance | Too heavy, expensive |

We selected the Raspberry Pi 4 (4 GB RAM) for its lightweight design, ROS Noetic compatibility, and cost-effectiveness. It runs Ubuntu 20.04 and ROS packages (e.g., move\_base, Hector SLAM), meeting system compatibility requirements [12]. At 46 g and $55, it fits within the payload capacity and budget, integrating with the Pixhawk 2.4.8 and F450 frame. The Jetson Nano’s GPU is advantageous for future AI tasks but is heavier, and the Intel NUC is too heavy.

*Resource*: [12] Raspberry Pi, “Raspberry Pi 4 Specifications,” Raspberry Pi Official Website, 2023.

**2.2.7 Power System: Battery and Power Management**

The power system must sustain flight and power all subsystems, including the RPLIDAR A2 (5V, 1.5A), Raspberry Pi 4 (5V, 3A), and Pixhawk 2.4.8 (5V, 0.5A), while ensuring modularity and cost-effectiveness. We selected the 4S 3500 mAh LiPo battery (14.8V, 30C) and a 5V 6A DC-DC buck converter to meet these needs. The battery supports the 1.857 kg drone, consuming 757.6 W at hover (Section 2.2.2). With 51.8 Wh capacity (14.8V × 3.5 Ah) and 70% usable energy (36.26 Wh), it provides sufficient endurance [8]. The additional power draw—7.5W (LiDAR), 15W (Pi), 2.5W (Pixhawk), and ~2.5W converter loss (90% efficiency)—totals 27.5 W, yielding 785.1 W. Flight time is (36.26 Wh / 785.1 W) × 60 = 11.58 minutes, exceeding the 10-minute requirement. The buck converter (12 g, $6), connected to the DJI F450’s PDB 12V output with a 7A fuse, supplies stable 5V to all electronics via standard connectors (JST, USB-C, Pixhawk pins), ensuring modularity. At 350 g ($35) for the battery and 17 g ($8) for the converter and wiring, the power system supports all components cost-effectively, as shown in Figure 2.1’s Power subsystem.

*Resource*: [8] EEMB, “LiPo Battery Care Guide,” EEMB Battery Manufacturer, 2023.

**2.2.8 Final Component Summary**

**Table 2.5: Final Component Overview**

| **Subsystem** | **Component** | **Specs / Key Features** | **Weight (g)** | **Cost ($)** | **Justification** |
| --- | --- | --- | --- | --- | --- |
| Frame | DJI F450 | GFRP arms, integrated PDB | 282 | 50 | Modular, durable, cost-effective |
| Motors | 1000 KV, 2212 | 4× motors, 1.01 kg thrust each | 280 (4×70) | 60 | Efficient, meets thrust needs |
| Propellers | 10×4.5" self-tightening | Nylon, crash-resistant | 20 (4×5) | 10 | Efficient lift, durable |
| ESCs | 30A BLHeli (Kit) | 4× ESCs, 2–4S, BLHeli 14.9, DSHOT600 | 25 (4×6.25) | 40 | Lightweight, supports 4S and motors |
| Battery | 4S 3500 mAh LiPo | 14.8V, 30C, 11.58 min flight | 350 | 35 | Balances endurance and weight |
| Power Management | 5V 6A Buck Converter | 5V output for LiDAR, Pi, Pixhawk, 90% efficiency | 17 (12+5) | 8 | Stable power for electronics |
| Flight Controller | Pixhawk 2.4.8 | ArduPilot/ROS support, sensor fusion | 70 | 70 | Reliable, community-supported |
| GPS | u-blox Neo-M8N | 10 Hz, 1.5–2 m accuracy, compass | 30 | 30 | Accurate, ROS-compatible |
| LiDAR | RPLIDAR A2 | 12 m range, 360° FOV, ROS-compatible | 190 | 100 | Lightweight, affordable, supports SLAM |
| Onboard Computer | Raspberry Pi 4 | 4 GB RAM, ROS Noetic, SLAM/navigation | 46 | 55 | Lightweight, cost-effective, ROS-compatible |
| Misc. (wiring, mounts) | Generic | Connectors, mounts (50 g wiring, 50 g mounts) | 100 | 10 | Standard components |

**Total Weight**: 282 + 280 + 20 + 25 + 350 + 17 + 70 + 30 + 190 + 46 + 100 = 1410 g  
**Total Cost**: $50 + $60 + $10 + $40 + $35 + $8 + $70 + $30 + $100 + $55 + $10 = $468

The selected hardware, with the buck converter powering the LiDAR, Raspberry Pi, and Pixhawk, ensures a lightweight, modular design with sufficient thrust (2.18:1 ratio), endurance (11.58 minutes), and compatibility with ArduPilot and ROS Noetic, supporting autonomous surveillance tasks cost-effectively.

## 3. Autonomous navigation simulation

Using ROS and Gazebo, we built a virtual quadcopter drone, complete with a Hokuyo UTM-30LX 2D LiDAR, IMU, and barometer. This setup let us test navigation in realistic scenarios without a single crash. We crafted a layered control system with PID controllers to nail down position, velocity, and attitude, while stacking 2D LiDAR scans into pseudo-3D maps via Hector SLAM. This chapter dives into how we simulated, tuned, and validated our autonomous navigation stack to ensure rock-solid performance for litter detection missions.

## 3.1 Simulation Stack Overview (ROS, Gazebo, RViz)

3.1.1 ROS

ROS is a middleware that connects different parts of a robot system using a publish/subscribe model. We chose ROS 1 Noetic because it is the final and most stable ROS 1 release, widely supported in the community. We used key ROS elements like roscore (the central manager), nodes (modular programs), and topics (data channels) to structure our drone's control, mapping, and navigation systems.

A black letter on a white background

AI-generated content may be incorrect.

Figure 9 : ROS

3.1.2 Gazebo

Gazebo is a 3D robotics simulator that integrates seamlessly with ROS. It allowed us to test our drone in a realistic environment, including physics, gravity, and sensor noise. Through the gazebo\_ros interface, Gazebo published simulated sensor data directly to ROS topics, enabling our system to behave as if it were running on a real drone.

### 3.1.3 RViz

RViz is a 3D visualization tool for ROS. We used it to view live sensor data, robot pose, planned paths, and occupancy grids in real time. This was essential for debugging and tuning the navigation stack. RViz subscribed to the same topics used by the drone, helping us verify that the system responded correctly to the environment and navigation goals.

## 3.2 Drone Model and Sensors

he simulated drone was modeled as a compact quadrotor, designed to replicate the dynamics and sensor layout of a lightweight aerial platform used for indoor and outdoor navigation. It featured a four-rotor configuration with accurate mass and inertia properties, allowing for realistic flight behavior within the Gazebo physics engine. The drone's frame was integrated with virtual mounting points for onboard sensors, ensuring proper alignment and data accuracy during motion and rotation.

A black drone with four round propellers

AI-generated content may be incorrect.

Figure 10 : drone urdf visualization in RVIZ

To simulate perception and navigation capabilities, we equipped the drone with four key virtual sensors using standard ROS-Gazebo plugins:

| **Sensor** | **Plugin Used** | **Characteristics Simulated** | **Notes** |
| --- | --- | --- | --- |
| IMU | libgazebo\_ros\_imu | Acceleration, rotation, realistic noise | Matches MPU6000 sensor’s noise profile |
| Barometer | libgazebo\_ros\_barometer | Altitude based on air pressure | Tuned to emulate MS5611 performance |
| LiDAR (2D) | libgazebo\_ros\_laser | Distance measurements, field of view, update rate | Simulates Hokuyo UST-10LX behavior |
| GPS | libgazebo\_ros\_gps | Position coordinates, signal drift | Includes realistic GPS noise and drift |

Each sensor was configured to closely mimic real-world hardware behavior, introducing noise, delay, and environmental effects where applicable. These data streams were published to ROS topics in real time, feeding into our mapping, pose estimation, and navigation pipelines for a fully autonomous simulation loop.

## 3.3Motion and Control System Design

### 3.3.1 Overview of Drone Motion in Gazebo Simulation

In order to enable precise and stable flight within the simulated Gazebo environment, I developed a complete motion control architecture tailored for the Hector Quadrotor drone model. This system was designed from the ground up using a layered PID-based control strategy, which I implemented in three core C++ nodes—each responsible for a specific aspect of the drone's motion control: position, velocity, and attitude.

The motion of the drone in the simulated world is governed by a realistic physics engine that models the effects of gravity, inertia, and applied forces. To interact with this environment, my controllers do not directly set positions or orientations; instead, they publish physical forces and torques as wrench commands, which are interpreted by the Gazebo simulation plugin responsible for applying them to the drone’s virtual body. This setup allows the simulated drone to behave in a way that closely resembles a real quadrotor operating in the physical world.

### 3.3.2Development of a Multi-Layered Control Architecture

A diagram of a system

AI-generated content may be incorrect.

Figure 11 : Multi-layered Control architecture

The motion control system I built is structured as a hierarchical architecture with three interconnected layers. At the top of this hierarchy is the **position controller**, which computes how the drone should move in space. The second layer is the **velocity controller**, which interprets those motion intentions and translates them into desired orientations and thrust levels. Finally, the **attitude controller** is responsible for applying the correct forces and torques to physically realize those commands within the simulator.

This modular structure reflects the natural division of responsibilities in quadrotor control: the upper layers focus on high-level trajectory and motion objectives, while the lower layers manage fast dynamic responses. By isolating control tasks into distinct modules, each tuned with independent PID controllers, the system achieves both flexibility and robustness.

### 3.3.3Position Control: Navigating Through Spatial Goals

The position controller is the first layer in our drone’s control stack. Its job is to move the drone toward a target coordinate in 3D space by generating a velocity command. It does this by comparing the current position (from odometry) to the desired goal, then feeding the position error along each axis (x, y, z) into three separate PID controllers.

For example, for the x-axis:

A black and white text

AI-generated content may be incorrect.

This gives us how fast the drone should move along x to correct its position. The same is done for y and z. This layer only outputs desired velocities — not angles or forces. It leaves that to the lower layers to figure out how to physically move the drone.

### 3.3.4 Velocity Control: Generating Attitude and Thrust Commands

This layer takes the desired velocity vector from the position controller and tries to make the drone match it. It compares the desired and current velocities and calculates how the drone should tilt and how much thrust to apply.

Since drones move forward by pitching and sideways by rolling, the velocity error is converted into an acceleration command using PID. Then, using basic physics, we estimate the required orientation:

A close up of a sign

AI-generated content may be incorrect.

Here, g=9.81 m/s2 is gravity, and ax,ay\_x, a\_yax​,ay​ are the accelerations needed in x and y. For vertical movement, we apply a PID controller to control vzv\_zvz​ by adjusting upward thrust.

The final output of this layer is:

* Desired roll (rad)
* Desired pitch (rad)
* Desired yaw rate (rad/s)
* Desired upward thrust (N)

These are passed to the attitude controller.

### 3.3.5 Attitude Control: Computing Wrench Commands

The attitude controller is the final layer. It takes the desired orientation and yaw rate, compares them to the drone’s current state, and calculates the torque and force needed to reach them.

Each axis (roll, pitch, yaw) is controlled using PID:



Similar equations are used for pitch and yaw. These torques, along with the vertical thrust, are packaged into a geometry\_msgs::Wrench message:

* force.z — upward thrust
* torque.x — roll torque
* torque.y — pitch torque
* torque.z — yaw torque

This message is published to /command/wrench, and Gazebo applies it to simulate real motion. The drone then moves according to the resulting forces and torques.

### 3.3.6 Force Application and Physical Simulation in Gazebo

Once the wrench command is received and applied by the Gazebo plugin, the resulting behavior of the drone reflects the laws of physics modeled within the simulation environment. Gazebo accurately simulates inertia, drag, gravity, and rotational dynamics, so the applied forces and torques result in corresponding linear and angular accelerations. This physical feedback is critical for closing the control loop: as the drone moves, its new position, velocity, and orientation are sensed and fed back into the control stack, allowing real-time adjustments to maintain stability and trajectory tracking.

## **3.4 SLAM and GPS-Augmented Autonomous Navigation**

3.4.1 Real-Time Mapping Using 2D LiDAR

Mapping is performed through the Hector SLAM framework, chosen for its compatibility with aerial robots that lack wheel odometry. Hector SLAM combines scan matching from the 2D LiDAR with high-frequency inertial data from the IMU to estimate the drone’s pose with high accuracy. This setup eliminates the need for external localization systems.

The LiDAR continuously emits a dense array of laser beams in a horizontal circle, capturing a real-time cross-section of the environment. Each scan is aligned with the current map using Gauss-Newton-based scan matching. Between successive scans, the IMU provides motion updates, ensuring robust pose estimation even under rapid flight dynamics.

As a result, the system publishes a dynamically updated occupancy grid (/map) and a precise pose estimate (/pose), both of which are consumed by the navigation stack for path planning and localization.

Figure 9.2: Real-Time Mapping with Hector SLAM — 2D laser scans fused with IMU data to maintain an accurate map and pose.

**3.4.2 Costmap Generation and Spatial Awareness**

To enable path planning and obstacle avoidance, the system constructs two layered costmaps: a global costmap, derived from the full SLAM-generated map, and a local costmap, built in real time from the latest LiDAR scans.

The global costmap represents the known environment, highlighting static obstacles and boundaries. It serves as the planning space for high-level goal-directed motion. In contrast, the local costmap is centered around the drone and refreshes continuously to account for dynamic obstacles. It reflects the immediate surroundings and is used for fine-grained, short-term trajectory adjustments.

Each costmap encodes the environment into a 2D grid, where cell values indicate the cost of traversal. Free spaces are marked with low values, while obstacles receive high costs. A decay-based inflation layer ensures that the drone maintains a buffer zone around obstacles, promoting safer navigation.

Figure 9.3: Costmap Layering — Global costmap (static environment), local costmap (dynamic surroundings), and inflation zones.

**3.4.3 Path Planning with Global and Local Planners**  
Path planning is divided into two stages: long-range path generation and local trajectory optimization. For global planning, the system employs either the NavFn or GlobalPlanner plugin, both of which implement variants of Dijkstra’s or A\* algorithm. These planners compute the optimal path from the current pose to the target location, minimizing cost over the static global costmap.

Local path planning is handled by DWAPlannerROS, an implementation of the Dynamic Window Approach. This planner simulates a series of velocity commands and evaluates them based on obstacle proximity (from the local costmap), alignment with the global path, and the drone’s dynamic constraints. The command that best balances safety, efficiency, and path-following accuracy is selected for execution.

Together, these planning modules ensure the drone navigates smoothly and safely through structured indoor environments, as long as all obstacles lie within the 2D LiDAR’s scan plane.

Figure 9.4: Path Planning Workflow — Global planner (goal-driven path), local planner (obstacle-aware command selection).

**3.4.4 GPS Integration for Outdoor Navigation**  
To expand the drone's navigation capability beyond SLAM-mapped indoor spaces, GPS data is integrated into the localization and goal-setting framework. The system uses the navsat\_transform\_node from the robot\_localization package to convert global GPS coordinates (latitude, longitude, altitude) into a local Cartesian frame aligned with the SLAM map.

By fusing GPS with IMU and barometric data, the node publishes a consistent local pose, enabling the drone to understand its position relative to a fixed world origin. When a goal lies outside the current SLAM map, its GPS coordinates are transformed into local coordinates and passed to the global planner. This allows the drone to autonomously explore and expand its map as it moves toward the GPS-defined goal.

This hybrid localization setup ensures continuity of navigation across indoor and outdoor environments, enabling long-range missions that bridge SLAM-mapped areas with GPS waypoints.

**3.4.5 Limitations of 2D Navigation**  
Despite its effectiveness in flat environments, 2D LiDAR-based navigation has notable limitations in complex 3D indoor settings. Vertical obstacles that lie outside the horizontal scan plane — such as overhanging furniture, suspended signs, or half-open windows — are effectively invisible to the drone. Consequently, vertical gaps that are actually navigable may be misrepresented as blocked, and navigation failures can occur.

One common failure mode is the stuck condition, in which the drone receives valid velocity commands but fails to move, often due to occlusions or ambiguous map representations. In such cases, the navigation system can enter a loop of re-planning without progress.

To address this, a fallback strategy was devised based on vertical exploration and reconstruction of the immediate 3D environment, using only the onboard 2D LiDAR and barometric altitude estimates.

**3.4.6 Pseudo-3D Recovery Strategy**  
i. Stuck Detection Logic  
The transition into recovery mode is triggered when the drone detects a persistent navigation failure. This is determined by monitoring velocity commands, position updates, and changes in the costmap. If the drone receives motion commands but its position remains static, and if the map appears unchanged over time, the system identifies a stuck condition and activates the pseudo-3D recovery strategy.

ii. Local 3D Map Construction via Vertical Exploration  
The recovery system initiates a controlled vertical exploration. The drone gradually ascends in discrete steps — typically 20 cm — pausing at each level to collect a complete horizontal LiDAR scan. Upon reaching a predefined ceiling (e.g., 2.5 meters), it begins a similar descent, capturing scans at each altitude.

Each scan is timestamped and tagged with the corresponding altitude, estimated from a fusion of barometric and inertial data. The result is a sequence of horizontally aligned 2D scans, each at a different height. These scans are then projected into 3D space by assigning Z-values to each scan point, producing a sparse but informative vertical representation of the nearby environment.

Figure 9.5: Pseudo-3D Mapping — Stacked LiDAR scans at multiple altitudes forming a coarse 3D voxel map.

iii. Recovery Planning with 3D Map  
With the local 3D voxel map constructed, the system searches for vertical corridors or gaps that may serve as escape routes. The recovery planner evaluates potential trajectories that involve ascending or descending while avoiding detected 3D obstacles.

If a feasible path is found, the drone follows the escape trajectory until it reaches a more navigable region. Once free from the obstruction, the system re-engages the standard 2D planner and resumes goal-directed navigation. If no escape path can be identified, the drone escalates to a higher-level fallback behavior, such as returning to the previous waypoint or awaiting external intervention.

Figure 9.6: Recovery Behavior — Vertical escape planning based on 3D scan stacking and obstacle avoidance.

iv. Benefits of a Layered Navigation System  
By combining efficient 2D navigation with a robust 3D recovery strategy and outdoor GPS integration, the system strikes a balance between simplicity, adaptability, and resilience. The 2D layer ensures low-latency path planning and real-time responsiveness, while the pseudo-3D layer empowers the drone to recover from vertically complex or ambiguous scenarios.

The addition of GPS allows the drone to operate seamlessly across mixed environments, from structured indoor spaces to unstructured outdoor zones, without requiring expensive 3D sensors. This layered approach significantly extends the autonomy and mission flexibility of the aerial system.

### 3.5 Autonomous flight test

# Conclusion and Strategic Path Forward

# Future Work