Faculty of Engineering and Applied Science
Department of Automation and Mechatronics

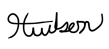
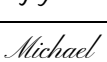
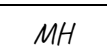
Capstone Final Engineering Report

Group: G06-2-6
April 9, 2021

Unmanned Aerial Vehicle for Structural Firefighting with Simultaneous Localization and Mapping

ENGR 4951U - 001

Supervisor Name: Dr. Carlos Rossa, PhD, B. Eng, MSc
Course Instructor: Dr. Remon Pop-Iliev, PhD, P. Eng, FCSME

Group Members		
Connor Jong	100621435	
Hudson Hulcoop	100621819	
Joshua John	100589460	
Joyson Sebastiampillai	100617006	
Michael Currie	100622176	
Myles Hunt	100623925	

Abstract

This paper will provide an in-depth look at the development of an unmanned aerial vehicle (UAV) that can be used in firefighting scenarios to increase efficiency of operations and lower the risk to emergency personnel. This UAV has robust vision and simultaneous localization and mapping (SLAM) capabilities, allowing it to create a 3D map, mark the location of victims, and autonomously navigate through the harsh environment of a burning building. All of this information is relayed to a base station for firefighters and emergency response teams to view. This will allow firefighters to better understand the situation before entering. The firefighters will have key information about potential dangers, the current state of the building's interior, and the location of trapped individuals.

In order to accomplish these tasks, a variety of specifications and requirements needed to be met, including robust vision for smoke-filled and low-lit environments, heat resistance, human detection, and 3D mapping capabilities. After a review of the current state of the art, it became apparent that nothing on the market is able to satisfy this criteria. Considering this, the project was approved for development. The final design of the UAV features a modified Parrot AR Drone 2.0 as the base. It has been equipped with a Raspberry Pi 4 as the onboard processor, an Intel Realsense D435i camera for infrared stereo vision, and a 2200 mAh battery to accommodate the new components. In terms of software, the current system is made up of various components. For the 3D mapping software, Kimera was chosen for its ability to turn a point cloud into a human readable map. In order to navigate this map, a modified dynamic window approach (DWA) path planner was utilized as the autonomous path planning program. Finally, a custom human detection program, which uses artificial intelligence, was created to identify any humans encountered while mapping and mark their location on the generated map. The current prototype is fully assembled and all aforementioned systems are fully functional. Some aspects of the design were not fully tested due to COVID-19 and can be improved upon, such as high temperature resistance and the inclusion of a pressure sensor.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

ABS Acrylonitrile Butadiene Styrene

AirSim Aerial Informatics and Robotics Simulation

BGR blue-green-red

CAD Computer Aided Design

CPU Central Processing Unit

DWA Dynamic Window Approach

FEA Finite Element Analysis

FOV Field of View

FPS Frames Per Second

GPS Global Positioning System

IMU Inertial Measurement Unit

IR Infrared

PCB Printed Circuit Board

ROS Robot Operating System

RPGO Robust Pose Graph Optimization

SLAM Simultaneous Localization and Mapping

UAV Unmanned Aerial Vehicle

UGV Unmanned Ground Vehicle

VIO Visual Inertial Odometry

VM Virtual Machine

VO Visual Odometry

1 Introduction

The market for unmanned aerial vehicles (UAV) has grown exponentially since 2015, and is expected to exceed a market cap of 120 billion USD by 2025 [1]. UAVs are typically small aircrafts that operate without a pilot. They can be autonomous or operated via remote control. The primary appeal of UAVs, compared to unmanned ground vehicles (UGV), is their dynamic movement capabilities, being able to move freely in three dimensions. This allows them to efficiently traverse environments that would pose issues for UGVs, allowing for a wider variety of applications. These applications can range from recreational photography to 3D mapping of large environments for military operations. As this technology continues to be further developed, UAVs are becoming a more valid solution to many real world challenges. An example of this would be applications where they would be implemented to assist people who work in dangerous environments. Engineers are currently developing methods to automate these tasks in order to avoid individuals taking unnecessary risks. Although the current state of technology is capable of being adapted for UAVs for the aforementioned applications, they are still early in the development stages and have not yet been implemented for all applications. For these reasons, it is beneficial to develop a solution which utilizes autonomous UAVs to increase the safety and efficiency of emergency operations.

2 Design Problem and Objectives

2.1 Project Scope

In a matter of a few minutes, a small fire can grow to the point of engulfing an entire building in flames [2]. It is the duty of firefighters to do their best to mitigate the damage and stabilize the situation. However, time is not always on their side, fires can quickly spiral out of control, resulting in irreparable damage and potential lives lost. In an effort to mitigate these damages, the goal of this project is to increase the overall efficiency of fire fighting response operations. Often, firefighters will have to search through chaotic environments, without substantial information regarding the locations of victims or hazardous areas. The longer they spend in these dangerous environments, the higher

chance there is for health deterioration due to high temperatures and inhalation of smoke or dust. Knowing critical information, such as hazard and victim locations, will allow the firefighters to determine high priority areas that require immediate attention from crew members. This will significantly cut down on total operation time, thus minimizing the risk to firefighters and increasing the likelihood of rescuing trapped victims.

The proposed solution is to develop a UAV that will enter the burning building prior to the entrance of the emergency response personnel. The UAV will search for potential victims while simultaneously generating a 3D map of the environment. The generated 3D map would be relayed back to the base station to show the locations of trapped victims and potential danger zones. This map would allow firefighters to plan the safest route to trapped individuals and reduce the overall time spent in the building.

2.2 Project Related Background

Firefighters are an integral part of modern societies' emergency response personnel. They are often the first to arrive at the scene of an emergency and are trained to handle a wide variety of situations. However, when entering the hostile environment of a burning building, no amount of training can guarantee the safety of the firefighters who enter. Every second counts when lives are on the line. Two minutes is all that is required for a fire to become life-threatening, and five minutes for a residence to be completely engulfed in flames. Heat from fires can rise up to 1000℃, and at these temperatures the heat alone can scorch lungs and cause worn clothes to become melted or fused onto skin. Visibility can be reduced to a few feet due to the thick smoke created by the flames, causing the environment to be extremely difficult to navigate [2]. That is why it is imperative to provide firefighters with the necessary information they need to be able to enter the building, take the fastest and safest route to trapped victims, and escort them outside safely. The proposed method to gather and relay this information is to send in a UAV before the emergency personnel enter the building.

UAVs are highly mobile and can enter from many openings located on the upper floors of buildings, allowing for quick assessment of the situation. UAVs are also modular and can be equipped with different sensors to obtain critical information about the environment, such as the temperature and atmospheric pressure of each room. These qualities are what make UAVs excellent candidates for simultaneous localization and mapping (SLAM).

2.2.1 Simultaneous Localization and Mapping (SLAM)

SLAM is a technology that enables a vehicle to detect its surroundings, create its own virtual map, and estimate its position within the map [3]. SLAM can be further broken down into the following two categories: localization, and mapping.

Localization: Localization is the part of the software that determines the UAVs position, using data collected via sensors. If the location is unknown, the UAV would use its sensors to collect information about its surroundings and then map its environment. Localization must also determine the vehicle's direction of motion, in order to track its position relative to the map [3].

Mapping: Utilizes the data gathered from sensors to create a virtual map that allows both the UAV and operator to view its surroundings [3]. In most cases, this map consists of a collection of dots that represent the location of objects identified by the sensors. This collection of dots is commonly referred to as a point cloud.

SLAM systems used in UAVs commonly utilize cameras to gather environmental data. The problem with regular cameras is that in a burning building, the fire and smoke will obscure the camera's vision, significantly inhibiting the effectiveness of traditional 3D mapping. In order to combat this issue, the addition of infrared (IR) sensors will be explored.

Some additional challenges for UAVs that arise during firefighting scenarios would be locating survivors, withstanding high temperatures, and maneuvering around various obstacles present inside buildings. Making use of visual inertial odometry (VIO) with SLAM can enable the detection of these obstacles, allowing the UAV to path around them.

2.2.2 VIO

Before explaining visual inertial odometry, it is first necessary to explain the difference between visual odometry (VO) and VIO. VO gathers information only from cameras, whereas VIO gets information from cameras and an inertial measurement unit (IMU) [4]. An IMU is a group of sensors that collect data related to a device's movement. They are typically comprised of a 3-axis accelerometer and a 3-axis gyroscope however, they can also include a 3-axis magnetometer [5]. Utilizing an IMU improves the accuracy of the data collection process, and the overall quality of SLAM.

One of the main applications for VIO is to allow a SLAM program to gather information about the environment with respect to the vehicle's location. More specifically, it is the process of determining the pose (orientation and location) and velocity of the drone [4]. When combined, these readings can be processed to track the drone throughout the map it creates. This is why VIO is an essential part of camera based SLAM. Without it, the vehicle would be unable to perceive its environment or location. With the implementation of a suitable SLAM program, a UAV will be able to map its surroundings, localize itself with this map, and detect potential obstacles. In order to navigate around these obstacles, an appropriate pathing planner would need to be paired with the SLAM program.

2.2.3 Autonomous Path Planning

Pathing Algorithms

A pathing algorithm's purpose is to find a route from a start location to the designated end goal. There are various ways pathing algorithms can perform this task. Most common pathing algorithms use nodes and grids with a nodal analysis pattern. A node is a specific point within the grid which can either be traversable space or an obstacle. The grid is the environment that needs to be explored and can be two or three dimensional. These grids are weighted, which means when moving from one node to another, they can have different distance values. These weights help the algorithms determine the shortest path to the end goal. Some algorithms will explore nodes in all available directions, while others utilize heuristics for faster computation. The resulting path depends on the algorithm selected. Some output non-smooth, longer routes, while others smooth out their paths at the cost of higher computation times [6].

A heuristic is an equation utilized as a shortcut when trying to find a route to the end goal. For pathing algorithms, the heuristic equations are a comparison of distance between the end and current node being surveyed. If the distance calculated decreases when moving to the next node, the pathing algorithm knows it is moving in the right direction [7].

Frontier Exploration

Frontier exploration is a method of setting goals on a map to be used for path planning. A frontier is an array of points on a 2D or 3D map that are marked as unknown. This array must be a continuous line with no explored points intersecting it. The minimum or maximum size of this array varies depending on the requirement of the path planner. If the robot being used is smaller, the frontier size could also be smaller. Likewise, larger robots will have a larger minimum size for their frontiers. An example of a frontier on a two dimensional map can be seen in Figure 1 [8]. The number of frontiers on a map is determined by the current map state and the preset frontier characteristics like length for example.

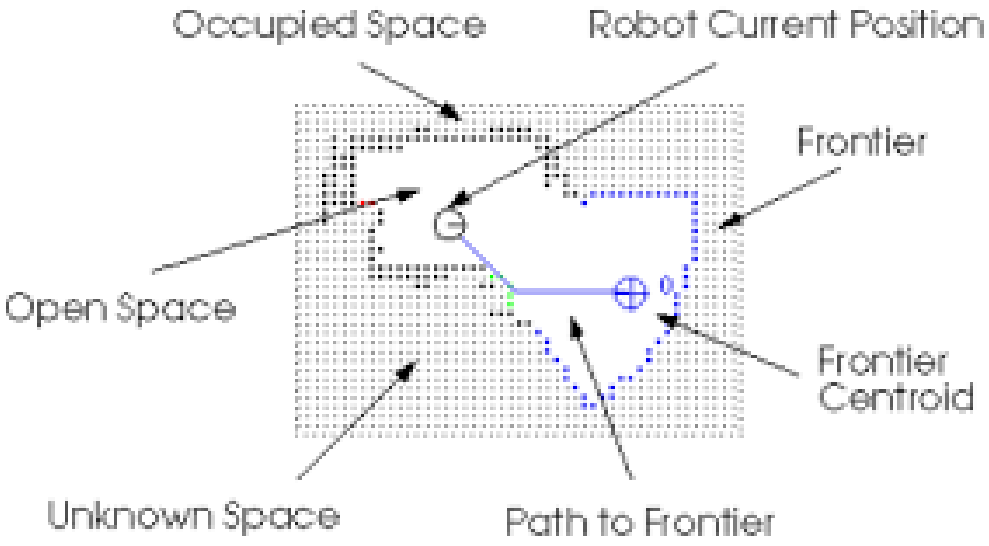


Figure 1: Frontier Exploration Example

Costmaps

A costmap is a tool that is used during path planning to pick the most optimized path that is viable. A costmap is a type of occupancy grid which stores values on a scaled range, for example 0 to 100. The higher the stored value of the point means a higher cost for moving through that point. Cost goes up when a point is closer to an obstacle or wall. These values are used by a path planner to pick a path to a set goal by optimizing it between cost and the shortest route. This is valuable since a short path may be less optimal since it could get too close to a wall causing a collision or slower speeds [9].

DWA Planner

The program dwa_local_planner is a pathing program that will create a path connected from a mobile base to the set goal on a map. This path will be a kinematic trajectory from the current location to the goal position and orientation. The dwa_local_planner will use a costmap to evaluate the costs of projected paths in the built in pathing algorithm. The controller part of the program will determine the dx, dy, and theta velocities to send the mobile base from these trajectories (Figure 2) [10].

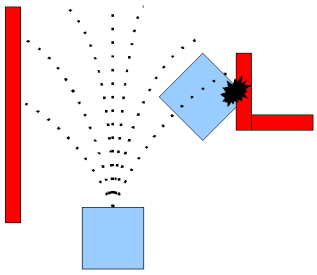


Figure 2: DWA Planner Trajectory Testing

2.2.4 Human Detection

In an emergency fire situation, it is essential that firefighters are able to search for and rescue as many victims as possible. In order to assist firefighters with locating survivors, a human detection program was developed. Human detection is the art of analyzing and determining facial and body features from frames of video that are being processed by the central processing unit. To ensure that these

features are properly analyzed and detected, OpenVIN's mobilenet-ssd is utilized to train facial and body data sets for detection. Mobilenet-ssd is a single-shot multibox detection system that inputs a blob of data consisting of a single image in 1x3x300x300 blue-green-red (BGR) order.

Each model of data consists of an array of five detection criteria: image_id, label, conf, minimum coordinates, and maximum coordinates. The image_id consists of the identification number or series of characters that identify a singular image in the dataset. The label is a placeholder for the predicted class identification information that is found by the detection training system. The "conf" consists of a numerical value for the confidence of the class predicted by the network. The minimum x and y values are the coordinates of the top left bounding box corner initialized and labelled by the system - these consist of numerical values between 0 and 1. The maximum x and y values are the coordinates of the bottom right bounding box corner initialized and labelled by the system - these consist of numerical values between 0 and 1.

2.2.5 Current State of the Art

In order to determine the need for this project, research on the current state of UAVs designed for firefighting and mapping applications was conducted. The relevant UAVs found are inclusive of the Skydio 2, DJI Matrice 210, Inspire 1 v2.0, and Sensefly's eBee [11][12][13]. After researching these UAVs, it was realized that none of these meet all the specifications listed in this project's objectives. All of the UAVs previously mentioned are lacking some of the following fundamental features, such as a robust vision system, SLAM utilization, heat resistance, and autonomous operation. This made it apparent that a UAV capable of meeting these requirements needs to be engineered. To ensure this process is done correctly, a list of desired customer requirements and technical specifications was created to outline the UAVs functionality.

2.3 Problem Definition Statement

As stated in the early section of this report, the proposed solution is to design and develop a UAV that will assist personnel during emergency and hazardous situations. The UAV will be capable of autonomously exploring unknown areas while simultaneously creating a 3D map that can be used for navigating through the environment. While the UAV is exploring, it must also be searching for any victims and relay their positions back to emergency personnel. To achieve the listed tasks specific criteria needed to be established, to clearly outline the customer requirements.

- The UAV must be able to create a map of its environment.
- The UAV must be able to detect a person.
- The UAV must be able to withstand high temperatures.
- The UAV must be able to enter buildings.
- The UAV must be able to explore quickly.
- The UAV must be able to explore autonomously.
- The UAV must be able to fly for an extended period of time.
- The UAV must be able to see through smoke.

2.4 Criteria for Success

Using the Quality Function Deployment method, a House of Quality matrix was created to distinguish relationships between the customer requirements and the engineering specifications (Figure 3). After comparisons and relationships were made, it was observed that the customer requirements that had the strongest ties to engineering specifications were: the high temperature resistance, ability for the UAV to be swift and agile, for it to include a simple interface when users access the SLAM generated map, and for the UAV itself to be cost effective. After analyzing the engineering requirements, it is observed that the ones with the most strong relations to the customer requirements were the physical attributes of the UAV, including build material, weight, and size, the communication signal type that will be used by the UAV, the processing power of the UAV and the basestation, and the sensors that will be implemented to the UAV.

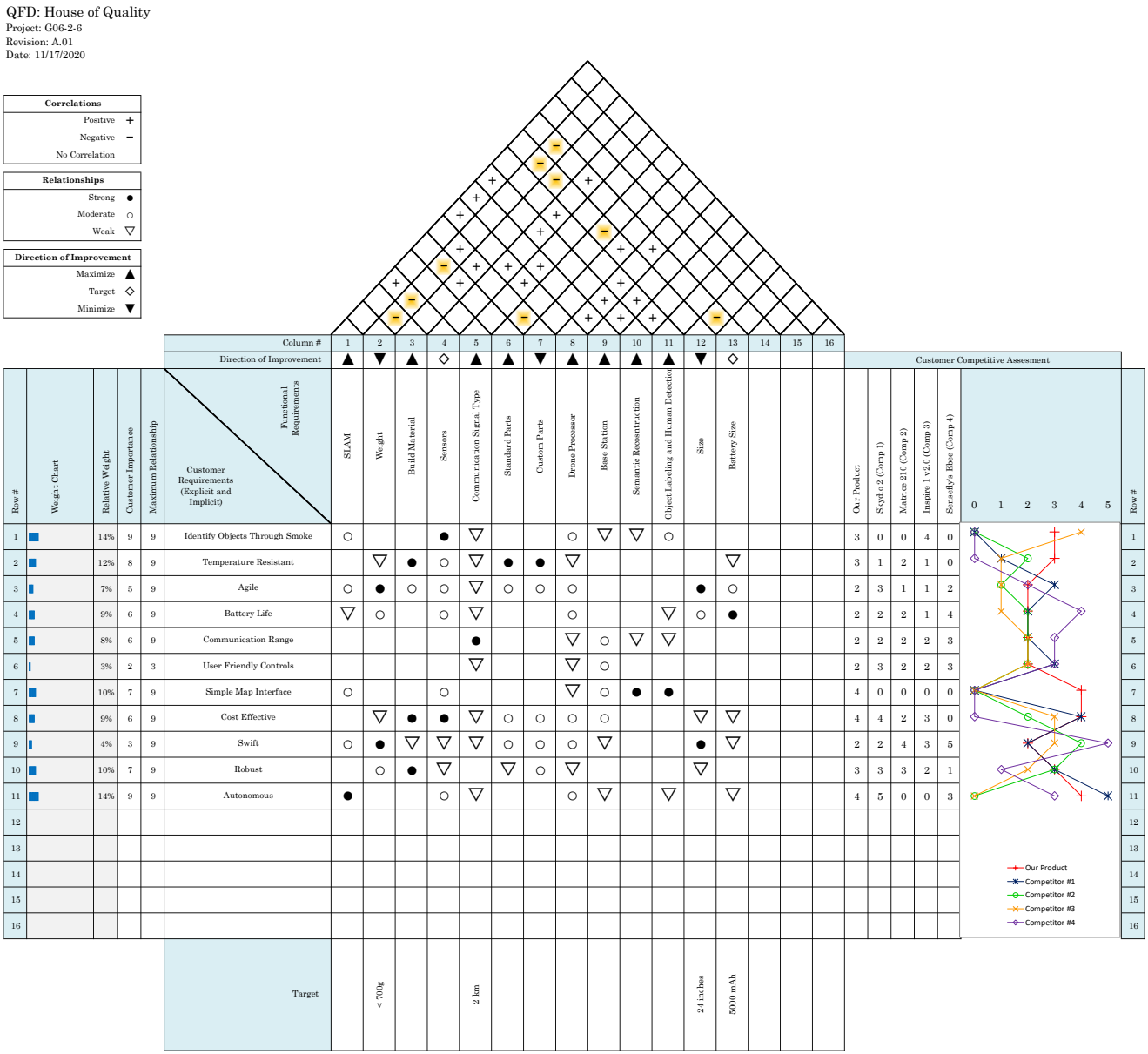


Figure 3: House of Qualities for UAV Prototype

2.5 Technical Specifications

After the implementation of the Quality Function Deployment method utilizing a House of Quality matrix, quantitative and qualitative goals can be set for many of the determined customer requirements. These properties can be further broken down into software and hardware technical specifications.

Hardware Specifications

- The maximum width must be less than 61.5 cm to be able to fit through most doorways and some windows.
- It must also be able to reach a max speed of 40 km/h.
- The UAV should have a connection range of up to 2 km.
- Flight time of up to 25 min to explore large complex layouts within buildings.
- Must be able to withstand temperatures up to 1000 °C for the UAV to safely operate in hot environments.
- The SLAM sensor must be able to operate in dark and smoke filled environments.
- The cost of the UAV will be aimed for approximately $1000.

Software Specifications

- SLAM software must be compatible with the selected hardware, allowing for mapping in common firefighting environments.
- A human detection algorithm that can identify the location of victims, mark their position on the map, and work in smokey or dark environments.
- A lightweight pathing algorithm to provide fast autonomous pathing and obstacle avoidance

3 Design Process

3.1 Hardware

3.1.1 Sensor Selection

After researching all feasible sensors for SLAM applications, further research into each one was conducted to assess all the potential benefits and issues that may arise when equipped on a UAV. The evaluation was conducted based on each sensor's ability to meet the customer and functional requirements. This research can be referred to in Table 1.

Concept	Specifications Met	Potential Issues	Other Notes
Lidar	Can see through light smoke, SLAM	Rotation can throw off center of gravity, limited FOV, inhibited by heavy smoke, can't recognize people	Smoke capability is expensive, heavy, large
Millimeter Wave	Unimpaired by heavy smoke, SLAM	Moderate FOV, not generally used for drones, can't recognize people	Highly directional, able to penetrate through drywall and other materials
Camera Vision	Big FOV, 3D SLAM, Lightweight, human detection	Smoke impaired, poor in darkness	Certain libraries/ algorithms require stereo camera configuration
Ultrasonic	Performs in smoke & darkness, SLAM, lightweight	Very Short Range, can't recognize people	Cost effective, can be used for obstacle detection
IR + Camera Vision	See through smoke & other conditions, robust, 3D SLAM, lightweight , Big FOV, human detection	May be difficult to combine camera feeds for SLAM	Can be used to detect people based on body heat

Table 1: Sensor Research Summary

Based on Table 1, the two best options that fit the project needs are millimeter wave and IR sensors paired with camera vision. Initially, millimeter wave sensors appeared to be the ideal option. However, due to the technology being relatively new, there are currently only 2D open source SLAM applications available for this sensor. Additionally, it has a limited field of view (FOV), which is sub-optimal for traversing in an indoor environment. This could result in incomplete mapping information or unexpected crashes. After determining that these limitations were not ideal, further exploration was conducted into the most widely utilized sensor for UAVs: cameras. Cameras can provide data

from a much larger FOV, as well as being compact and lightweight. Additionally, there are many open source SLAM resources for 3D mapping that utilize cameras as the main visual input. However, in a smoke-filled or dark environment, a camera's vision becomes ineffective. It was decided that these issues could be overcome by using infrared cameras. This will also allow the UAV to collect data based on temperature, which could be used to create a heat map that can be overlaid with the 3D map. It will also allow for vision through smoke in any lighting conditions. This sensor configuration adds the necessary robustness to perform SLAM under the harsh conditions encountered in firefighting.

During initial sensor testing, the plan was to utilize a sensor configuration that featured four cameras paired with IR sensors, ensuring full vision in all directions. However, a variety of issues were encountered when pursuing this design. First, there were concerns that having four camera streams to process would require too much processing power, and result in a map that was generated too slowly. Moreover, when performing distance calculations based on the location of each camera frame, it would be difficult to determine their pose due to the cameras not having an IMU. This could cause inaccuracy in the SLAM map, and add unnecessary calculations to an already intensive process. Finally, this configuration would require a combination of features from multiple SLAM programs because multi-camera SLAM that works with infrared cameras is not readily available. For these reasons, an alternative camera setup was explored, utilizing a single camera.

During sensor research, an infrared stereo camera was discovered, the Intel Realsense D435i. This camera retains the robust vision capabilities of the previous cameras, while eliminating some major concerns. The camera has a built-in IMU, allowing for easy tracking of its pose at any given moment. Additionally, it comes with its own software, allowing for easy testing and ROS integration. Furthermore. utilizing one camera instead of four eliminates the need to add multi-camera capabilities to the SLAM node. This will reduce the FOV of the drone, but will reduce computational load, allowing for real time mapping, a necessary feature for this application. One potential issue with a single camera design is that a front mounted camera will cause a weight imbalance on the drone. Fortunately, this is easily solved by relocating some components to offset the imbalance of weight.

3.1.2 Central Processing Unit

In terms of the central processing unit (CPU) of the UAV, the primary concerns were computational power and size. The CPU would have to be powerful enough to process the data provided from the camera and transmit via Wifi to a separate base station. Since Wifi communication was necessary and a custom printed circuit board (PCB) was not being considered, a CPU that paired with a wifi compatible motherboard became a requirement. The three primary options that stood out were the RaspberryPi 4, RaspberryPi Zero, and Jetson Nano.

The Jetson Nano was the first to be prototyped with the UAV. It was the most powerful of the proposed computers but it was also the largest and heaviest. Moreover, it was selected due to the previous prototype design that required 4 cameras. The Jetson Nano was the only board that supported 4 camera peripherals. Since the team decided to move away from this design in favour of a single stereo camera, the computational power and peripheral ports provided from the Jetson Nano were deemed unnecessary. The weight of the board also proved to be an unfavorable trade off since it resulted in the decrease of the UAVs battery life, speed, and mobility.

The RaspberryPi Zero was briefly considered due to its small form factor. However, due to its lack of peripheral support and the lower computational power compared to the other options, the RaspberryPi Zero was also disregarded as the main processor.

The board that was deemed best for this project was determined to be the RaspberryPi 4. It offered the correct amount of ports for external peripherals while being computationally powerful enough to do the required processing. In addition, the footprint and weight of the RaspberryPi 4 was reasonable. It was smaller and lighter when compared to the Jetson Nano thus it did not significantly reduce battery life or disrupt the balance of the drone.

3.1.3 Battery Configuration

The power supply unit was an important part of the design process for the UAV. The original design had two power supply units. One to power the flight controller and motors, and a second one to power the central processing unit and additional peripherals. The reason for this was concerns regarding the flight time of the UAV and voltage. The Parrot AR Drone 2.0, the base design of this UAV project, utilized a single 1500 mAh battery. During early development of this project, a specialized power supply unit designed for the Jetson Nano was used in addition to the primary battery of the drone. This setup proved to be too heavy and resulted in the drone not being able to lift off the floor during takeoff. Therefore, a single battery setup was quickly adopted.

When designing the first power supply system with the Jetson Nano, a voltage regulator was required to step-down the 11.1 V battery to the maximum allowable input voltage of the Jetson Nano, which was 5V. For the single battery design, a 2200 mAH, 11.1 V, 50C, was chosen for the UAV. This battery allowed for enough current to power all the motors and the Jetson Nano, while being light enough to allow the drone to lift off. However, when the decision to switch to a Raspberry Pi 4 from a Jetson Nano was made, the same battery was chosen as the power supply. The only change made was the removal of the voltage regulator. This is because the Raspberry Pi receives power via USB from the drone flight controller.

3.2 Software

3.2.1 Simulation Environment

For testing features like autonomous pathing, a compatible simulation was needed in order to verify functionality. Different simulation software were evaluated and tested to see if they could be of use for testing the UAVs programs. Four notable simulation softwares that were examined were Airsim, Texas Aerial Robotics, tum_simulator, and Hector Quadcopter. The following are the summaries of each simulations' evaluation and whether or not it could be used for testing.

AirSim

AirSim (Aerial Informatics and Robotics) is an open-source cross platform simulator utilized to simulate drones or ground vehicles such as cars. It was developed by the Microsoft Corporation built on Epic Games' Unreal Engine 4 for testing algorithms and AI research [14]. The simulation was used to simulate a Parrot AR drone within a custom virtual environment to test the tasks that must be achieved by the UAV. A 3D virtual house can be created for the drone to test the path planning algorithm and let observers measure the success rate using quantitative references such as the number of nodes created and the duration of exploration. AirSim can also be used to test the functionality of SLAM programs by interfacing the simulation with ROS. However, one tradeback with the simulation software is that a large amount of processing power is needed for the simulation to run fluently and with no issues. This restricted the number of tests that can be performed and who could perform them due to the workstations that can run AirSim not being always available.

Texas Aerial Robotics

Texas Aerial Robotics is a UAV simulation program that runs utilizing ROS Melodic and Ubuntu 18.04. It simulates a UAV called Pixhawk in a three dimensional environment with a satellite view and a local view. The simulation is capable of being run in a gazebo simulation environment or can be used to pilot a physical UAV. Two issues were encountered that made this simulation unusable for testing in this project. First, the program was too computationally taxing as the computer would struggle to run the simulation alone. This would be troublesome when launching more programs to test them in simulation, and would be too heavy of a load on the computer. The second issue was that the navigation of the simulated UAV could only be done using preset or manually set waypoints. This unfortunately was not compatible with the autonomous pathing software the UAV prototype will be using [15].

Tum Simulator

The next UAV simulation, called tum_simulator, was also a Gazebo based simulation which would use ROS and Ubuntu. This simulation had the same UAV that was being used as the physical prototype for the project. This would be convenient for viewing and flight controller conversion purposes. Unfortunately the simulation was unable to be launched properly due incompatibility with software on the computer (Figure 4) [16].

Figure 4: tum_simulator

Hector Quadcopter

Hector Quadcopter was the third simulator that utilized ROS and Ubuntu as a base. Contrary to the previous simulations, Hector Quadcopter does not use a specific type of drone. This software simulates an environment in Gazebo with a UAV that adheres to real world physics. This simulation is lightweight enough for the hardware available, allowing for the use of other software that needed to be tested with the simulation. Another benefit was that Hector Quadcopter came with pre-built test environments and provides the capability to create custom ones. For these reasons Hector Quadcopter was selected as the main simulation to be used for testing the autonomous pathing software (Figure 5) [17].

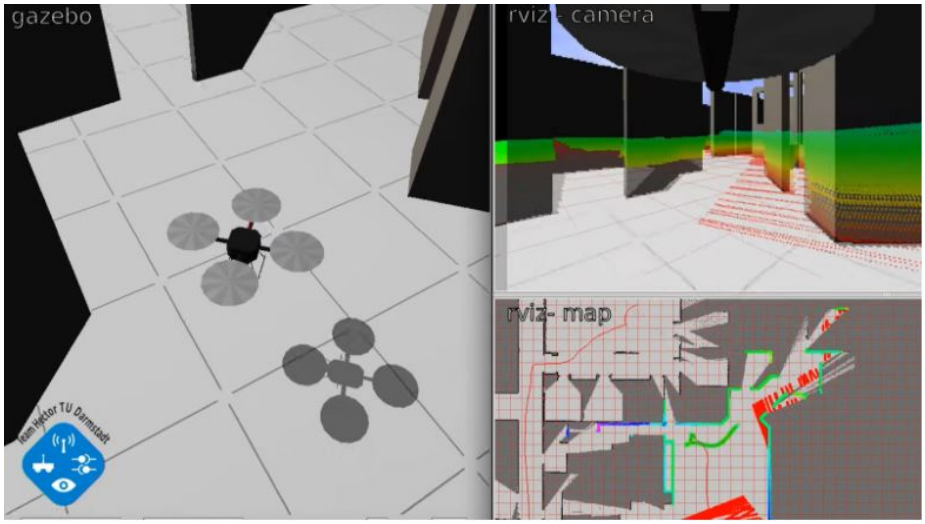


Figure 5: Hector Quadcopter

3.2.2 SLAM Software

During early stages of development, a variety of SLAM software were being explored and tested. For the first design of the drone, a four camera infrared setup was desired. For this reason, SLAM software with multicam capabilities, infrared compatibility, and the ability to perform 3D dense reconstruction was necessary. It was determined that there was no individual software available which could meet all these specifications, so the plan was to merge the functionality of multiple programs. As previously mentioned, various issues came to light, such as issues with the sensor configuration, processing power, weight, etc. After a hardware overhaul was conducted, the software plan also needed to be adjusted. Changing the camera configuration to use the D435i stereo camera instead of four infrared cameras greatly simplified the software plan. Considering that Kimera already has the ability to process infrared feeds for stereo cameras, Multicol and ThermalVis were no longer needed. Thus Kimera was selected as the primary SLAM program to integrate and test.

After setting up the camera and running Kimera, there were some issues, as Kimera did not work immediately. While debugging was conducted for Kimera, another SLAM software that utilizes the D435i camera was explored as a backup, RTAB Map. Fortunately, the issues with Kimera were resolved through extensive testing, and RTAB Map was no longer required. In depth discussion with regards to software testing and debugging will be discussed in the laboratory test section.

3.2.3 Autonomous Path Planning

3D Mesh with A* Pathing Algorithm

The first concept of path planning was to integrate the three dimensional mesh that is used for 3D SLAM construction. Multiple A* algorithms were tested to find the most optimal algorithm for this application. From these tests, the A* Manhattan heuristic algorithm was chosen due to its speed and accuracy. The next step of this concept was to figure out how to extract the 3D mesh while they are being constructed by Kimera. This is where three main issues were encountered with the concept. The first issue was that the three dimensional mesh being built by Kimera could not be properly extracted for path planning to use. Secondly, the extracted mesh would not have unknown positions marked in it. This means that there would not be a way to mark unexplored areas as goals for the path planning algorithm. The third problem was the compatibility between the algorithm and the Kimera generated mesh. The C++ algorithm would have to go through many changes to get the mesh to be an acceptable input into the algorithm.

ROS Programs

An approach for autonomous path planning using only ROS based programs would be an ideal solution. This is because many ROS programs used for exploration and path planning are widely available and tested. In addition to this, Kimera already runs within the ROS workspace which would increase chances of being able to pass data between them. In order to properly design the overall path planning system, programs needed to be located that could:

- Set unknown areas as goals for path planning.
- Calculate an optimal trajectory to the goal.
- Convert trajectory into velocity commands and send them to the UAV.

The third requirement was found first since there was prior knowledge of a program called "move_base". This program is capable of working with costmaps and ROS path planners to create trajectories and send them to a robot as velocity and acceleration values in the x, y, and z directions. Another program called costmap_2D works with move_base by default and generates a local and global costmap to be used by a ROS path planner. Three different ROS path planners were evaluated to be used with the two previously mentioned programs. The names of these path planners were base_local_planner, dwa_local_planner, and teb_local_planner. All three programs were very similar in how they operated and the parameters that could be adjusted The dwa_local_planner was picked out of the three since it operated more consistently and seemed to be better for the current application.

Finally for the exploration requirement, two different frontier based programs were tested, "Frontier_Exploration" and "Explore-Lite". After evaluating the code and the adjustability of the parameters for frontier placement, explore-lite was chosen. To work in tandem with the frontier exploration algorithm, a two dimensional SLAM program called "slam_gmapping" was chosen to provide obstacle avoidance.

With the current setup, the UAV would be able to navigate an unknown 3D area while maintaining a specific altitude. To make the autonomous path planning, a program was needed to give the UAV altitude control. To accomplish this, a github repository was referenced which provided a program called force_hover. This program would give the ability to maintain an altitude and adjust it accordingly if an object goes under the UAV.

3.3 Prototype Concepts

This section will discuss the different iterations of the drone assembly, issues encountered, and how they were solved.

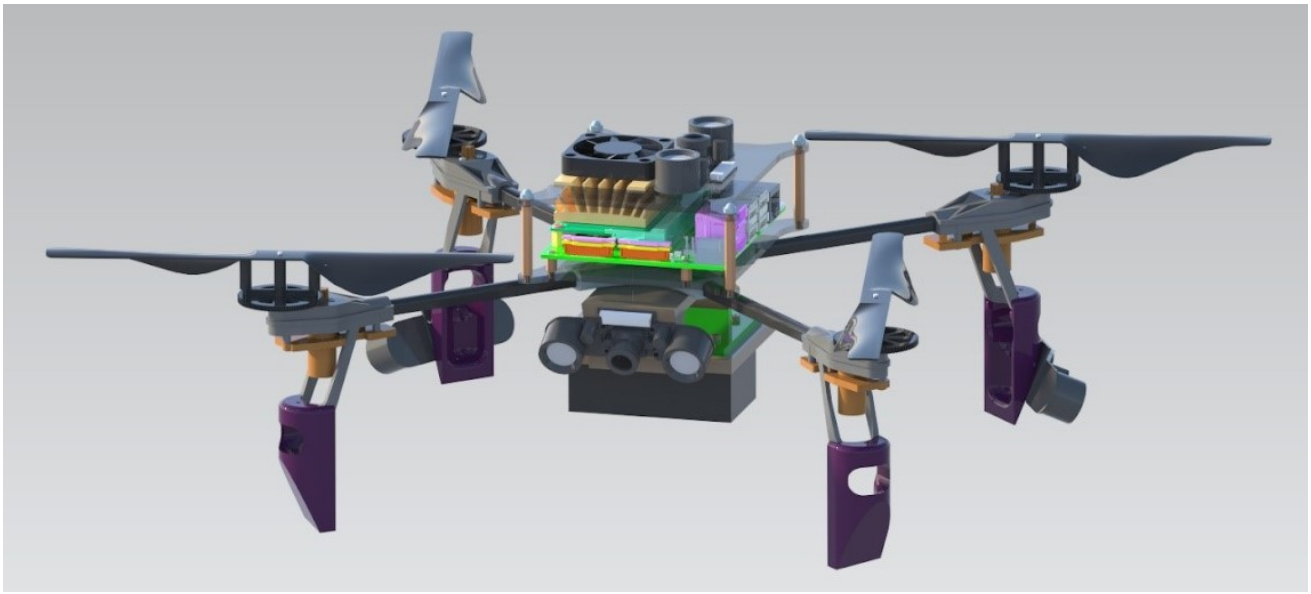


Figure 6: First Iteration of Drone Design: Four IR Cameras and Jetson Nano

As seen in Figure 6, the first drone design incorporates four IR cameras and a Jetson Nano as the CPU. After further development was made on this design iteration, a number of issues arose. The weight of the extra parts caused the flight time to be very short, and inhibited the top speed and maneuverability of the drone. As mentioned previously, this camera configuration had various issues, such as lack of usable IMU data, and having too much data to process. Since the cameras did not have their own IMU, the one on board the Jetson would have to be used. However, it was realized that the SLAM software "Multicol" performed distance calculations based on the pose of each camera. So an IMU for each camera is a requirement. Another issue with this configuration was that in order to connect all the cameras to the Jetson, an Arducam Multiplexer was necessary to act as additional MIPI CSI-2 ports, as the Jetson only has two. Unfortunately, the Arducam Multiplexer functions by rapidly switching between its ports, and did not allow true simultaneous streaming. This would cause the timestamps of each camera to be slightly different. This made the images unusable by Kimera, because it requires multiple images with the same timestamp in order to generate keyframes, which

are then converted into a point cloud. These issues were seen as too large to ignore, so a redesign began with the goals of reducing overall weight, using only one camera, and finding an infrared camera that has a built in IMU.

Based on the component analysis previously conducted, the components selected to fulfill the new design specifications were the Intel Realsense D435i Stereo camera, and the Raspberry Pi 4. This reduced the weight of the drone from 776g to 706g, allowing for longer flight times, and more stable flight. Moreover, the SLAM modules were able to run notably faster now that significantly less data is being sent and processed. Another major advantage to this design is that it works with Kimera, and does not require integration of any other SLAM program. The design removes a large amount of complexity from the hardware and software design aspects. The redesigned drone can be seen in Figure 7 seen below.

Figure 7: Final Drone Design: D435i Stereo Infrared Camera and Raspberry Pi 4

4 Detailed Design Documentation

4.1 Assumptions Made

Several assumptions were made throughout the design process of the UAV based on who will be using the UAV and how it will be used. The main assumption that was made was based on what kind of environments the drone will be flying in and exploring. The most ideal environment the UAV is intended to fly in is a building with multiple points of entry, such as several open doors and windows. Also, it is ideal that all the rooms and locations in the building are accessible, such as no rooms with closed doors or with large debris blocking a corridor.

Another assumption made was that only experienced users would be controlling the UAV. They have gone through the training to practice flying drones at high speeds in close quarters environments. Even though a majority of the tasks that the drone must accomplish can be done autonomously, a user must stand by with the controls in the event that there is an emergency where the drone is unable to perform adequately and user intervention is needed. To fly the UAV safely out of the hazardous environment, an experienced user must take control of the UAV.

4.2 Function of the System: Hardware

The hardware of the UAV can be primarily summed up into five systems: power supply, flight controller, central processing unit, motor controllers, and peripheral sensors. Each of these systems is vital to the performance of the drone. An additional base station is also used to communicate with the main processing unit. It receives data from the UAV while simultaneously sending flight commands. Figure 8 depicts a block diagram, showing how each of these systems communicate and operate together.

When planning the modifications for the hardware of the drone, it was decided that it would be best to only include one power supply. Previously, issues were discovered where the drone would have difficulty taking off due to its weight. To combat this, the team decided on moving forward with a

design concept that utilized only one power supply to provide voltage to the drone, its integrated components, and the additional central processing unit.

The flight controller acts as an intermediate between the motors and the central processing unit. It also handles the power management between the peripherals. The flight controller communicates with the motor controllers to handle the balance, speed, and altitude of the UAV. Previously, a serial connection was set up to communicate with the original central processing unit, the Jetson Nano, via a UART bridge. However, since the computer was changed to a Raspberry Pi 4, communication was initialized via USB.

The Raspberry Pi 4 handles the main computational processes the drone has to perform, including processing sensor data from the primary peripheral, the Intel Realsense D435i. It communicates with the base station via a Wifi network emitted from the flight controller. It receives sensory data from the drone in order to process it using its SLAM, path finding, and exploration algorithms. It then transmits the movement commands to the Raspberry Pi 4. The Raspberry Pi 4 then processes the commands and relays them to the flight controller to control the drone.

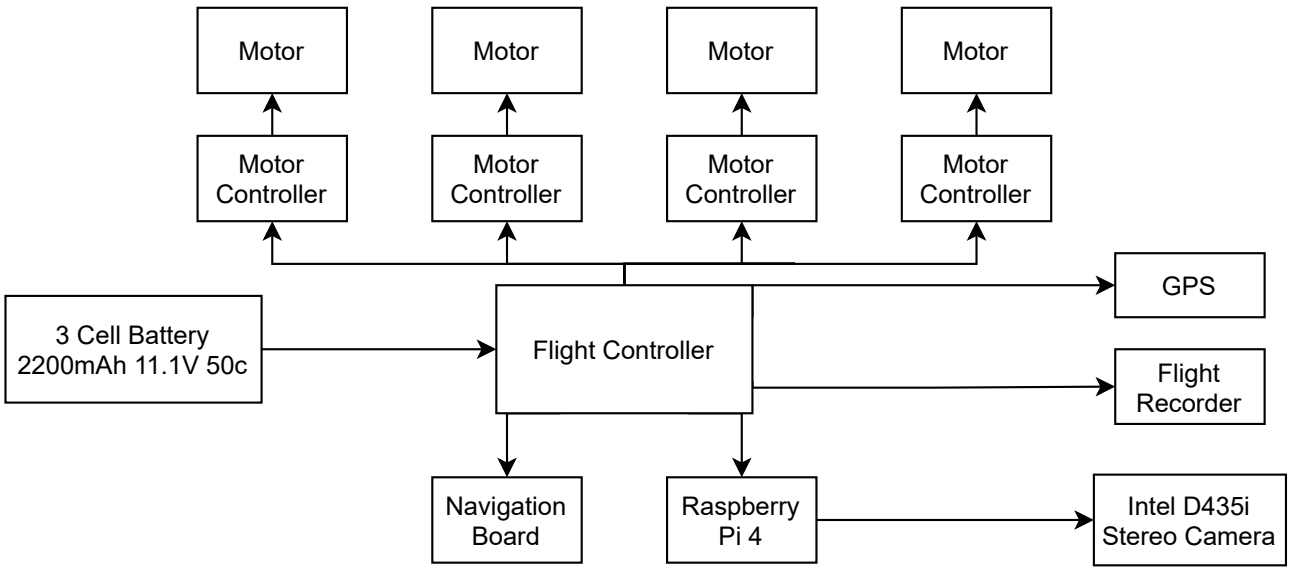


Figure 8: Electrical Block Diagram

4.3 Function of the System: Software

The software system for the drone integrates multiple complex programs in order to achieve its goal. The software system of the drone consists of four primary programs: VIO, SLAM, path planning and human marking. A functional decomposition diagram can be seen in Figure 9 for reference. This diagram shows a high-level depiction of the software system and how all the features interact with one another.

The VIO program integrates sensor data from the camera and IMU to create point clouds out of the environment. This is used to understand what objects are around the UAV. The data provided by this program is then used in the SLAM program and the path planning program.

The SLAM program receives point cloud data from the VIO module and turns the data into a map for the drone to navigate and for people to use as a visualization tool. The SLAM program can be split up into 2D SLAM and 3D SLAM. The 2D SLAM is used for autonomous navigation while the 3D SLAM is primarily used for visualization purposes for the user. VIO produces a 3D point cloud, which is useful for 3D SLAM, however, in order to use it with 2D SLAM it first gets converted into a 2D laser scan. A separate program converts the 3D point cloud into a 2D laser scan by placing all the points within a certain Z range on a singular plane. The laser scan can then be used to create a 2D map on the aforementioned plane.

The same 2D laser scan and 2D SLAM map are used by the path planning program for navigation. It uses the laser scan to create a 2D cost map in order to properly path around objects and obstacles. The goal of the path is determined via an embedded frontier based exploration algorithm. Frontiers are essentially open space that is represented as unknown to the drone. When there are no longer any frontiers to be explored, the drone then returns to its initialization position.

The last major program is the human marking program. This program used the 3D map generated from the SLAM module and sensor data to place indicators on the visual map. This program can be separated into two primary components: human detection and marker placement. The human detection portion uses the camera to identify people in its field of view using a trained neural network. Once a person has been identified, the location of the drone is referenced via the IMU and the 3D SLAM program and a marker is placed at the person's location.

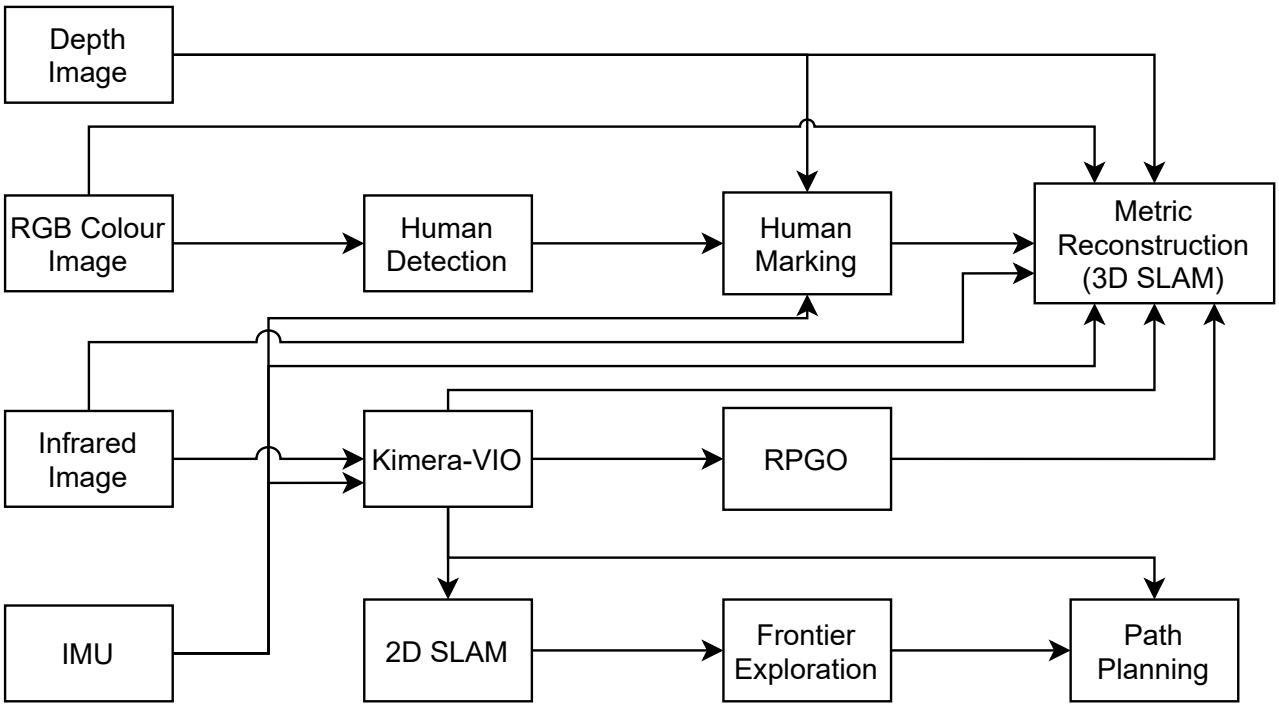


Figure 9: High Level Software Block Diagram

4.3.1 VIO & RPGO

Kimera VIO and Kimera RPGO are both separate modules that are part of Kimera. The VIO module combines camera vision with data from an IMU to determine the pose and velocity of the UAV. With this module, the UAV system would have information about its own movement, but would not know its position with respect to the generated map. The RPGO module alleviates this predicament by tracking the trajectory of the UAV system with respect to the map generated [18]. The RPGO module is also responsible for loop closure, the ability to detect when a location has already been visited, and update the map accordingly. Together, these modules provide a fast method for detecting obstacles in 3D space, while also localizing the drone within the map. Using a tool called Rviz, a 3D visualization tool, it is possible to view the data of the UAV using the respective modules.

4.3.2 3D Global SLAM and Metric Reconstruction

3D Dense Metric Reconstruction is a feature that is part of the third module of Kimera, Semantics. This module utilizes a volumetric mapping library called Voxblox to reconstruct a 3D point cloud into a 3D map that closely resembles the real world. This feature allows for an accurate visualization of what the drone has seen, enabling firefighters to better understand the situation while learning the layout of the building. It should be emphasized that this module does not function on its own, a 3D point cloud must be provided. For this project, Kimera VIO was used to generate the 3D point cloud.

4.3.3 2D SLAM

A two dimensional SLAM algorithm is used within the autonomous navigation system of the UAV. The purpose of the two dimensional SLAM is to create a 2D map for the frontier exploration program, explore-lite, to use. A SLAM program called “slam_gmapping” is used to create this 2D SLAM map. The input of the SLAM map is a point cloud published by Kimera-VIO. To utilize the point cloud with ““slam_gmapping” a conversion needs to be done. This conversion is taking the point cloud and selecting points based on a z axis threshold. If a point of the pointcloud falls in between this threshold it will be accepted into a two dimensional laserscan. This laserscan is based on the 0 point of the z-axis and is then used by “slam_gmapping”.

The slam_gmapping node will use this point cloud to mark free space and obstacles in an occupancy_grid called “map”. The map can be updated if obstacles move over time by using marking and clearing functions. The pose of the UAV on the 2D SLAM map is done by the path planning software. The following Figure 10 shows an example of the two dimensional SLAM map outputted by slam_gmapping.

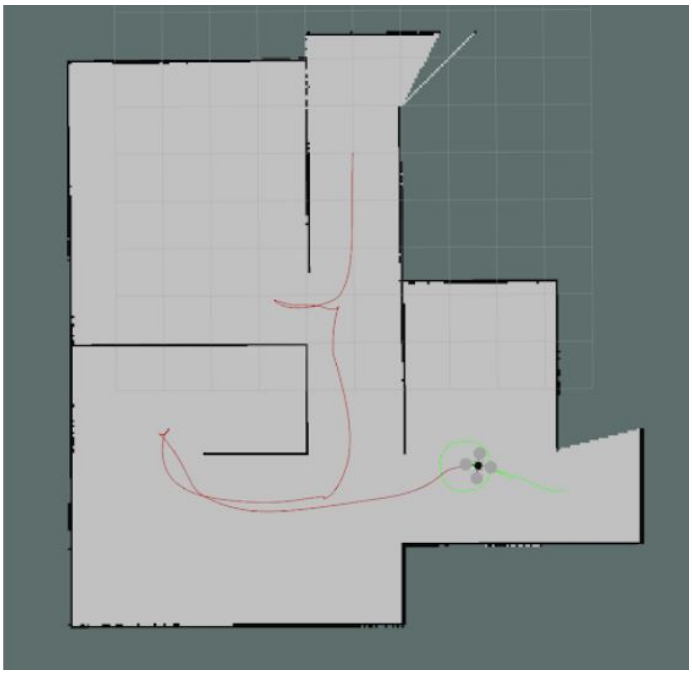


Figure 10: 2D SLAM Map slam_gmapping

4.3.4 Frontier Based Exploration

To set goals for the path planning to use, a frontier based exploration algorithm is needed. A program called “explore-lite” was chosen since it is compatible with the path planning and 2D mapping software currently in use. Using this map, explore-lite will search for frontiers based on adjustable parameters. All frontiers found will be published in an array which can be displayed using the RVIZ program as seen in Figure 11. The frontiers are the highlighted dark blue lines and their centroids are the lime green spheres. The size of the centroid sphere is larger if the frontier is more optimal for the UAV to head towards at its current position.

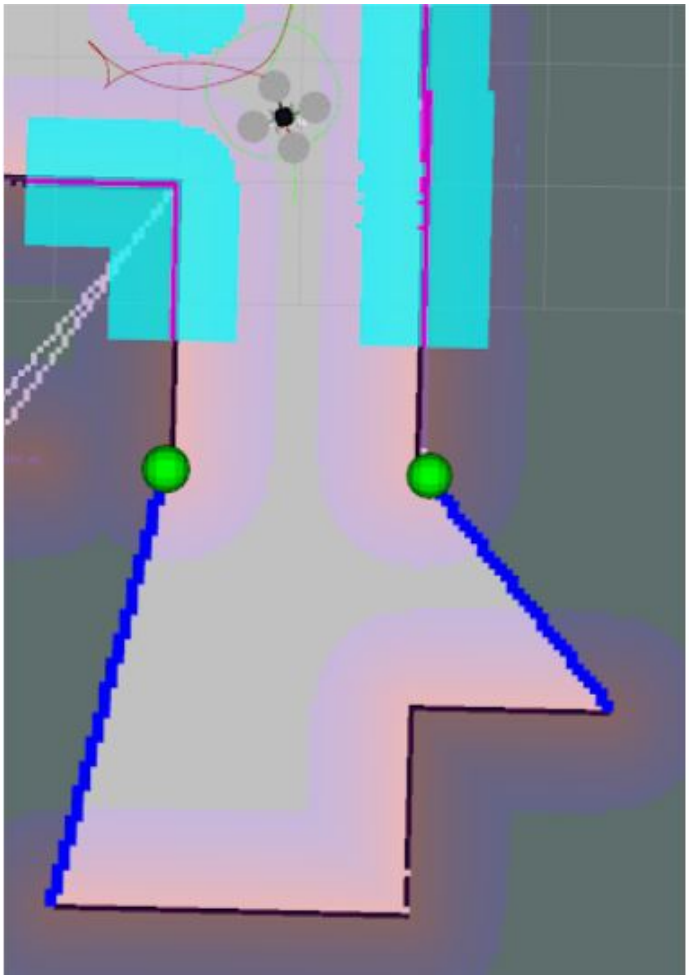


Figure 11: RVIZ View of Explore-Lite Frontier Exploration

To decide on which frontier to set as the goal for path planning, parameters are used. These parameters are set in a .yaml file and pertain to the frontier size, orientation and distance. Since all possible frontiers need to be evaluated the size of the frontiers is redundant and set to 0. The orientation and distance of the frontier is given equal weighting since both are ideal for making the exploration faster. This is because going to a frontier the UAV is already facing is ideal since the UAV has a camera forward bias. Also going to the frontiers closer first will cause for a faster evaluation of all possible frontiers in the area.

Once a frontier is set as the goal it will be sent to the path planning software. The explore-lite program will continuously update frontiers even while the UAV is moving to its current goal. Sometimes a frontier the UAV is not currently travelling towards may become a more ideal goal. In this case explore-lite will change the goal for the path planning software to the new frontier. When no more frontiers can be determined, the explore-lite program will initiate the "return home" function.

4.3.5 Path Planning

There are three programs used within the path planning software to create paths and move the UAV along them. These programs work in parallel to plot paths and send the velocity commands to the UAV. The first program used is "move_base" which moves the UAV to a set goal by utilizing the odometry and transform frames of the UAV. This program receives goals from outside programs and passes them along to the other two pathing programs. The goals passed along may be received from "explore-lite" or "return home" depending on the state of the UAV. After receiving a goal, "move_base" will move the UAV by sending data to costmap_2D and DWAPlanner to retrieve trajectories in return. Once move_base receives the trajectories it will then publish the linear and angular velocities and accelerations. The velocities are sent to the flight controller of the UAV to move it towards the set goal. Note that prior to being retrieved by the UAV the force_hover program will alter the linear z velocity values for altitude control. This continues until no more goals are being sent to move_base or the UAV is shut down.

Costmap_2D is used to create two costmaps, a local and global. These costmaps store values in an occupancy grid on a scale of 0 to 254. The higher the value stored for a point in the occupancy grid the higher chance there will be a collisional travelling through that point. The inflation chart for the costmaps is included in Figure 12 [9].

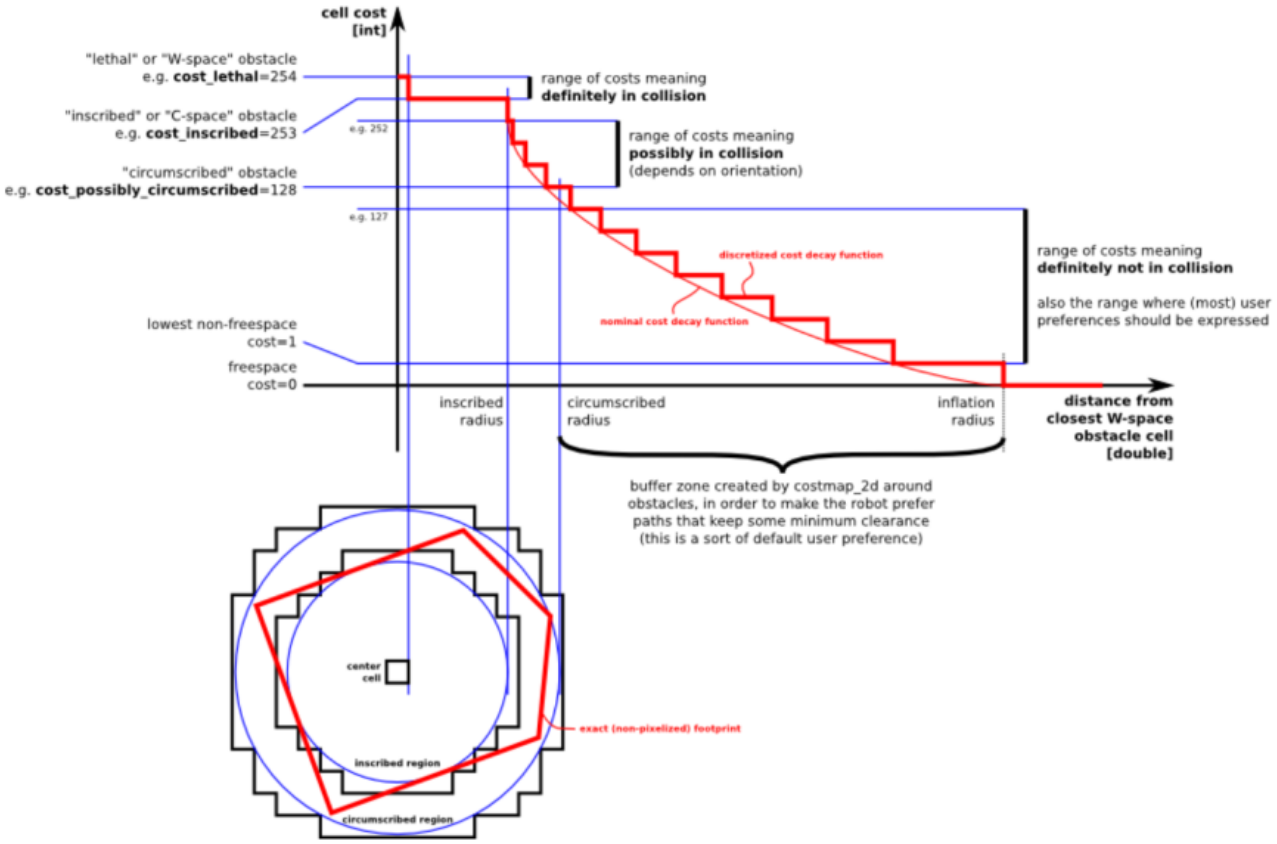


Figure 12: Costmap Chart for Cost Values

The costmaps use a set of parameters that are set in three .yaml files. There are individual files for specific parameters for either the local or global map and one file for common parameters. The common parameters file is for parameters that are the same for both the local and global costmaps, for instance the sensor inputs. Within these .yaml files the transform frames, sensor topics, and costmap sizes are assigned. The inflation and obstacle layers of the costmaps are modified to use the point cloud from Kimera. The footprint and inflation is set according to the size of the UAV to assign appropriate costs for both the local and global costmaps.

The local costmap is a non-static, smaller map that uses the rolling window feature to follow the UAV with it always at its center. The reason for the rolling window is to always update the local surroundings around the UAV. This costmap is four by four meters and updates at a higher frequency than the global costmap. The local costmap is used mainly for obstacle avoidance since it is used by the local trajectory planner within DWA planner. The following Figure 13 shows the local costmap around the UAV.

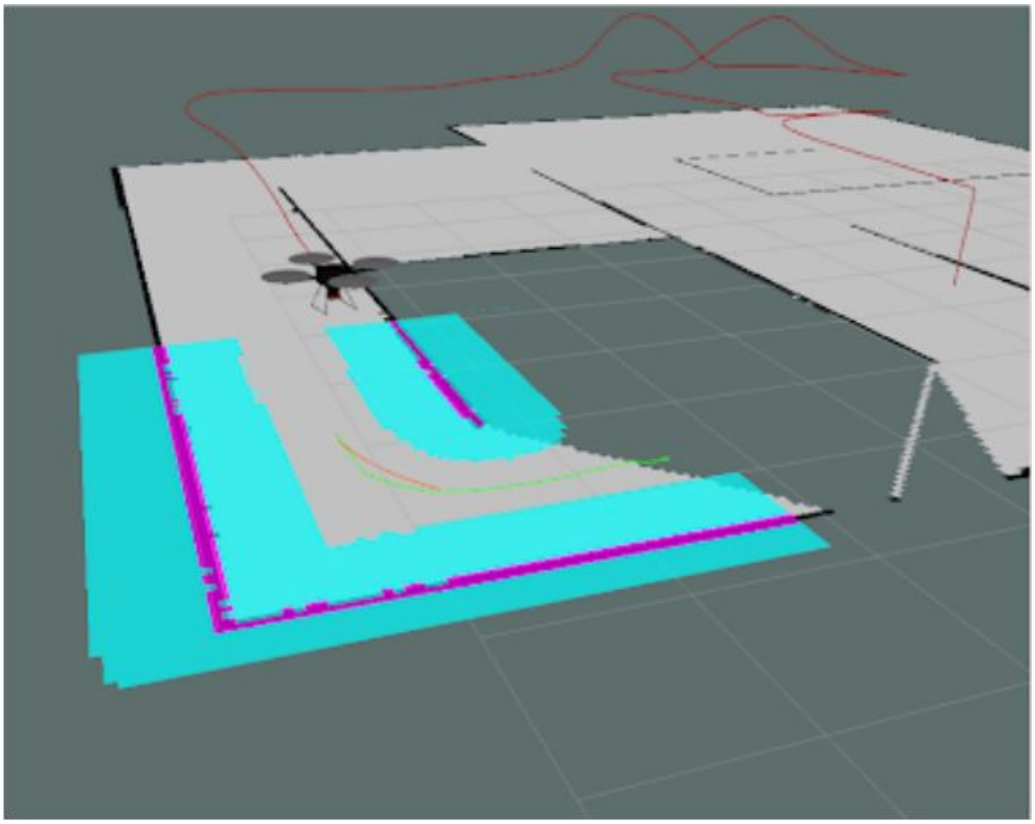


Figure 13: Local Costmap

The global costmap is the larger scale map which keeps track of a weighted map of the entire explored area (Figure 14). This map is static and stays stationary on the map transform frame called "map" at the origin. The global map uses the 2D SLAM map from slam_gmapping for data along with the inputted point cloud. This results in a global map that is almost identical to the two dimensional map from "slam_gmapping". The difference is the added inflation radius and costs to be used by the DWA global trajectory planner.

Figure 14: Global Costmap

The DWA planner uses the two published costmaps from costmap_2D to create two trajectories to the set goal (Figure 15). When the DWA planner is initialized it will load preset parameters from the a .yaml file. These parameters set the max velocities and accelerations of the UAV. Within this file the planner is set to have a holonomic feature off, meaning the planner can make the UAV strafe if need be. The path planning of the UAV has a camera forward bias meaning that it will make the UAV almost always travel while facing forward. This is because the stereo camera needs to be able to see where the UAV is going to map new areas and to perform obstacle avoidance.

The global trajectory utilizes the global costmap to create a long trajectory with waypoints to the currently set goal. This trajectory does not control the UAV specifically but gives the optimized path for the local trajectory to try and follow. The global trajectory will tend to get as close as possible to obstacles without infringing too much cost. Since the costmaps are calibrated to the size of the UAV the global will not put the UAV in any scenarios where it could have a collision.

The local trajectory uses the local costmap from the previously mentioned costmap_2D to create a smaller localized path. The local path will try and follow the global trajectory depending on the bias parameters set in the startup. In this case some freedom is given to the local planner in case it is more convenient for the UAV to make a wider turn for example. The local path is published at a quicker rate and is used for obstacle avoidance since it will react to changes in scenery when the global plan may not. The local trajectory is used to calculate the velocities and accelerations for the UAV.

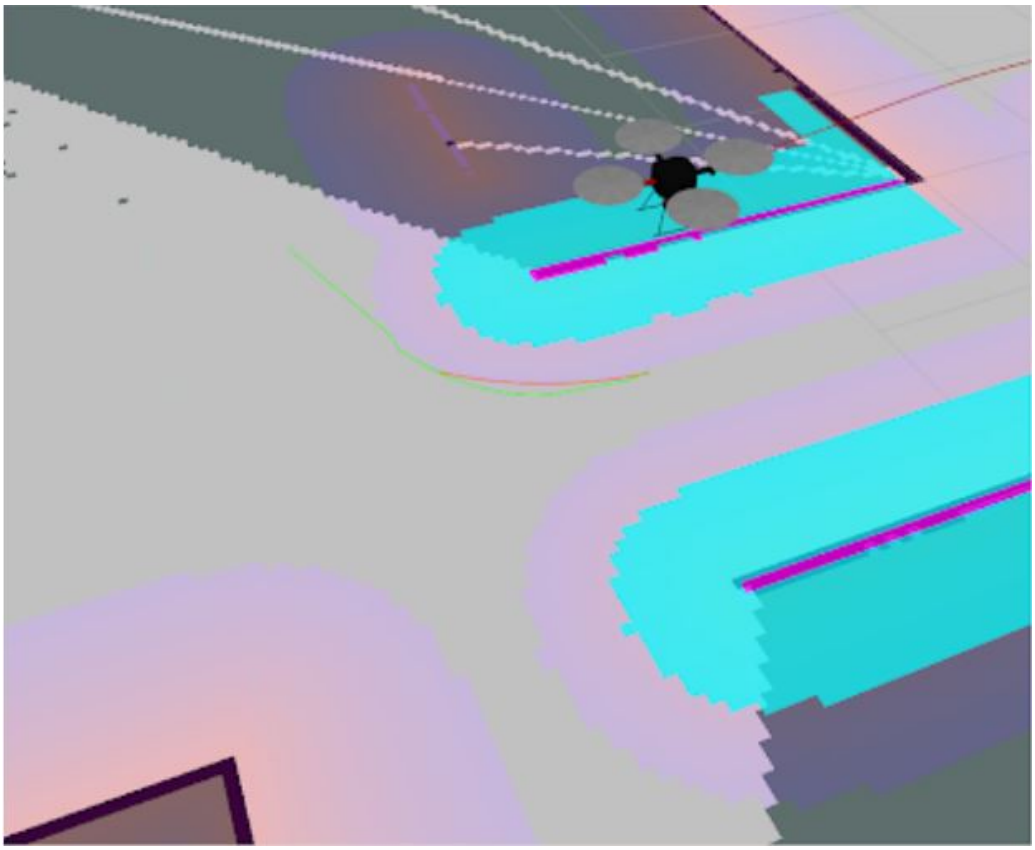


Figure 15: DWA Planner Trajectories

To control the altitude of the UAV during exploration "force_hover" is used. This program was accessed on the following github and was written by "Tim Kuhz" [19]. Upon start up of the autonomous path planning this program will begin and raise the UAV's altitude to one meter. To determine the height of the UAV from the ground, the program uses the ultrasonic sensor. The program is capable of altering the altitude when the ultrasonic sensor detects obstacles under the UAV. For example when the UAV flies over a table and it is too low, it will gain altitude. This program will function until the UAV returns home and the altitude needs to return to a value of zero.

4.3.6 Return Home Function

The return_home program is a Python script that was created to save the UAV's starting pose as its home location. When the explore-lite node cannot find any new frontiers it will then initiate this program. Once initiated it will publish the saved home coordinates and orientation to the move_base node as the current goal for path planning. This in turn will make the UAV begin to traverse out of the building towards its initial position. An example of the return_home program being used can be seen in Figure 16 below where the UAV is returning to its initial position.

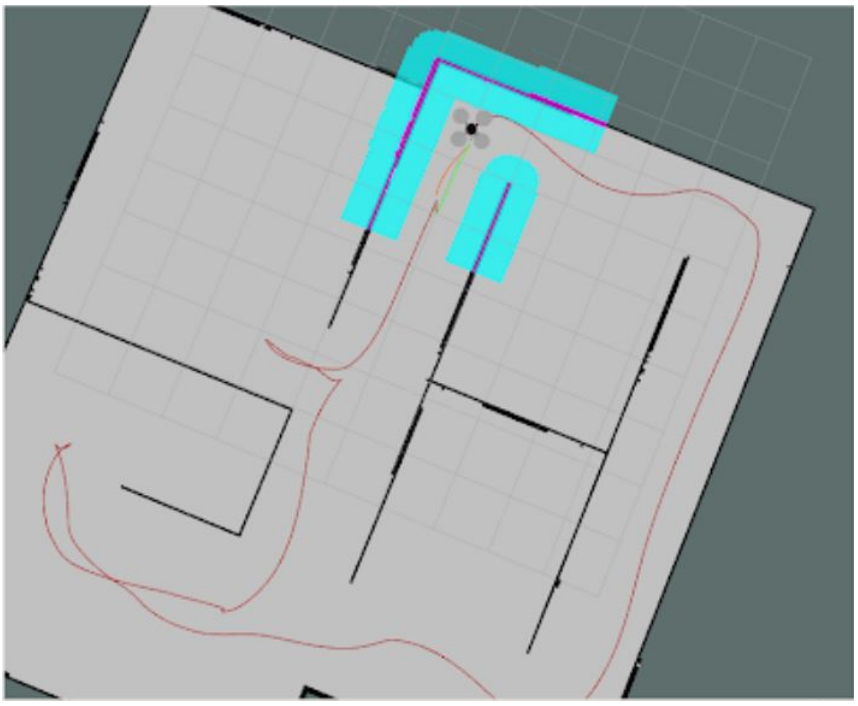


Figure 16: Return Home Function in Simulation

4.3.7 Autonomous Path Planning

The block diagram in Figure 17 shows how all of the programs for autonomous path planning are connected. The point cloud from Kimera VIO is sent to the Path Planning software to be used for the costmaps. It is also converted into a two dimensional laser scan by the "Point Cloud to Laser scan" program. Then the converted laser scan is given to the "slam_gmapping" program to make a 2D SLAM map. Frontier exploration is performed on this 2D map by "explore-lite" to identify frontiers. The found frontiers are then sent as goals to "move_base" in the Path Planning Software. The "DWA Planner" then creates global and local trajectories which are sent back to "move_base". The "move_base" program will publish the velocity commands which are received by the "force_hover" program. The altitude is then changed by "force_hover" by editing the z-axis velocity. The new velocity commands are then sent to the UAV. Once "explore-lite" cannot see any more possible frontiers on the 2D map, it initiates the "return_home" program. This program will set the initial position of the UAV as the goal for "move_base" which will take it home outside the building.

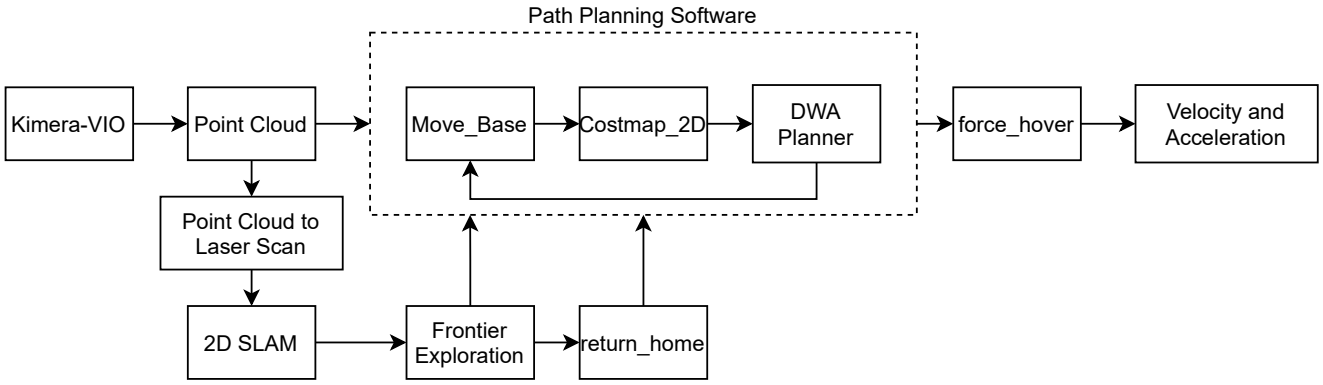


Figure 17: Autonomous Path Planning Block Diagram

4.3.8 Human Detection

In order to properly detect an object or humanoid figure, a caffe model must be utilized. A caffe model is a deep learning framework that contains a neural network of data representing information about how a class is represented on a pixel grid; a class is a labelled classification of items such as a human, door, objects, houses, etc. A typical caffe model utilizes two files, the caffemodel file and the

prototxt file. The caffemodel file contains information pertaining to the pre-trained model consisting of class labels that mobilenet ssd was trained to detect - these labels are then used to generate a set of bounding box colours for each class of object. To initialize the training data, the serialized model is imported from the operating system. Once imported, this trained serialized model is utilized along with the deployment file to analyze the incoming video stream from the depth camera. When an array of pixel data is analyzed and is found to match the initialized model, a blob is created and passed through the network to obtain directions and predictions. Each detected blob is then iterated over and once the confidence threshold for the detection has been reached, a rectangle is created and drawn over the video stream in the location of the analyzed pixel array to alert the operator that a class (human, object, door, animal, etc.) has been detected by the system.

To prevent the system from recognizing faces and bodies where people are not represented, a confidence scoring system was implemented. This system would effectively filter out unnecessary noise and prevent false marking of identified victims. This system works by providing weights to the facial detection and body detection, with a slight preference for body detection. This is because, during the testing, it was determined that the body detection more accurately identified bodies and would less likely identify unrelated items as bodies. In addition, the distance a face or body is away from the camera factors into the confidence scoring. To determine the distance, the size of the area recognized as a face or body is used. The larger the recognized area the closer it is to the camera. The closer it is to the camera, the more accurate the system is at correctly recognizing the face or body. The final measurement is the time of length an object is marked. The longer a face or body is recognized, the more likely the recognition is correct. All these factors are given specific weights and then added together to create the confidence score. The system will then identify an object as a "person" if the confidence score is above 70. The optimized score was determined through rigorous testing and will be explained in a later section.

4.4 Human Marking

In order to properly have an accurate visual representation of an identified person's position, a marking function was employed. This function would utilize the 3D metric reconstructed map, the depth-image processing, and the human detection module to place a 3D mesh marker of a person at the location the UAV found them. When the human detection algorithm determines that a person is within the camera frame, it references the coordinates of the pixels that the system identified as the face. It then takes the coordinates of the pixels and uses a built in function to grab the depth value of the same area of pixels in the depth-image. In order to relate the depth value received from the depth-image to the rest of the 3D metric reconstructed map, the coordinates values from the IMU of the drone are referenced. With the coordinates of the identified person registered, a marker transform message is published to the visualization tool to display the 3D human mesh on the map. Figure 18 illustrates the relationship of this system.

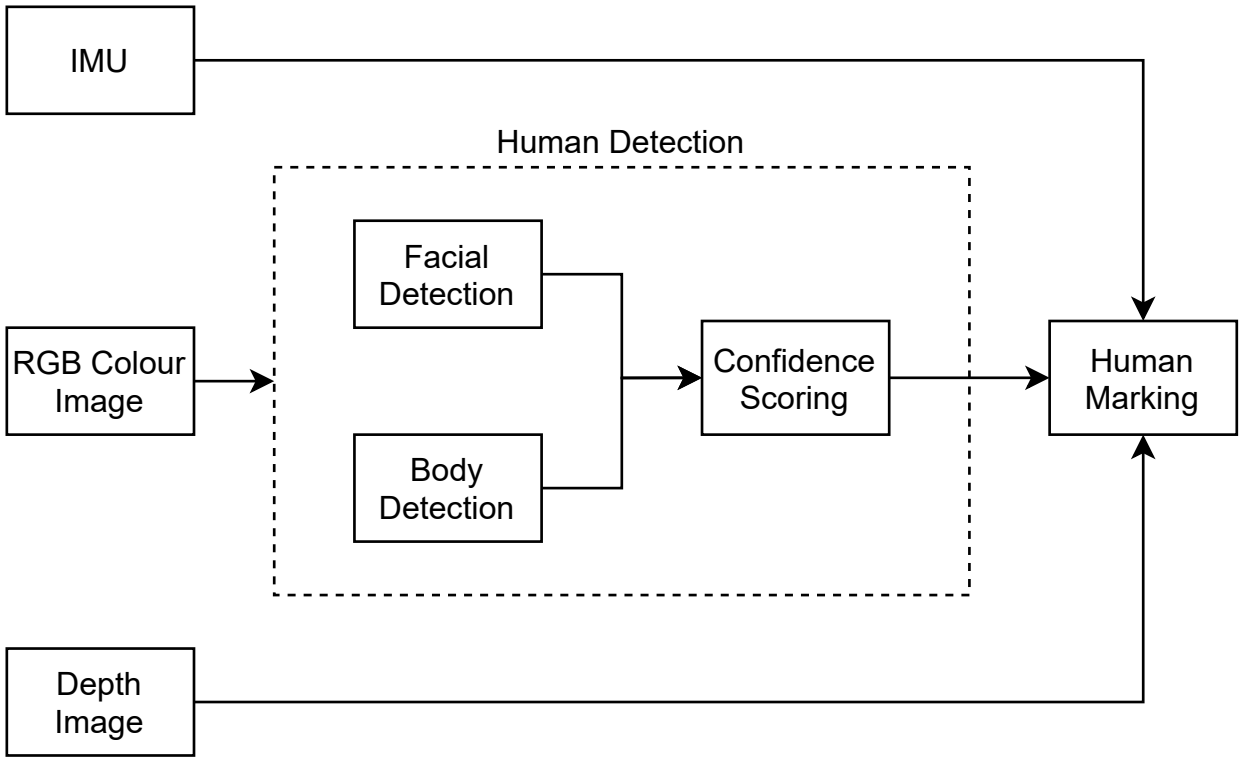


Figure 18: Human Detection and Marking Block Diagram

4.5 Ability to Meet Engineering Specifications

A rigorous test plan was developed to assess the UAV and its abilities to achieve all of the predetermined engineering specifications. Virtual simulation software was used for preliminary testing of the path planning and infrared camera to see through smoke. Real world tests were then conducted to confirm the results of the simulations and to perform tests on UAV functions that have not been assessed yet. The real world tests were able to confirm that the battery life, top speed of the UAV, SLAM, and human detection functionalities were able to meet the engineering specifications. However, the functionality of the path planning and max connection range was not able to be assessed with real world tests as a result of government regulations and COVID-19 lockdown measures.

4.6 Prototypes Developed

4.6.1 Prototype One

The first prototype design involved the mounting of four cameras and IR sensors in such a way to give as close as possible to 360° coverage in all directions around the UAV at all times. Through testing in NX 12, it was possible to achieve an approximation of the required camera positions for them to overlap and give full 360° coverage around the UAV (Figure 19). This was possible by generating a cone which emulated the same FOV of the camera option being analyzed and placing each at appropriate locations. The specific camera which was chosen for this prototype had an FOV of 160°, however to be conservative, the testing was done assuming the FOV would be 150°. The result was still the same, there were no blind spots located around the UAV, thus the desired coverage was achieved.

It was found that the most optimal setup for achieving the desired coverage involved placing the first camera on the top of the UAV and have it pointed directly upwards. The other three cameras were placed approximately 120° apart from one another and angled downwards by approximately 25° from the horizontal. These cameras would be connected directly to the Arducam Multiplexer, which would aggregate the data, and the Multiplexer would then pass the data to the Jetson Nano,

which would be used for data processing. This allowed for the cameras to overlap as required and ensured that the UAV will be able to see everything happening around it. This improves the speed at which the mapping can take place and the quality of the data being retrieved at the base station. Another benefit to having full coverage is that UAV will not be required to rotate to map an area or reverse to view an area which may have been missed while traveling forward, as the rear facing camera will be able to see it. This would greatly reduce the amount of time required to map a region of a building as the UAV can continuously travel forward into new areas without having to perform any wasted motion. Additionally, the UAV would produce increased amounts of usable data on an object due to it having multiple camera angles available and having a reduced number of potential blindspots.

Another key consideration was where to place the Jetson Nano with its case and the other new components, such as the cameras and battery. The original foam body was no longer being used so the required components from the existing Parrot AR Drone 2.0 had to be relocated as well. It was determined that the best location for the Jetson Nano and its case would be directly on the existing frame, minimizing the amount of additional materials required to mount the Jetson Nano to the UAV. The frame would not only be strong enough to handle the weight but it would also allow the weight of the Jetson Nano to be centred on the UAV. The Jetson Nano accounts for the majority of the weight which would be added, so having it in a central location helps to prevent the center of gravity from shifting during flight. The Jetson Nano case would have four holes drilled into it, which line up with the existing holes in the frame of the UAV so that the two could be bolted together as a simple, strong, and light mounting solution. The other main components which needed to be added or repositioned were the camera and IR sensor combo units, the Arducam Multiplexer, the flight controller, the battery, and the step down converter.

The Arducam Multiplexer could simply be attached to the top of the Jetson Nano case, as there is ample room for at least one board to be mounted there. Since it is not realistic to house many of the other components above the frame, some form of mounting hardware was required. A simple mounting system was designed which could be attached directly below the center of the UAV frame.

This system would hold or support a number of the remaining components. This mounting location allows for the center of gravity to be maintained at the center of the UAV and helps to minimize the overall height. The flight controller was positioned on a plate within the mounting system while the battery and step down converter were to be placed below the mounting plate using velcro as a strong yet light attachment solution. This solution posed a challenge with regards to the legs not being tall enough to allow the battery and step down converter to clear the ground, meaning new feet need to be designed. This provided an opportunity to redesign the feet to not only provide additional ground clearance but to also accommodate the cameras, which could be easily mounted using velcro.

The camera and IR sensor bundles needed to be placed in specific locations to achieve the desired coverage. To do this the first camera was mounted on top of the Jetson Nano case facing upwards, along with the Arducam board, the other three were distributed around the UAV. Two of the three remaining cameras were positioned on the newly designed legs of the UAV and the remaining one was placed on the opposite side of the UAV onto the wall of the mounting hardware mentioned previously. This positioning allowed for as close to full 360° coverage in all directions as possible without adding significantly more material or weight. The side of the drone that has the two leg mounted cameras would be used as the front of the UAV, as combined they provide clear sightlines with everything in front of them. The camera mounted to the rear of the drone, on the wall of the mounting hardware, would be used to monitor and map anything behind the UAV and would still provide a large FOV. The only downside to this location is that the legs and UAV body will block some vision.

Overall this design should have been capable of a majority of the outlined engineering specifications from a hardware standpoint, however suffered from some major software complications and limitations with data aggregation and processing. As described earlier, the main issues with this prototype design are the limitations of the Arducam Multiplexer, Jetson Nano, and integration of the SLAM software “Multicol”. The Arducam Multiplexer was not capable of handling four synchronous camera feeds, as was originally planned, making it impossible to use a four camera system with the available hardware. Additionally, the SLAM software “Multicol” required IMU data for each camera, which was not

available with the camera's chosen. This made it so that it was not viable to use this program moving forward. Finally, it was determined that the Jetson Nano was not capable of handling the required processing to perform SLAM and it added a significant amount of weight which was detrimental to flight performance and battery life. These three main issues, along with some other minor ones, were the main reasons this design could not be used for the final prototype (Figure 20).

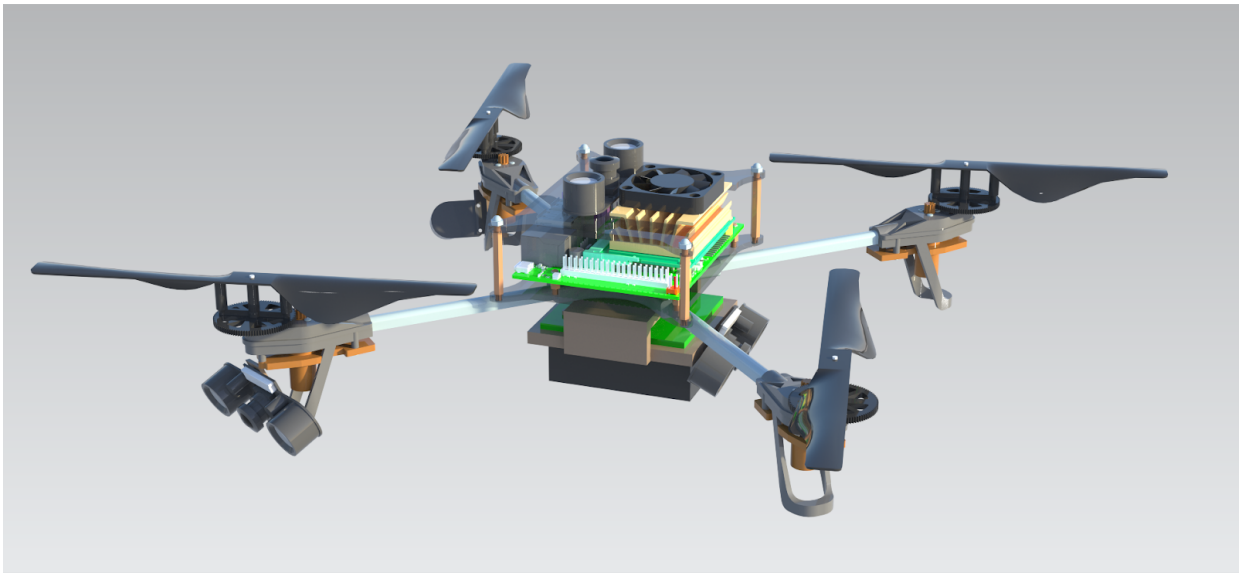


Figure 19: Prototype One: NX CAD Model

Figure 20: Prototype One: Physical Build

4.6.2 Prototype Two

Given the issues with the first prototype design, a redesign was required. It was possible to continue with the use of the Parrot AR 2.0 as the base and the upgraded 2200 mAh 11.1 V LiPo battery, however the rest of the design needed to undergo significant modifications. The first major change was the removal of the four wide angle IR cameras with a single IR stereo camera unit in the form of the Intel Realsense D435i. The D435i not only has the required IR capabilities but also includes a built in IMU, which is beneficial for SLAM performance, and has its own custom software, which allows for simple integration with ROS. These features make the D435i camera unit much less complex to implement with ROS directly, with no special hardware required, and allows the camera feeds to be used more seamlessly for performing SLAM due to the included IMU. This change alone addresses two of the main concerns from the previous design iteration, the inability to use four cameras simultaneously and having inaccurate IMU data. It also allows for the SLAM software "Kimera" to be used instead of "Multicol", as it better fits the new use case and is specifically designed for use with a single stereo camera sensor.

The third major issue encountered previously was the Jetson Nano being unable to handle the processing required to perform SLAM. Due to the Jetson Nano being incapable of handling the required processing, it was determined that the processing should be completely offloaded to a remote base station, as this would also significantly reduce the UAV power draw and weight. The decision was made to use a Raspberry Pi 4 as the main processing unit, as it is lightweight and requires much less power. The camera feed and IMU data from the D435i is sent by the Raspberry Pi 4 to the remote base station, where it can be used to perform SLAM and determine the required UAV movement to traverse an area. The movement commands are then sent from the base station to the Raspberry Pi 4, which is then able to pass them along to the flight controller. An additional benefit to this system is that the Raspberry Pi 4 can be powered directly from the flight controller, rather than requiring a 5V step down converter and additional cabling as the Jetson Nano had. This further reduces the total weight and amount of components which need to be mounted, which helps improve the UAVs flight time.

Overall, the four wide angle cameras, Arducam Multiplexer, 5V step down converter, and Jetson Nano were replaced with a Intel Realsense D435i and Raspberry Pi 4. With these component changes, the mounting hardware below the drone was no longer required, as the remaining components could all be positioned on top of the original UAV chassis, which is designed to hold the flight controller and flight recorder. With the custom mounting hardware removed and the UAV chassis being re-added, it meant that the UAV did not need the additional clearance below its body and therefore the foot extensions were removed. Removing these custom components reduces manufacturing complexity and costs while reducing the weight. These changes from the previous design iteration address all of the major concerns and make some improvements to other aspects of the design. Therefore, this prototype design was selected as the final system design (Figure 22). The NX assembly can be seen in Figure 21.

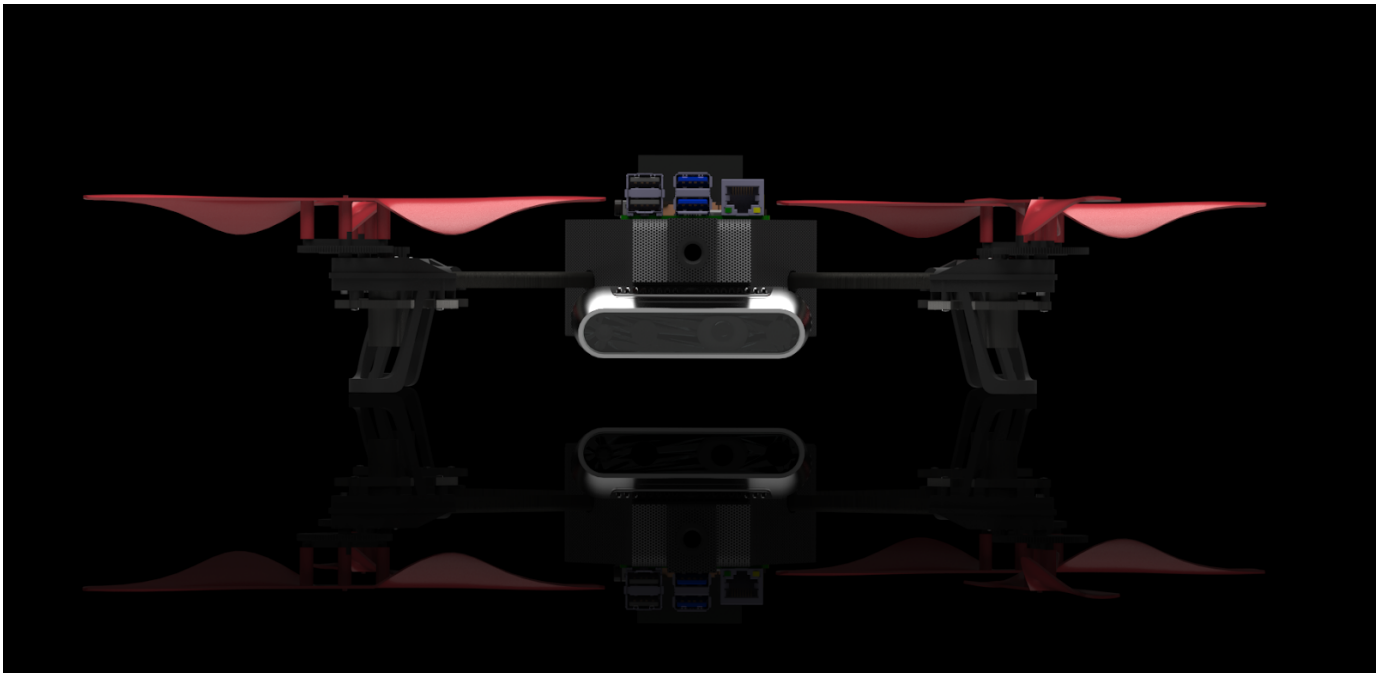


Figure 21: Prototype Two: NX CAD Model

Figure 22: Prototype Two: Physical Build

4.7 Cost Analysis

As can be seen in the Bill of Materials Table 2, the total cost of all of the system components was $997.82, excluding assembly costs. This is within the predetermined budget of $1000 for the parts, but an important consideration is whether or not the benefit is worth this cost. This is a difficult question to answer since the main benefit of the UAV is saving human lives, through quickly finding trapped victims and by reducing the risk to firefighters. The value of a human life is priceless, so comparing the benefit to the cost makes it clear that the benefits are certainly worth the cost. However, some government agencies have set a price on a human life, with the United States Department of Transportation setting the value of a human life as of 2020 at $11.6 million USD or the equivalent of approximately $14.5 million CAD [20]. Even if the same comparison is made with the cost of $14.5 million, the benefit of the system through saving lives and reducing injuries is well worth the cost

of under $1000. Based on the $14.5 million valuation, saving even a single life would make the cost worthwhile to buy up to 14.5 thousand UAVs. In 2019, there were 67 fire deaths in Ontario alone, while excluding vehicle accidents [21]. This is a significant number of deaths per year, and would be much larger if data for the entire population of Canada was included. It may not be possible to save every single one of those lives but even a small decrease in the number of deaths per year would result in a large overall benefit. Even a single life saved would make the cost worthwhile, with the potential of nearly 70 lives total saved annually being an even greater benefit. Saving an average of 67 lives annually would result in the equivalent of a financial benefit of just under $1 billion every year. Overall, it is clear that the cost is well worth the benefits, especially when considering the fact that the same UAV can be used numerous times as long as it does not sustain significant damage.

4.8 Final Prototype Design and Thought Process

The final prototype design chosen was prototype 2, as mentioned previously, and it is important to consider its ability to meet the outlined engineering specifications. This system meets the maneuverability requirements with a size of 53 cm, and the top speed was measured to be 40 km/h, matching the targeted maximum speed. This means the UAV is capable of fitting through an average sized doorway and should be able to traverse a building quickly with minimal clearance issues. With regards to connection range, proper testing could not be performed due to COVID-19, however the connection range was measured to be at least 100 m, with the expectation that it would be much larger if properly tested. The battery life of the UAV prototype was measured to be approximately 20 minutes under the limited testing conditions available. This falls just short of the desired 25 minute flight time, but a small increase to the battery capacity should allow for the target to be met. With the overall reduction in weight from the previous prototype design, down to just over 700 grams, there is enough lift still available to replace the existing battery with one of a higher capacity without significantly impacting the total weight and flight performance.

The mass of just 706 grams makes the UAV lightweight, portable and beneficial towards the overall user experience. The chosen D435i IR stereo sensor provides the system with the ability to see through smoke and work in low light conditions, meaning it can perform human identification and mapping even in hazardous environments. The engineering requirement for the UAV to be physically robust enough to handle high temperature environments could not be implemented or tested due to manufacturing and testing limitations. Finally, the UAV needs to be relatively affordable for mass use by emergency responders to be viable. As was explained in the cost analysis section, the UAV total part cost was $997.82, which is just below the target maximum of $1000. However, as was also explained previously, the true potential benefit of saving a life is in the tens of millions of dollars, making even a price over $1000 still well worth the cost. Therefore, outside of the small number of engineering specifications which could not be properly verified or needed small changes, all the requirements are met.

5 Laboratory Tests

5.1 Kimera VIO RPGO Preliminary Testing

After installation, these modules are tested in various scenarios, each with different setups. Initially, the modules are tested on a laptop with the Euroc ROSbag provided by Kimera. This test is conducted to verify functionality of the software, and that all desired features are working properly. The test was successful, recreating the results showcased on the Kimera Github page as seen in Figure **??**. Now that the modules are confirmed to work as intended, they can be tested with the third module, Kimera Semantics.

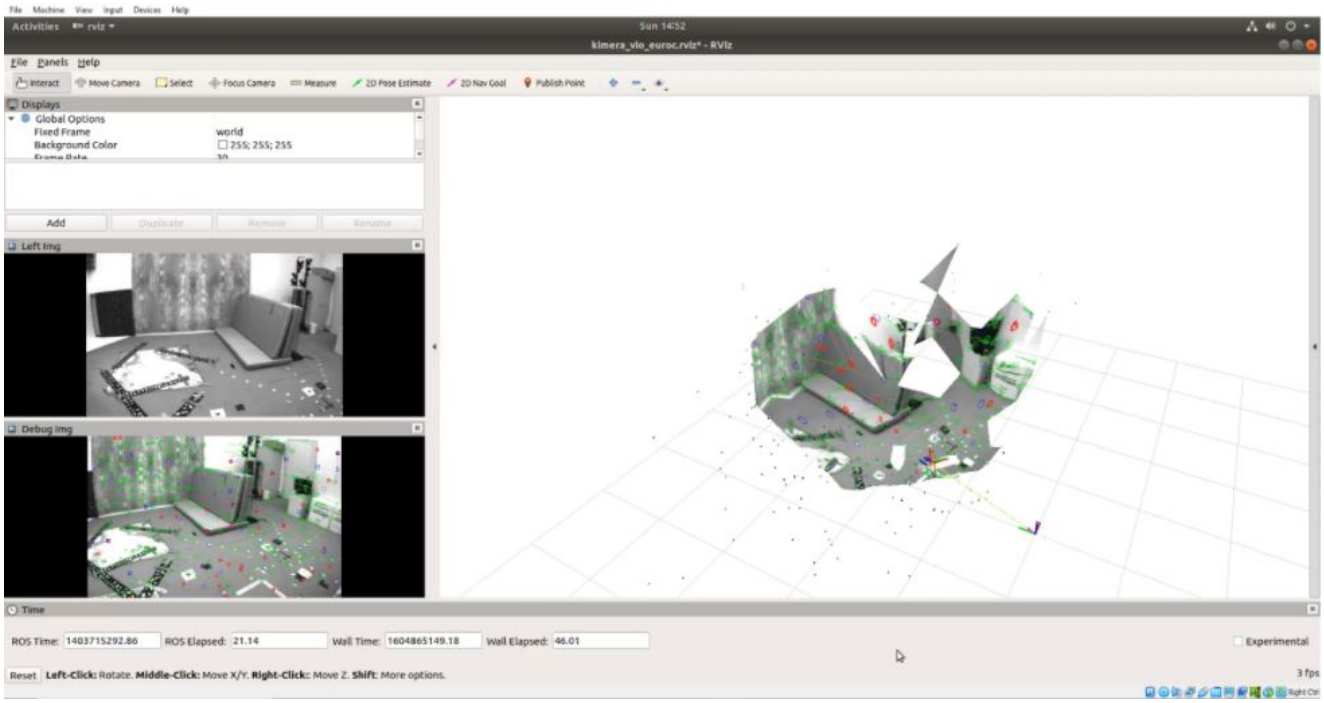


Figure 23: Kimera VIO RPGO Using Euroc ROS Bag

5.2 Kimera Semantics Preliminary Testing

It is important to distinguish that Kimera Semantics does not generate its own point cloud, but can accept one from another module, such as Kimera VIO. For this reason, Kimera Semantics, VIO, and RPGO are run simultaneously for proper functionality. When run on a laptop and fed a Euroc ROS bag, this module functioned as expected, creating an accurate 3D map of the environment viewed in the ROS bag. This result can be seen below in Figures 24 and 25.

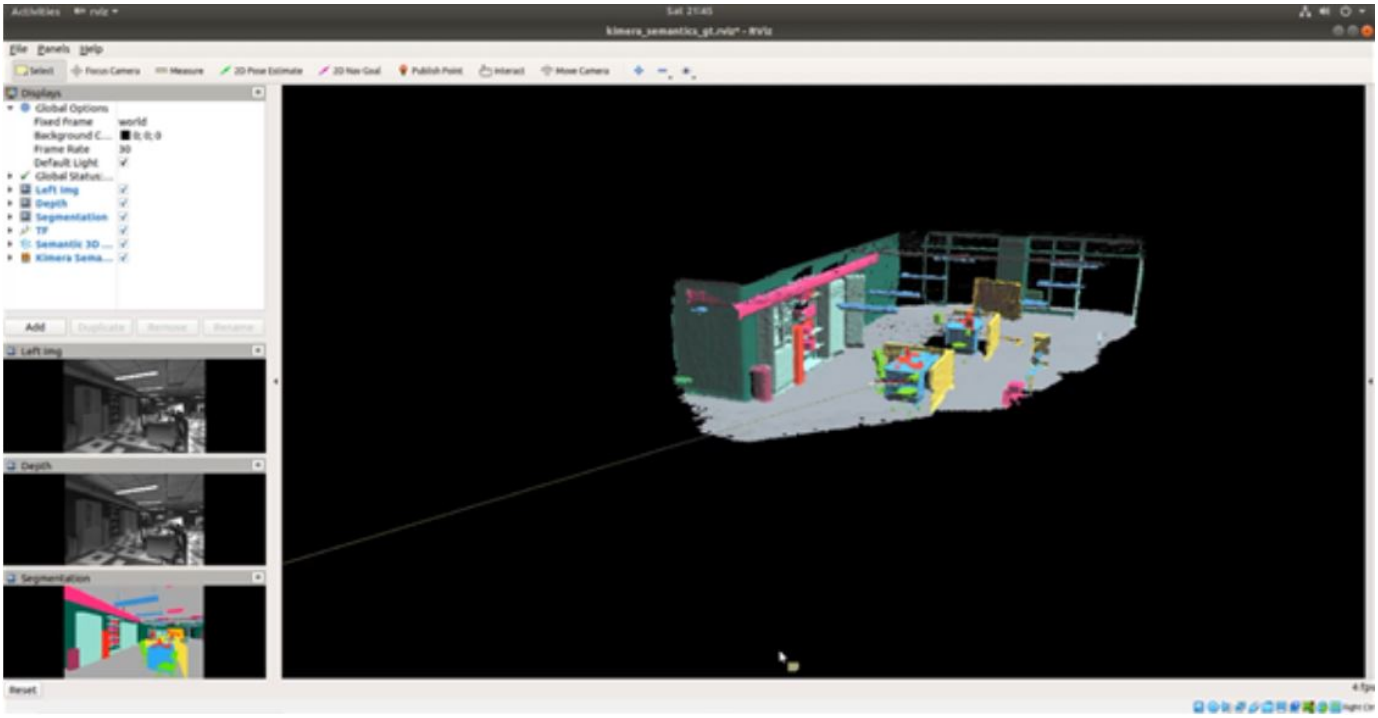


Figure 24: Kimera Semantics Using Euroc ROS Bag

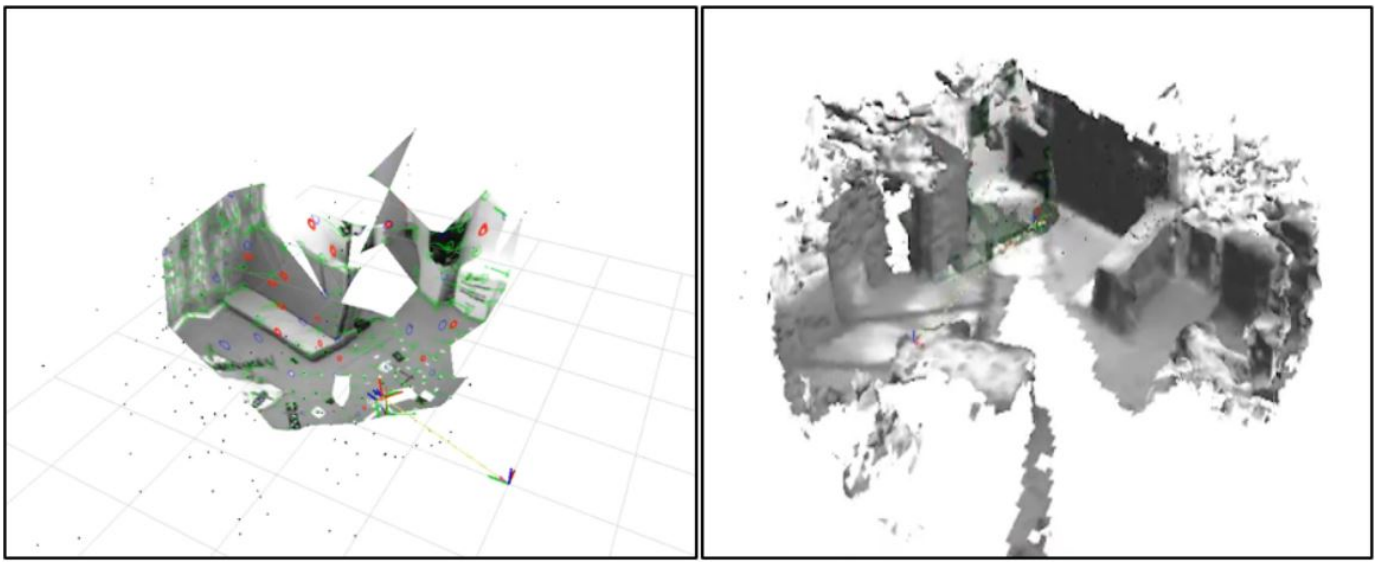


Figure 25: Kimera Module Comparison with Euroc ROS Bag

Before conducting the live tests, the parameters of the Kimera launch files need to be modified. The Kimera parameters need to be switched to online mode, enabling them to work with a live video feed, instead of a pre-recorded ROS bag. Moreover, functionality of the D435i camera needs to be verified before attempting to utilize it with Kimera.

5.3 Intel Realsense D435i Camera

SLAM Related Tests Setup

The Intel Realsense D435i camera has its own ROS launch file provided by Intel. This file, along with all necessary software, need to be downloaded prior to testing. After successful installation, the first camera test revealed that the streams were not working. It was discovered that a variety of settings needed adjustment before the camera was ready to use. First, the appropriate camera streams have to be enabled, including infra, colour, and depth. There are two infra streams to allow for stereo vision, the colour feed is useful for making more visually appealing maps, and the depth feed is necessary for 3D Metric Reconstruction. These streams also need to be assigned an appropriate resolution, and set at a framerate that the processors could handle. In order to reduce the computational load, the frame rate and resolution of each stream are set to the minimum, at 15 FPS and 640 x 480 respectively. Another key parameter is the ability to synchronize the gyro and accelerometer, as they normally operate at different rates. Setting the "unite_imu_method" to "linear_interpolation" allows both sensors to publish their data at the same frequency. This prevents any IMU inaccuracy that would be normally caused by the difference in operational frequency. After setting all the necessary parameters, the issues with Kimera not working as expected were mostly resolved. Further testing was now possible that live mapping was now functioning. Upon additional testing, it was discovered that if the localization of Kimera was off, it would overlay new areas onto existing areas, causing an unusable map to be constructed. An example of this can be seen below in Figure 26.

Figure 26: Distorted Map with Erroneous IMU Calibration

Rigorous testing revealed that this was a problem with the IMU of the D435i camera not being tracked properly. At this point, a calibration of the IMU was performed, and the issue was resolved.

Smoke and IR Testing

As a result of COVID-19, testing equipment and facilities were severely limited for this aspect of testing. However, the robustness of the IR camera still needed to be verified. In order to recreate a dark and smokey environment, a campfire was constructed at night time. To better demonstrate the IR functionality, a group member stood behind the smoke generated by the fire. A side by side comparison of the Intel Realsense D435i feeds can be seen in Figure 27, showcasing the difference between IR and RGB camera footage. Both feeds were taken at the same timestamp and location.

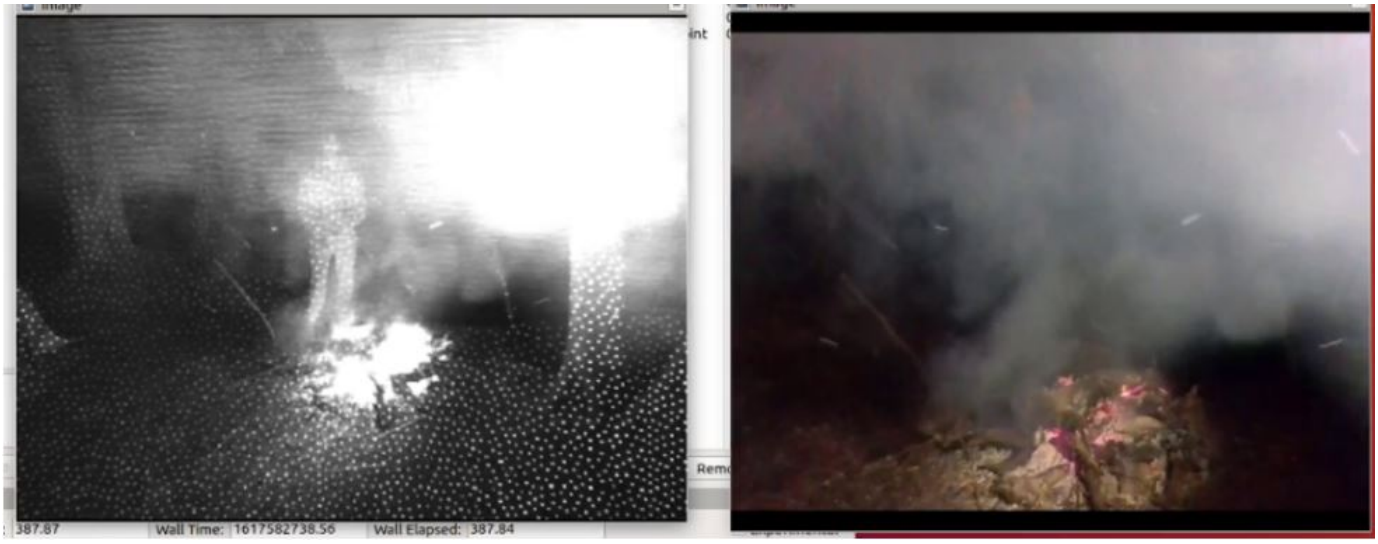


Figure 27: IR vs Colour Feed Comparison in a Dark, Smokey Environment

Based on the results seen from the side by side photo comparison, two key requirements were verified. The D435i camera was deemed as highly effective for vision in the dark, and for smoke-filled environments. More images of the feed can be viewed in the Appendix in Figures A.2, A.3, and A.4.

5.4 Kimera: Infrared

The infrared stream is the most important feed, as it provides stereo vision and the ability to perform SLAM in dark or smoke-filled environments. Now that both the camera and SLAM software are working, a live test utilizing the infrared stream was conducted. For this test, only the depth and infra streams are enabled. By panning an apartment room with the camera, and viewing the Kimera modules in RVIZ, it was confirmed that all modules of Kimera work with the hardware available. An image of the infrared 3D Metric Reconstruction can be seen below in Figure 28.

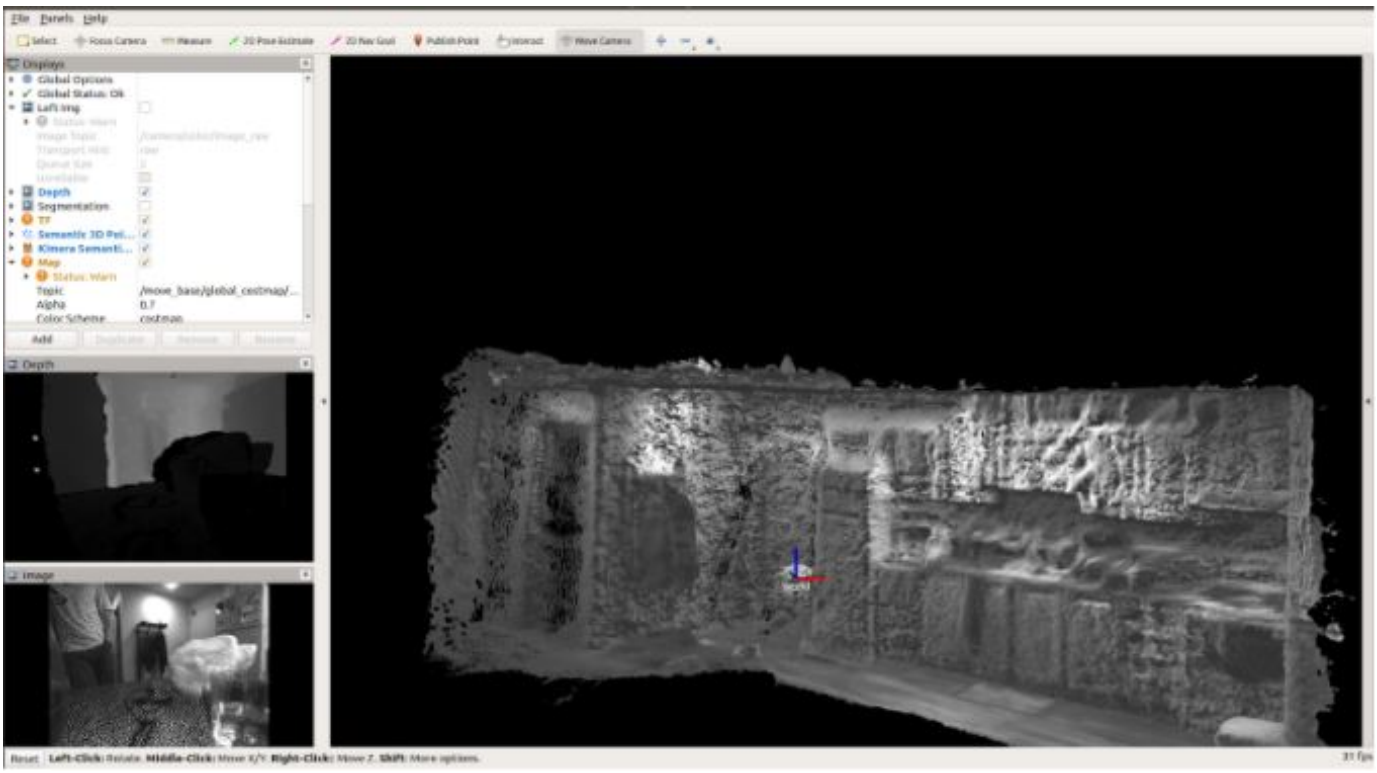


Figure 28: Infrared 3D Metric Reconstruction of Apartment

5.5 Kimera: Colour

Although not necessary for functionality, the colour feed makes the 3D reconstruction easier to understand, and may assist firefighters with discerning key locations. For this reason, Kimera was also tested with colour. A colour version of the map seen in Figure X, can be seen in Figure 29.

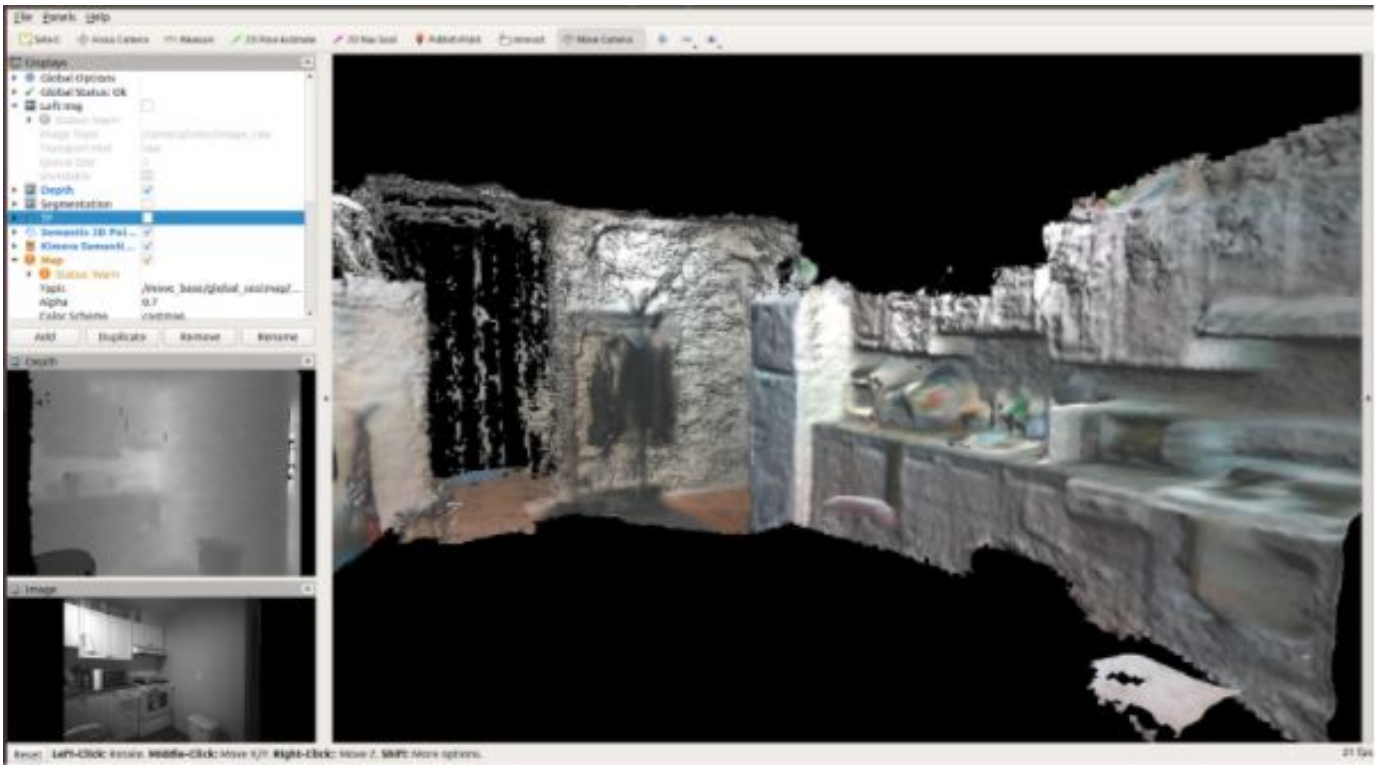


Figure 29: Colour 3D Metric Reconstruction of Apartment

5.6 Human Detection

When setting up a test plan for the human detection software, the goal was to identify the reliability of the detection system. The Intel Realsense D435i was used as the primary camera while testing this software. The first trial involved one individual and was using solely the face detection program. It was determined that a person with glasses was harder to distinguish than a person without glasses. The next trial involved two people. The software was easily able to identify both subjects and would accurately mark and track them. However, during testing, the software would occasionally identify other objects as faces. This unnecessary noise was undesirable and would need to be filtered out in the future. The next trials involved using the body detection software individually and then again

with the face recognition. Those tests proved the body recognition software was fairly reliable but struggled to pick up individuals due to lightning conditions and distance. Figure 30 shows an example of a test using the facial recognition and body detection software. After the scoring algorithm was implemented, another test was conducted to determine the new accuracy of the system . This test was successful as the program would successfully filter out the noise and would identify people with high accuracy. A few more tests were conducted to fine tune the desired score value to more reliably mark individuals. After a few iterations, it was determined the ideal score for recognizing people consistently was a confidence score of over 70%.

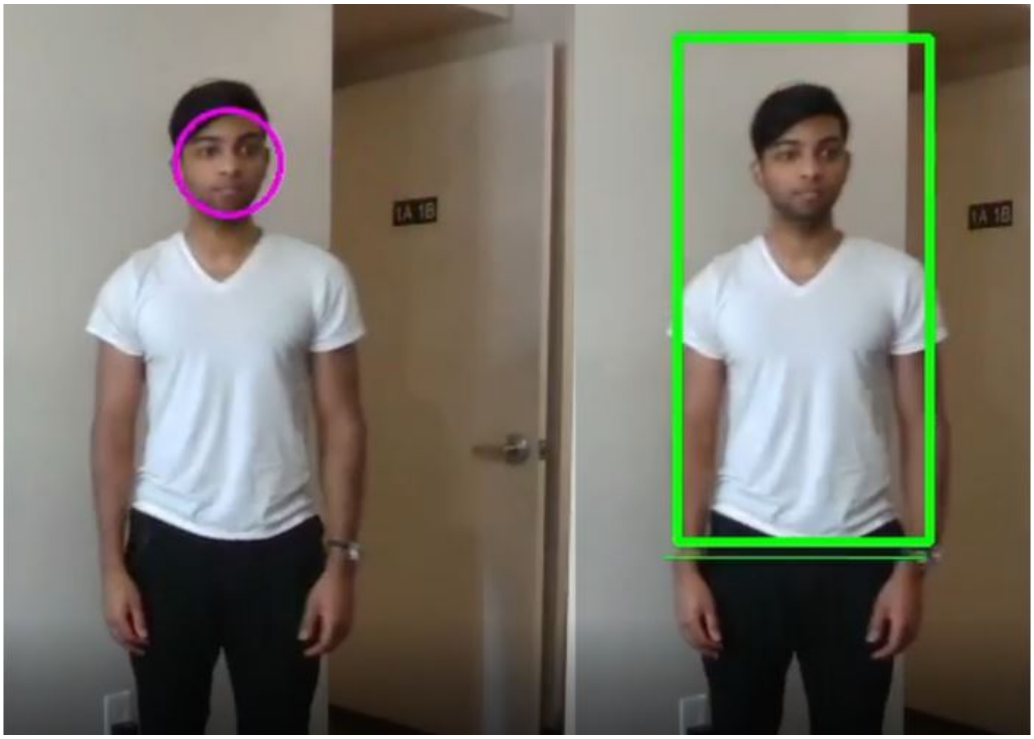


Figure 30: Facial Detection (Left) and Body Detection Tests (Right)

5.7 Human Marking

The human marking program was tested using both live data and pre-recorded data. Initially, it was tested with the 3D metric construction SLAM program using the Intel Realsense D435i camera. However, the testing computer would often struggle under the intense computational load and often result in delayed marking, an incomplete metric reconstructed map, or crashing of the program. Although the live testing was not optimal, it appeared to show the marker program correctly marking the location of identified individuals. To further test the program, a prerecorded rosbag was used. The rosbag reduces the load on the computer since it does not have to process the input stream provided by the camera. With the rosbag, the marker program would place the human indicator when the system was detecting a person and would have it stay in place when not in view. This remained successful as the 3D map of the room was able to be built successfully with the person marker in the correct location. Furthermore, no false markers were placed due to noise. Figure 31 shows a sample test of the marker program being displayed using RVIZ.

Figure 31: Human Marking Test

5.8 Autonomous Path Planning and Exploration

To test the autonomous path planning of the UAV a safe environment was needed that would give real world results. The previously discussed UAV simulators called Hector Quadcopter and Gazebo were used to provide this environment. A Gazebo world was made to have multiple rooms and hallways for the UAV to explore. The created Gazebo world can be seen in Figure 32 below and some modifications would be made during different tests to provide different scenarios.

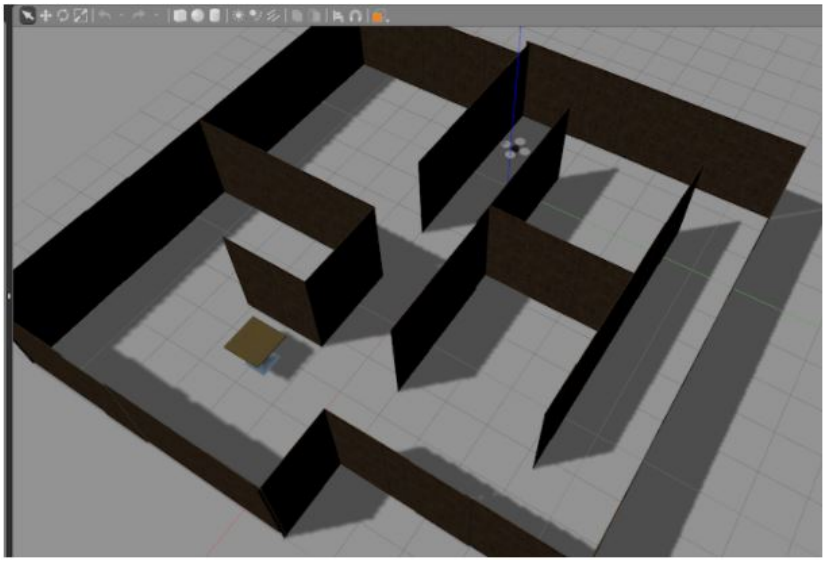


Figure 32: Gazebo World for Autonomous Simulations

Once the Gazebo world was ready and the UAV model spawned into it, a pre-made launch file would initiate all of the programs used for autonomous navigation. As expected, a 2D SLAM map and the two costmaps would be created. The UAV would also take off and ascend to an altitude of approximately one meter. A goal would then be set using the frontier exploration of explore-lite. Next, the UAV calculates the trajectories using the DWA planner and begins its exploration. The global and local trajectories would continuously be updated towards the most optimal frontier. The following Figure 33 shows the UAV simulation during exploration.

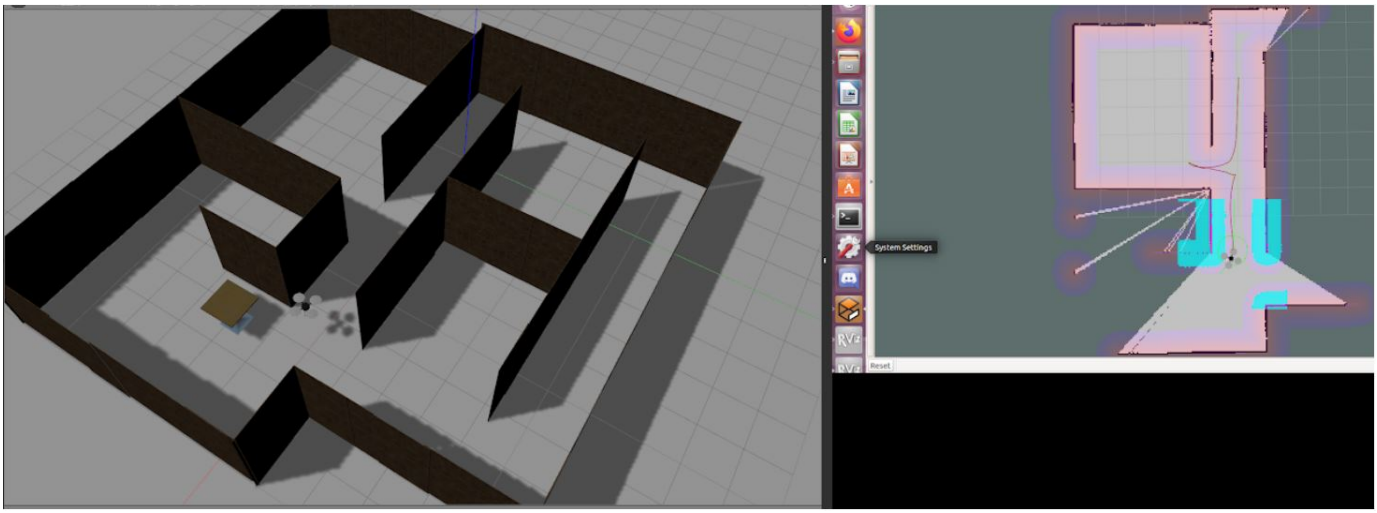


Figure 33: UAV Simulation Exploring Unknown Areas

During exploration, a table is placed that is almost one meter high in height. The purpose of this is to test the altitude control of the UAV with the ultrasonic sensor. When the UAV flies over this table, it adjusts the altitude as seen in Figure 34 and makes it fly higher temporarily. Once the ultrasonic sensor does not sense the table anymore the altitude will return to the default one meter.

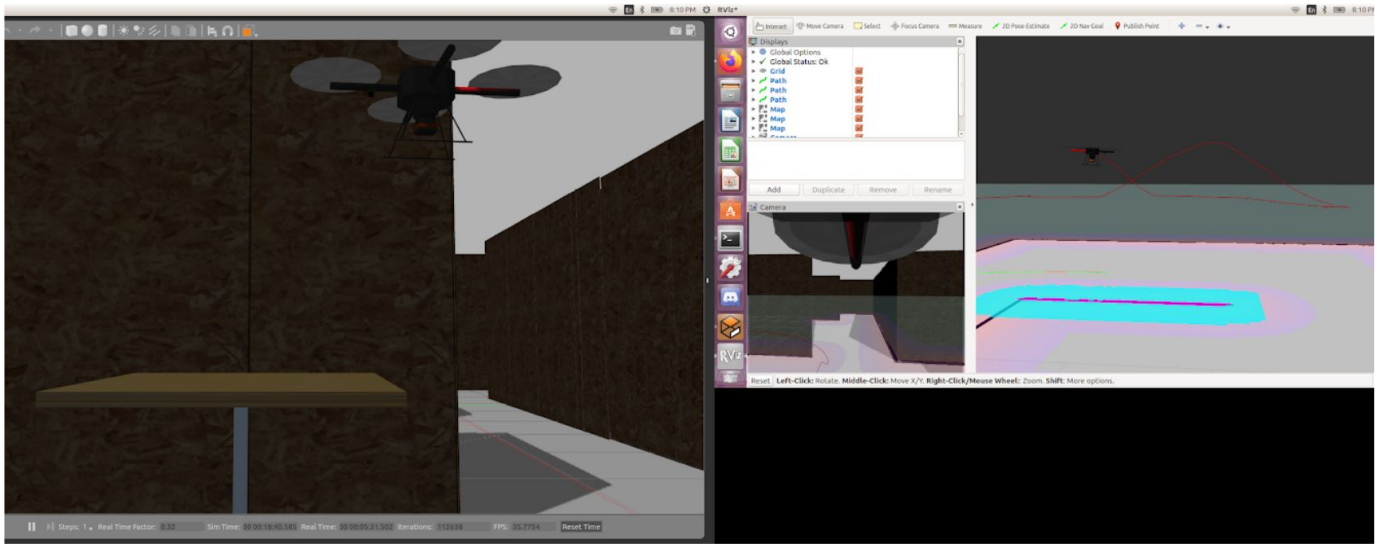


Figure 34: Altitude Adjustment During Simulation

Once all of the rooms have been mapped the explore-lite program will notify that there are no more frontiers to be explored. The return home program will then initiate as expected and take the UAV back to the saved starting location. Once the UAV returns to its initial position it will perform no more actions. The entire area was explored and the outputted 2D SLAM map confirms that all possible frontiers were explored as seen in Figure 35.

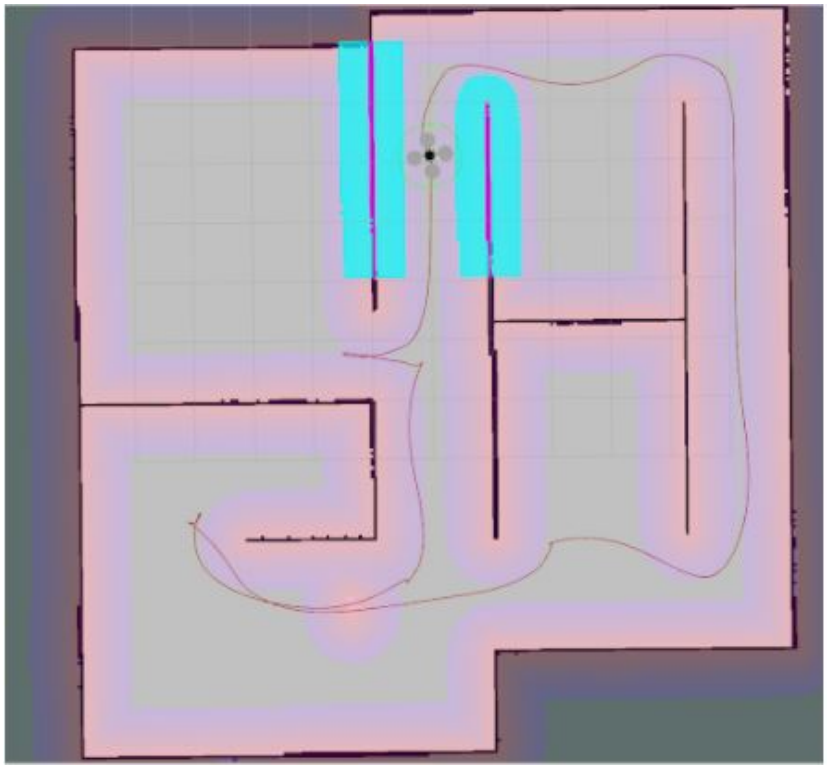


Figure 35: Explored Simulation Area for Autonomous Exploration

5.9 Flight Tests

When conducting flight tests of the UAV, the primary goal was to determine the stability of the drone while hovering and under movement controls. The secondary goal was determining whether the drone could be controlled using the custom designed GUI. Figure 36 depicts the custom GUI on a laptop screen. Flight tests were performed periodically throughout the design process when modifications to the frame and prototype were being confirmed. This was, as mentioned previously, to determine stability of the drone. With all the previous interactions of the prototype, the drone was determined to be unstable during the flight tests.

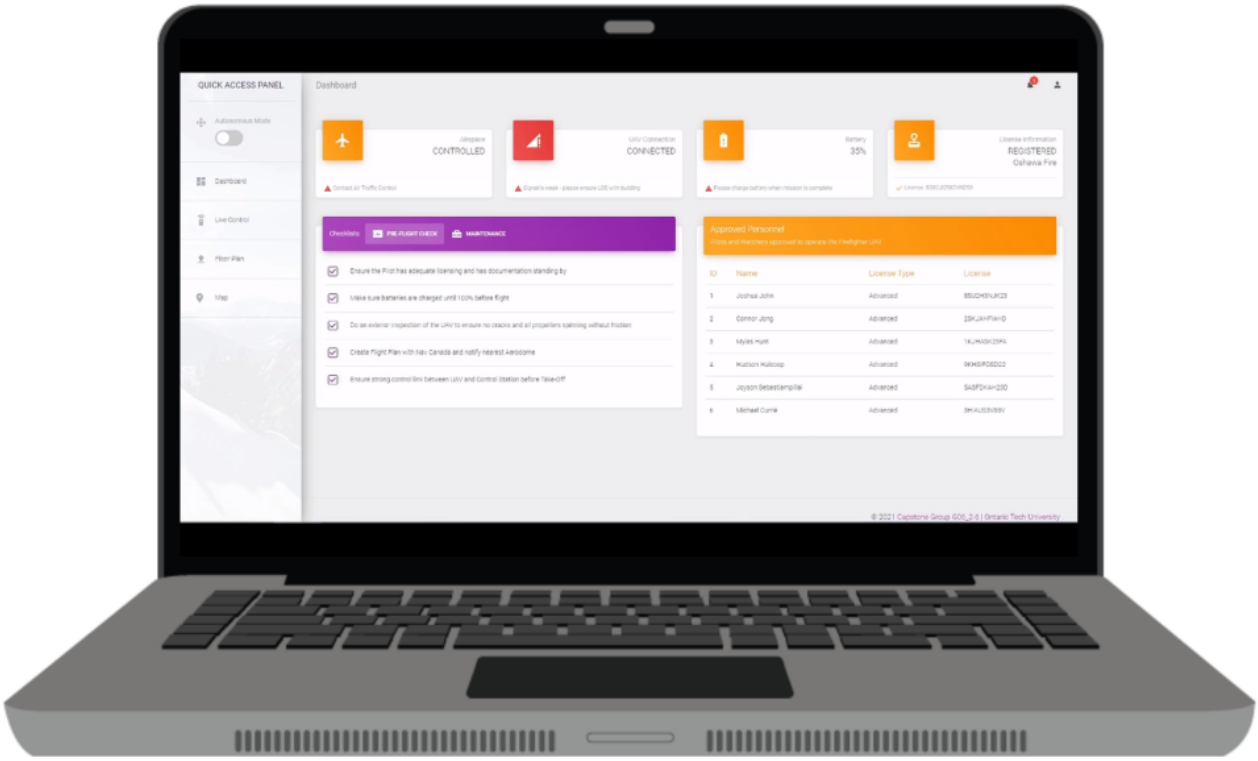


Figure 36: Cross Platform Guided User Interface

With the current prototype design iteration finished, the flight test was the last check to determine whether the design would need additional revisions. The UAV was taken to an outdoor environment to begin flight testing. Using the default controls provided by Parrot, the drone was able to take off successfully under its own weight and was able to hover in a stable manner. Altitude manipulation was tried next. The drone was found to be able to successfully control its altitude. Finally, the movement of the drone was tested, as well as its ability to pivot and turn mid-air. Under these trials, the drone was able to successfully pivot and maneuver in 3D space. Figure 37 shows the drone hovering during one of the flight tests.

Figure 37: Flight Testing

The final test involved the custom designed GUI. This was to confirm that the movement controls would be able to be carried out through a separate API and be used with the autonomous navigation module. Once connected to the GUI, the drone performed as if it were connected to the default Parrot controls. This helped clarify concerns related to latency issues and controls. Furthermore, with this being successful, it could allow for UAV testing using autonomous navigation commands.

However, before testing the autonomous navigation in a real-world environment, some of the issues that arose through testing need to be addressed. The primary issue being the movement in the X-Y plane. During testing, the drone would have to fly at a controlled speed to prevent losing control. This was due to it being slightly front-heavy in nature. Furthermore, when the speed of the drone was above a certain threshold, the drone would appear to dip downward and decline in altitude as it moved forward. These issues would have to be factored into the autonomous navigation program before testing it using the UAV.

6 Bill of Materials

Part	Manufacturer	Part Number	Availability	Quantity	Cost Per Unit (Total Cost)
2200 mAh 11.1V LiPo Battery	Ovonic	ASIN: B07CVCKHQ8	Amazon Canada	1	$29.99
Central Cross Frame	Parrot	PF070036AA	Amazon Canada	1	$114.53
Drone Motor w/ Motor Controller	Parrot	PF070040AA	Amazon Canada	4	$58.45 ($233.80)
Drone Propellers	Parrot	PF070045AA	Amazon Canada	4	$4.99 ($19.99)
Flight Controller	Parrot	PF070039AA	Amazon Canada	1	$57.91
Intel RealSense Camera D435i	Intel	00735858403931	Intel	1	$325.32
Navigation Board	Parrot	PF070041AA	Amazon Canada	1	$58.31
Raspberry Pi 4 Model B 8GB	Raspberry Pi Foundation	8GB-9006	BuyaPi	1	$144.47
SanDisk Ultra 32GB microSD	SanDisk	ASIN: B073JWXGNT	Amazon Canada	1	$13.49
				Total:	$997.82

Table 2: Bill of Materials

7 Gantt Charts

Project management was done using a web service called "ClickUp" throughout both semesters. The project was broken down into sections such as electrical, mechanical, SLAM, and path planning. Each section would then have individual tasks that needed completion. These tasks would be assigned to the group members responsible for completing them within a set deadline that can be seen on "ClickUp". The stage at which a task is at could be updated on "ClickUp" so all members knew the progress one. Members would also give updates on the current tasks assigned to them at routine meetings held 3-4 times per week. The Gantt Chart feature on "ClickUp" would give an organized timeline on all scheduled tasks. A Gantt Chart for the Fall semester and Winter semester can be viewed in Figure 38 and Figure 39 respectively.

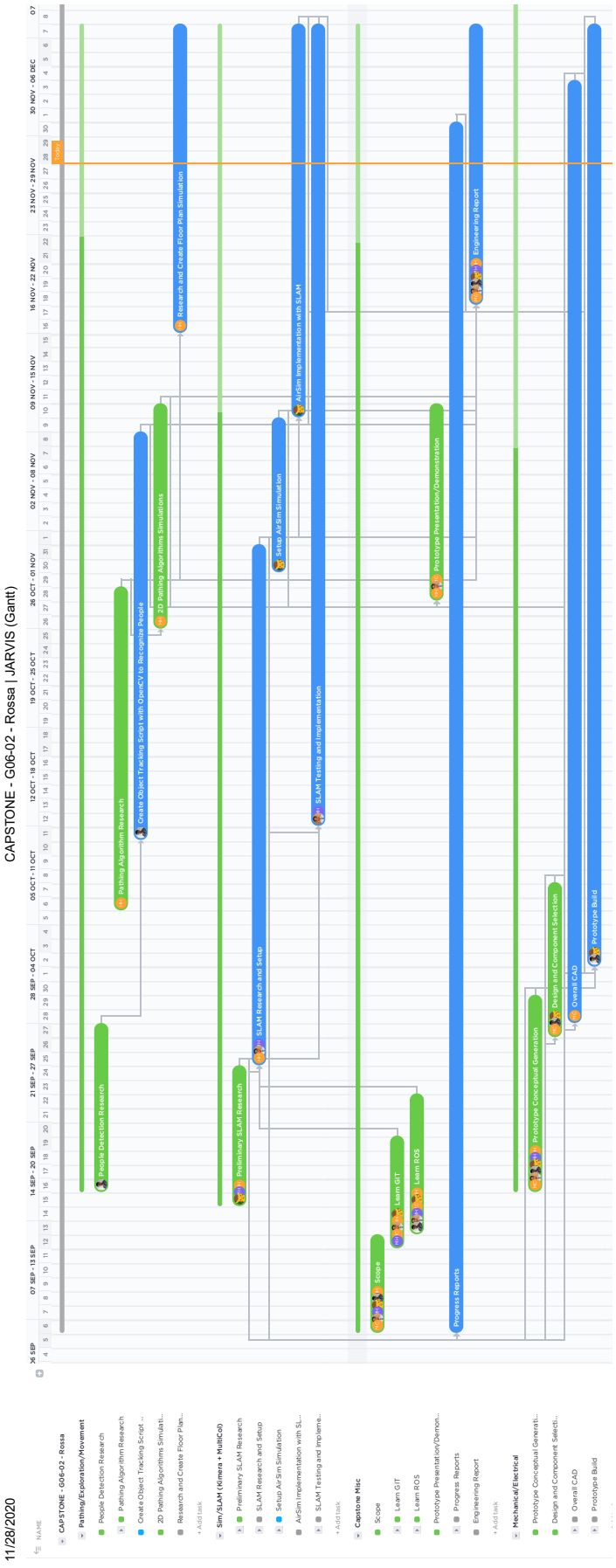


Figure 38: Fall Semester Gantt Chart - [Sept-Dec]

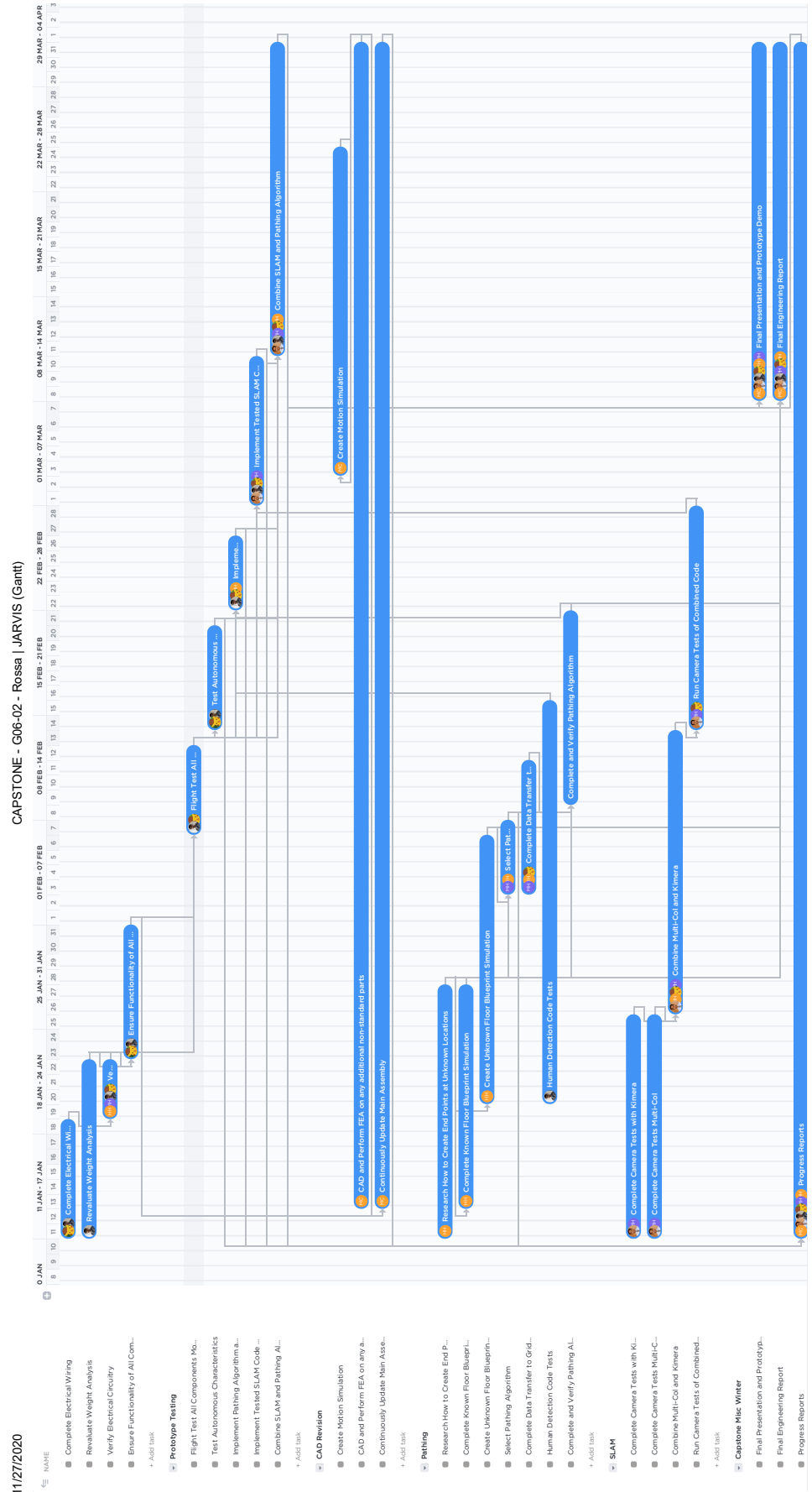


Figure 39: Winter Semester Gantt Chart - [Jan-Apr]

8 Ethical Considerations

Throughout the design process of the UAV, decisions and considerations have to be made from an ethical standpoint. While the UAV is in use for SLAM or human detection, data is collected from its surroundings to achieve its tasks effectively. The data is then stored on the base station that is connected to the UAV and is kept there until the user does not need it and will have to delete it manually. The data is not sent to other 3rd party companies for external processing and is not intended to be sold to data collection companies for extra profits. The privacy of consumers must be maintained, especially during situations where they are affected by significant financial loss.

The purpose of the UAV is to aid, not replace, fire fighters, therefore it does not raise any moral or ethical concerns. Considering additional ethical considerations, the construction of the UAV was planned in a way that ensures minimal impact to the environment. All the components and materials used for the project will be ethically sourced. Moreover, in order to support the local economy, Canadian vendors will be utilized whenever possible.

For legal use of any aerial vehicles, they will need to first be legally registered and all users of the vehicle require an operation license. In Canada, UAV operators must have taken the Transport Canada Advanced Unmanned Aerial Vehicle exam, have passed the Flight Review, as well as passed any federal, provincial, and municipal fire regulations that were imposed by the Fire Department.. A flight space must be defined for the surrounding area and required altitude range for the UAV for all possible scenarios. If the UAV happens to leave the flight space threshold, a flight termination device must halt all operations and land the UAV safely [22].

9 Considerations for Safety

When a consumer receives the UAV, it will come with an in depth user manual informing the user how to properly use the UAV. This can help prevent any injuries due to the user being more informed and better able to control the UAV in a safe and efficient manner. Along with teaching users how to control the UAV, the instructions will also inform them how to perform a thorough pre-flight checklist to ensure all functions of the UAV will operate as intended. This will reduce the chance of any issues occuring that may cause a severe malfunction. The instructions and pre-flight checklist will also show users how to properly prepare an area for the UAV to take off from. It will instruct users to designate a space with no aerial obstructions and to warn people nearby so they will not accidentally walk into the designated take-off area.

The GUI utilized to control the drone contains several features to help promote and enforce safe use of the UAV. The dashboard of the GUI shows the pre-flight checklist which will assist in reminding users to go through it. The GUI also has a feature that only allows registered users to access the controls of the UAV. This can ensure that only personnel who have received training to fly the UAV are able to operate it.

The UAV has been designed with safety in mind, in terms of software and hardware implementations. The software of the UAV includes a flight termination process that can be triggered if the UAV flies too far away from the user. The physical body of the drone will be fire-resistant to protect the electrical components contained inside. If poorly chosen materials are used, toxins can be released when the UAV comes into contact with high temperatures and dangerous sharp debris can be scattered if the UAV faces significant damage.

10 Conclusion

After a complete redesign of the initial UAVs hardware and software systems, a prototype meeting the most significant engineering specifications was successfully created. The final design is capable of autonomously navigating a building while simultaneously generating a 3D map and identifying the location of any victims encountered along the way. All of these features remain functional in common firefighting environments, such as dark or smoke-filled rooms. The information is broadcasted live to a base station for firefighters to view what the UAV is seeing and has seen previously. This will provide firefighters and other emergency responders the ability to properly assess new environments, identify hazards, locate victims, and plan out their operations more effectively. Thus, they can operate with higher efficiency and less risk to everyone involved. Less time will need to be spent in the building, since the map will depict all key information, reducing the likelihood of potential injuries or accidents, while increasing the odds for effective rescues. Therefore, the project has been considered successful in solving the design problem that was initially presented.

11 Acknowledgements

The team would like to thank Dr. Carlos Rossa for the guidance and mentorship he has provided over the course of this project. The group held weekly meetings with Dr. Rossa in order to receive critical feedback on many aspects of the project. Dr. Rossa provided useful insights and assisted with isolating key problems, while ensuring the project was developing smoothly.

The team would also like to thank the Faculty of Engineering and Applied Science of Ontario Tech University. With the funding they provided, the team was able to create a working prototype for the project which gave more opportunities in learning new experiences and practicing developed skills acquired from past courses.

References

[1] G. V. Research, "Commercial drone market size, share & trends analysis report by application (filming & photography, inspection & maintenance), by product (fixed-wing, rotary blade hybrid), by end use, and segment forecasts, 2019 - 2025," 2019.

[2] U. Government. (2020) Home fires. [Online]. Available: https://www.ready.gov/home-fires

[3] MAXST. (2019) Slam, core technology of ar, what is it? [Online]. Available: https://medium.com/maxst/slam-core-technology-of-ar-what-is-it-e6c9ae4839b4

[4] D. Scaramuzza and Z. Zhang, "Visual-inertial odometry of aerial robots," 2019. [Online]. Available: https://arxiv.org/pdf/1906.03289.pdf

[5] C. Pao, "What is an imu sensor?" 2018. [Online]. Available: https://www.ceva-dsp.com/ourblog/what-is-an-imu-sensor/

[6] C. Chopra, "Pathfinding algorithms," 2019. [Online]. Available: https://medium.com/swlh/pathfinding-algorithms-6c0d4febe8fd

[7] J. Chen, "Heuristics," 2020. [Online]. Available: https://www.investopedia.com/terms/h/heuristics.asp

[8] A. K. Anirudh Topiwala, Pranav Inani. (2020) Frontier based exploration for autonomous robot. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1806/1806.03581.pdf

[9] ROS.org. (2020) costmap_2d. [Online]. Available: http://wiki.ros.org/costmap_2d

[10] ——. (2020) dwa_local_planner. [Online]. Available: http://wiki.ros.org/dwa_local_planner

[11] Skydio. (2020) Skydio 2. [Online]. Available: https://www.firerescue1.com/fire-products/communications/articles/5-drone-technologies-for-firefighting-8wLtpDrDLmgDEReO/

[12] D. Fly. (2017) Drones in the field: Firefighting drones. [Online]. Available: https://www.dronefly.com/firefighting-drones-drones-in-the-field-infographic

[13] F. Rescue1. (2014) 5 drone technologies for firefighting. [Online]. Available: https://www.firerescue1.com/fire-products/communications/articles/5-drone-technologies-for-firefighting-8wLtpDrDLmgDEReO/

[14] T. Sandle, "Microsoft ai simulator includes autonomous car research," 2017. [Online]. Available: http://www.digitaljournal.com/tech-and-science/technology/microsoft-ai-simulator-includes-autonomous-car-research/article/508552

[15] Texas-Aerial-Robotics. (2019) Texas-aerial-robotics. [Online]. Available: https://github.com/Texas-Aerial-Robotics/Controls-ROS

[16] ROS.org. (2018) tum_simulator. [Online]. Available: http://wiki.ros.org/tum_simulator

[17] ——. (2014) hector_quadcopter. [Online]. Available: http://wiki.ros.org/hector_quadrotor

[18] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: an open-source library for real-time metric-semantic localization and mapping," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2020. [Online]. Available: https://github.com/MIT-SPARK/Kimera

[19] T. Kuzh. (2018) Caltech samaritan. [Online]. Available: https://github.com/TimboKZ/caltech_samaritan

[20] U. D. of Transportation. (2020) Departmental guidance on valuation of a statistical life in economic analysis. [Online]. Available: https://www.transportation.gov/office-policy/transportation-policy/revised-departmental-guidance-on-valuation-of-a-statistical-life-in-economic-analysis

[21] M. of the Solicitor General. (2021) Ontario fatal fires: Fire death rate. [Online]. Available: https://www.mcscs.jus.gov.on.ca/english/FireMarshal/MediaRelationsandResources/FireStatistics/OntarioFatalities/FireDeathRate/stats_death_rate.html#: :text=In%202019%20there%20were%2067,fire%20death%20rate%20was%204.6.

[22] T. Canada. (2020) Flying your drone safely and legally. [Online]. Available: https://tc.canada.ca/en/aviation/drone-safety/flying-your-drone-safely-legally

A Appendix

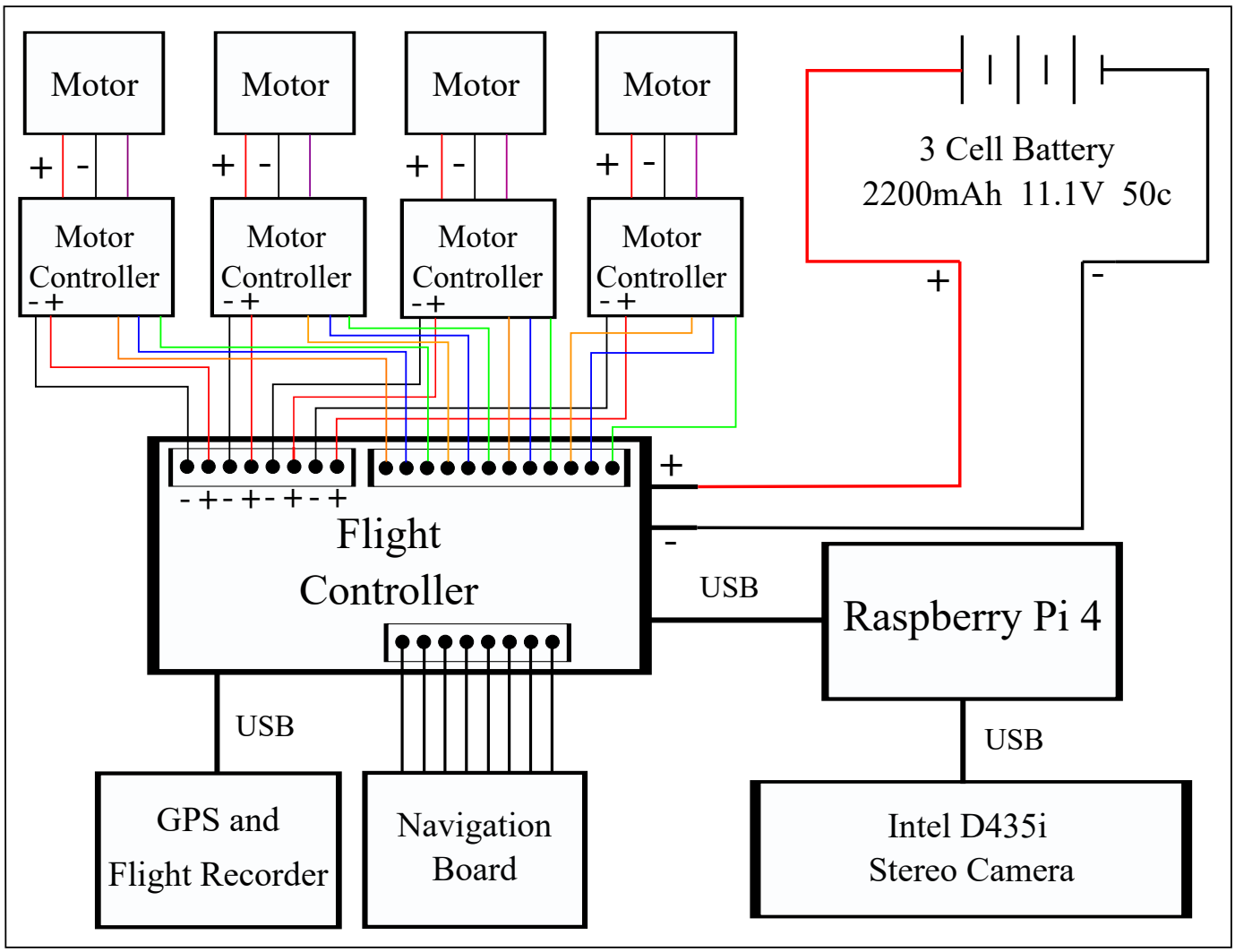


Figure A.1: Electrical Schematic

Figure A.2: IR vs Colour Feed Comparison in a Dark, Smokey Environment Example Two

Figure A.3: IR vs Colour Feed Comparison in a Dark, Smokey Environment Example Three

Figure A.4: IR vs Colour Feed Comparison in a Dark, Smokey Environment Example Four