Aerial Pathfinding Reconnaissance

Jason Gilman
Max Griffith
Mark Johnson
Dilanka Weerasinghe

CONCEPT OF OPERATIONS

REVISION 3 Draft
23 November 2020

CONCEPT OF OPERATIONS
FOR
Aerial Pathfinding Reconnaissance

TEAM 2

APPROVED BY:

Jason Gilman	Date
Prof. S. Kalafatis	Date
Souryendu Das	Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	9/6/2020	Jason Gilman		Draft Release
2	9/20/2020	Jason Gilman		Networking Subsystem Edits
3	11/23/2020	Jason Gilman		403 – Final Report Edits

Table of Contents

List of Figures

1. Executive Summary

Autonomous vehicles are becoming more popular for a litany of reasons, including safety, efficiency, and convenience. However, autonomous ground vehicles often suffer from a lack of situational awareness in attempting to navigate their environments. To combat this, product development in this field has tended towards increasing sensory capabilities – at a rapidly rising price point. Drones, while agile in their movement capabilities, suffer from poor cargo capacity. On the other hand, aerial drones offer a convenient platform for gathering data over large swathes of land. By combining the efficiency of autonomous data gathering with the capabilities of a ground vehicle, we can reduce sensory costs while maintaining its facilities.

This project's intent is to develop a prototype system which implements this relationship. To accomplish this, our proposed system will take a two-phase approach. First, the drone will survey a user-defined area and collect topographical data to be forwarded to a desktop application. The application will analyze this data to produce a map of obstacles in the area, and using the dimensions of the ground vehicle, determine a suitable path between the user's chosen endpoints. The application will then send navigation instructions to the ground vehicle to execute this path.

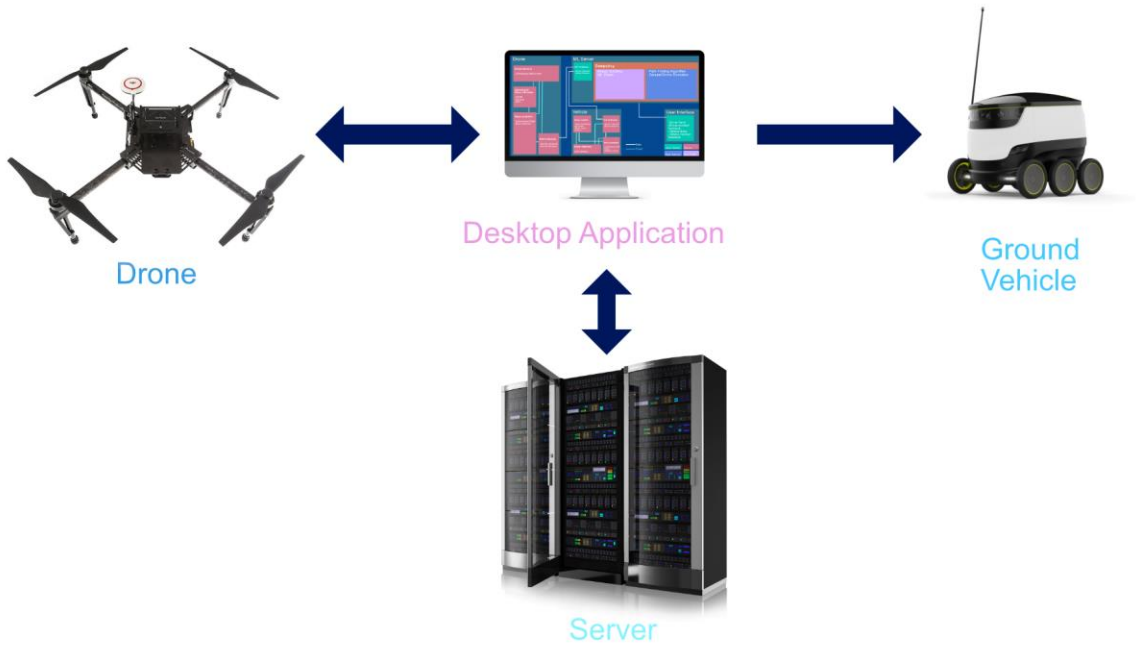


Figure 1. High-Level System Overview

2. Introduction

This document is an introduction to the Aerial Pathfinding Reconnaissance System (APRS), a system that will monitor a given area and provide a safe, reliable path for an otherwise-blind ground vehicle. This system applications are widespread, including navigation of natural disaster areas to military scouting. The APRS will serve as a protectant to the drone and the user as it eliminates human error in flight and provides an efficient and safe path to the land traveler.

2.1. Background

Aerial reconnaissance has special applications in military and industrial environments alike. Among these domains, drones equipped with LiDAR technology are sometimes used to supplement or replace traditional land surveys to provide an even more accurate and detailed physical map of the land. However, for some practical applications of aerial recon, the use of LiDAR may not be entirely appropriate, as it runs the risk of incurring unnecessary expense, complexity, and in extreme failure cases, even physical injury.

For example, consider land pathfinding. Pathfinding does not require the same degree of precision and accuracy as other industrial or military applications. In theory, the problem of pathfinding on land can, in general, be reduced to a simple distinction between traversable space and obstacles on a two-dimensional surface. For this purpose, the use of LiDAR is beyond excessive, as a reasonably accurate determination of this data can be made with much simpler hardware, such as an ordinary video camera.

The process of creating a map of an area using image data from a drone is referred to as drone photogrammetry, and it is a far cheaper process than an equivalent LiDAR solution, with industrial LiDAR drone mapping systems reaching prices between 2.5x and 10x the cost of a high-end photogrammetry system.[1] With a cost-efficiency motivation and a particular use-case in mind, this project focuses on the development of an even more hardware-flexible and economical approach than ordinary LiDAR-equipped solutions, trading a high degree of accuracy for practicality and mission-specific functionality.

2.2. Overview

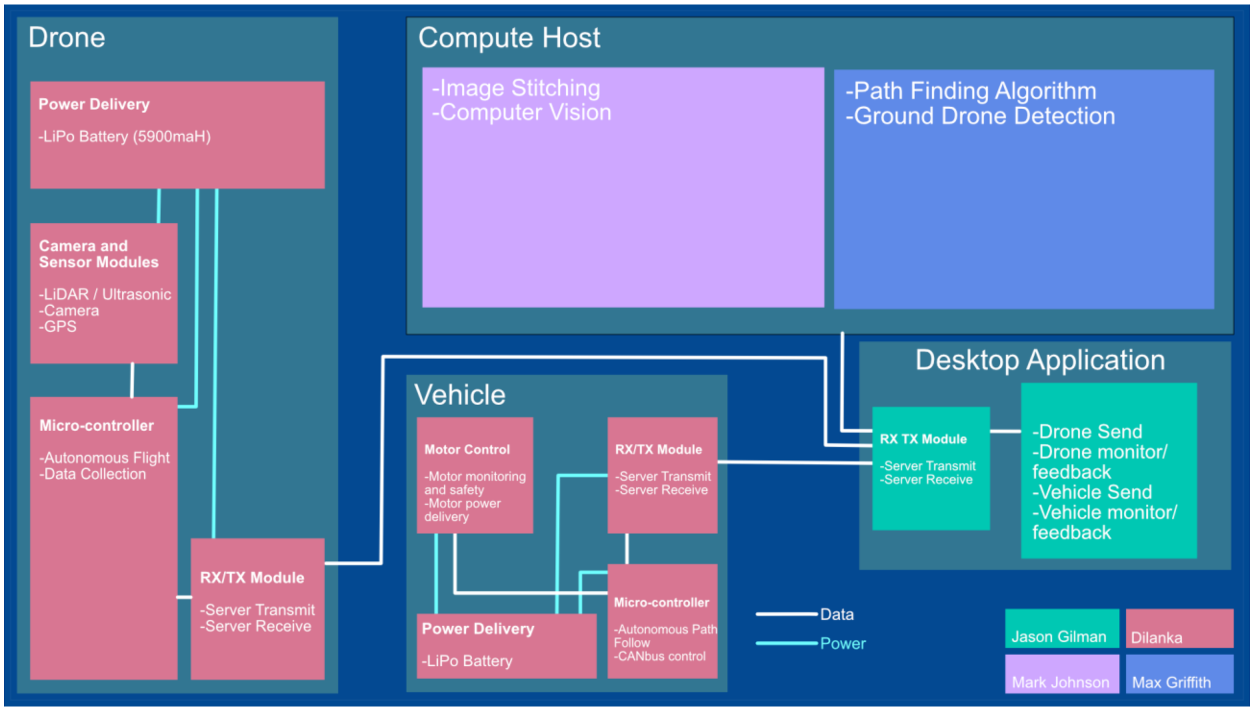


Figure 2. Functional Block Diagram

Our system uses video captured by a drone to provide a path that a user can blindly follow to its destination. We will first create a land map using video from a drone. Using an Aerial Pathfinding Reconnaissance System (APRS) application, the user will provide an area that the drone will survey. The drone will then navigate the chosen area at an altitude provided by the user.

The APRS system will utilize an array of ultrasonic sensors for object avoidance in the air. While flying, if the drone detects an area it is not capable of maneuvering it will utilize object avoidance techniques to continue its path. Upon returning from flight, data is transmitted over a wireless local network to a locally run desktop application that will stitch the land-map together and begin to generate the land drone’s navigation instructions. Using computer vision algorithms, it will be programmed to detect stationary obstacles and moving obstacles on the map of the area.

Once the full map is completed and the obstacles are pinpointed on the map, the program will path-find an efficient and safe path for the land vehicle to follow. At this point the path is relayed to a small land vehicle and the vehicle is programmed to follow the path blindly until it reaches its given destination.

2.3. Referenced Documents and Standards

[1] "Drone Photogrammetry vs. LIDAR: What Sensor to Choose for a given Application" https://wingtra.com/drone-photogrammetry-vs-lidar/

[2] IEEE 802.11 https://standards.ieee.org/standard/802_11-2016.html

[3] FAA Part 107 https://www.faa.gov/uas/media/RIN_2120-AJ60_Clean_Signed.pdf

3. Operating Concept

3.1. Scope

3.1.1. Functions of the APRS

The Aerial Pathfinding Reconnaissance System (APRS) will include two drones, an aerial drone and a land drone. The aerial drone will survey an area and transfer its data to an application which will plan out a specific path for a land drone to follow. This system will allow vehicles to safely navigate terrain without any human interaction or monitoring once the data is transported to the land drone.

3.1.2. Features of the APRS

The aerial drone will be using a camera that will locate and distinguish fixated objects from objects that can be removed (e.g. humans or other animals). The aerial drone will also have a distance sensor to accurately calculate the size of objects and the correct travel distances for the land drone to follow.

3.2. Operational Description and Constraints

The Aerial Pathfinding Reconnaissance System will be operated to identify a safe ground path for an autonomous land drone. The encompassing area of the ground path should be surveyed by an aerial drone within the timeframe of 20 minutes and be no greater than 1 km^2. Once the aerial drone has surveyed the area, the data collected from surveillance will be manipulated with machine learning to create a map that then will be deployed to a land drone, which will be able to blindly follow the designated path.

The limitations of the video capturing device will create an operating constraint for the APRS. In low light, using pure surveillance could potentially obstruct views. Any usage that would be as a means for stealth will be impacted. Furthermore, the operating range of the drone is constrained by battery capacity and the computational power available to the user.

3.3. System Description

3.3.1. Data Collection and Drone Control

The hardware attached to the aerial drone will handle the data collection, which will be used by the Computer Vision and Mapping program. There will be sensors on the aerial to handle obstacle avoidance and recording video of the ground. The land drone will lack sensors as it will follow the provided path blindly. It will have its necessary controllers to perform the required movement. The following are the smaller subsystems related to hardware on the drone and the ground vehicle:

- Power Distribution (Aerial and Ground): The drone will be using Lithium Ion Batteries for high efficiency, reliability under extreme weather and weight constraints. This system will power the drone cameras, sensors and transmission to the application. The drone is capable of holding one 8000mAh LiPo batteries and this will allow for 30 minutes of flight time. A LiPo battery will also be used to power the ground vehicle. A series of buck/boost converters will translate the LiPo voltage that will power the microcontroller, motor controller and motors on the system.

- Camera (Aerial): The camera will take video of the ground at certain points in the user defined area. Upon landing, the data captured by the camera will be transmitted to the desktop application for further software analysis.

- GPS (Aerial): GPS connects to orbiting satellites to triangulate the location of a sensor on the earth. GPS will be used to triangulate the precise location of the drone so that the computer vision can create an accurate map of the area. Without the use of GPS, the camera data would not be connected to a given location.

- Controller (Aerial): All system sensors are monitored through the mRo x2.1 Rev2 controller. This flight controller also handles the path generated by the user preferences. This controller runs the firmware for PX4 and ArduPilot. Through the ArduPilot firmware, MAVlink and MAVproxy be modified to accommodate for any flight path and sensor detection needed. The flight controller will be sent commands via a Raspberry Pi. The Pi micro controller will also have the wireless data transmission capabilities that will relay information back to the application.

- Controller (Ground): The control of the ground vehicle is monitored and controlled by a Raspberry Pi. Connected to the control unit is a Sabretooth motor driver that will act as the bridge from command to motor.

3.3.2. Computer Vision and Mapping

Once video data has been received by the desktop application, a set of custom software will analyze the data and produce an overhead view of the entire traversable ground area. First, the video data will be consumed in pieces to produce stereographic ground views, with moving obstacles eliminated from the data. These stereographic views of each area will then be analyzed further to produce a depth estimation map. With some lightweight mathematics performed over this field, a final map can be produced which delineates areas in which too great of a height difference is encountered at once, forming an obstacle.

3.3.3. Pathfinding and Drone Detection

The land vehicle's starting position within the environment and the ideal ending position will be sent to another subsystem. After these values are specified, this subsystem will use the dimensions of the vehicle and a two-dimensional map of obstacles to produce a path, converting these directions and distances into commands for the land vehicle to follow.

3.3.4. Device Networking and UI

Utilizing a locally run desktop application and microcontrollers onboard the aerial and land drones, data will be routed through a wireless local network in our system. A UI will keep the user informed of the system's progress, as well as act as a controller for the system. The system can be launched, paused, and stopped through use of the UI.

3.4. Modes of Operations

The Aerial Pathfinding Reconnaissance System will have a single mode of operation. Initially, data is collected from sensors onboard an aerial drone. Upon returning from flight, the data will be sent through a wireless local network to a local machine that is running the systems application. The application will analyze the collected data and generate navigation instructions. Finally, the navigation instructions will be sent through a wireless network to an autonomous vehicle. A user interface on the application will be utilized to launch, track, and update the system.

3.5. Users

The installation for the Aerial Pathfinding Reconnaissance System will require installing the desktop application and obtaining operational aerial and land drones. The data and means of operating this system will be on a user interface contained in a desktop application.

The controlling of the aerial done will require a coordinate system so that the drone will only survey a specified area with height requirements that the drone will follow. The parameters of this system will have to be determined by the user within the desktop application.

3.6. Support

Support for the Aerial Pathfinding Reconnaissance System will be provided in the form of user manuals outlining setup, UI navigation, system maintenance, system usage. The user manual will describe how to read incoming feedback from the drone and from the programmed vehicle. This user manual would be locating in the help section of the application and provided to the user in a printed format. A setup pamphlet will be provided detailing network setup as well as assembly of the drone and land vehicle.

4. Scenario(s)

4.1. Aiding Natural Disaster Relief

The Aerial Pathfinding Reconnaissance System could provide exceptional navigation assistance in dangerous environments. Natural disasters often create problems with physically getting relief to disaster sites. Utilizing an aerial drone to capture video data of disaster areas, and optimizing the object detection to avoid floods, blocked roads, or fires could expedite the arrival of aid to areas impacted by disaster.

4.2. Expanding Navigation Apps

Navigation apps such as Google Maps and Apple Maps utilize databases to find the fastest route for the user, although these databases often take a long time to map newly built roads. The Aerial Pathfinding Reconnaissance System could be deployed to newly developed areas to quickly detect roads and obstacles, map the area, and update the apps' databases.

4.3. Military Scouting

The Aerial Pathfinding Reconnaissance System can have a profound impact in a military application. By mapping dangerous environments, detecting obstructing or dangerous objects, and creating navigation instructions, military transport vehicles would be able to shift towards autonomy. This has the potential to save lives and expedite the transportation of soldiers, equipment, or aid.

5. Analysis

5.1. Summary of Proposed Improvements

The primary benefit of using advanced image processing over LiDAR technology is the potential for extreme cost reduction and maintainability in applications. As LiDAR sensors are expensive and sensitive devices, those wishing to use aerial reconnaissance for pathfinding are limited to options with potentially excessive upkeep.

Aside from being notoriously expensive, three-dimensional LiDAR is known to require extreme amounts of bandwidth. Potential alternatives, such as using multiple two-dimensional LiDAR sensors, requires the extra work of joining these very large datasets accurately on top of the initial cost. By using video, however, we have more control over the amount of data, and therefore bandwidth, required to function.

Leveraging advanced computer vision algorithms may potentially result in performance improvements as well, as processing hardware will be able to deal with the sensory data in discrete chunks, as opposed to arbitrarily large point clouds.

Furthermore, the use of video data as a primary source of sensory input can result in more hardware adaptability. For example, a LiDAR-based system may have the need to be finely tuned for a specific sensor to function effectively. Important details such as a sensor's operating range and environment may require special consideration, while in a purely video-based solution, any sufficiently detailed imaging will suffice.

Beyond practical advantages, video mapping also has the advantage that it may be safer to use in inhabited areas. Many LiDAR sensors, especially more powerful models, may use powerful lasers that can cause eye damage upon malfunction. Meanwhile, video recordings by themselves are never capable of directly causing physical harm.

5.2. Disadvantages and Limitations

The strongest limitation of a video-based approach, when compared to LiDAR, is its ability to function in low-light settings. For LiDAR sensors, which emit their own lasers or infrared light already, this is of no concern. However, an ordinary camera sensor will not be able to function in low light without its own lighting, or the use of a more expensive infrared camera.

One potential concern with using video to survey an area may be related to privacy concerns. LiDAR data is not sufficient to reasonably identify anyone without extreme precision and facial recognition. On the other hand, video data includes far more information about those observed by it. However, assuming that the system is only ever used in a public setting (i.e. in which no individual has any reasonable expectation of privacy), potential legal issues related to privacy are avoided altogether.

5.3. Alternatives

5.3.1. LiDAR

- More accurate depth estimation
- Sensors alone are very expensive
- Higher bandwidth requirements
- More dangerous on failure, in some cases

5.3.2. Machine learning

- More direct detection of static and dynamic obstacles
- High setup cost, requires long training sessions
- High runtime cost, requires powerful hardware to run
- Difficult to configure "black box" design

5.3.3. Live Pathfinding

- More stringent synchronization requirements
- May require real-time computing onboard
- Sensitive to network failure

Aerial Pathfinding Reconnaissance

Jason Gilman
Max Griffith
Mark Johnson
Dilanka Weerasinghe

FUNCTIONAL SYSTEM REQUIREMENTS

REVISION 2 – Draft
23 November 2020

FUNCTIONAL SYSTEM REQUIREMENTS
FOR
Aerial Pathfinding Reconnaissance

PREPARED BY:

Jason Gilman	Date
Max Griffith	Date
Mark Johnson	Date
Dilanka Weerasinghe	Date

APPROVED BY:

Jason Gilman	Date
Prof. S. Kalafatis	Date
T/A	Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	9/15/2020	Jason Gilman		Draft Release
2	11/23/2020	Jason Gilman		403 – Final Report Edits

Table of Contents

List of Figures

1. Introduction

1.1. Purpose and Scope

The Aerial Pathfinding Reconnaissance System (APRS) is intended to provide an effective approach to generating navigation instructions for autonomous vehicles. The APRS is able to efficiently analyze a user defined area such that an autonomous vehicle can travel between endpoints that are specified by the user. The APRS is designed so that the functionality of the system is abstracted from the end-user. A front facing UI will allow any person with basic knowledge in setting up drones to launch, update, and suspend the system. Figure 1 below shows a high-level overview of the APRS.

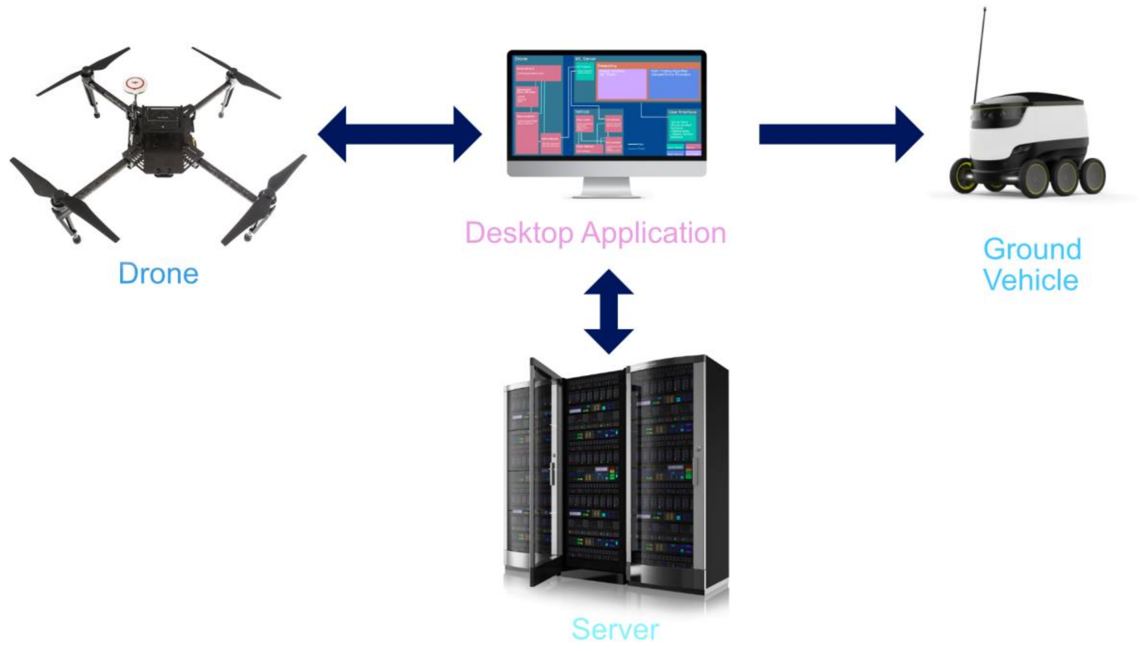


Figure 1. High-Level System Overview

The APRS' UI will be run on a local machine that will serve as the main node in the APRS network, which connects all of the devices, vehicles, and software encapsulated in the system. Data collected by the drone will be sent to the local machine so that software can be utilized to generate navigation instructions. These instructions will then be routed to the ground vehicle where they will be used to traverse the user defined area.

The following definitions differentiate between requirements and other statements.

Shall:	This is the only verb used for the binding requirements.
Should/May:	These verbs are used for stating non-mandatory goals.
Will:	This verb is used for stating facts or declaration of purpose.

1.2. Responsibility and Change Authority

Jason Gilman, the team leader, has the responsibility to confirm that the APRS follows all of the defined specifications. Changes made to project documents or system specifications must be approved by Jason Gilman.

2. Applicable and Reference Documents

2.1. Applicable Documents

The following documents, of the exact issue and revision shown, form a part of this specification to the extent specified herein:

Document Number	Revision/Release Date	Document Title
IEEE 802.11	2016	IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications
PEP 8	2001	Style Guide for Python Code
Part 107	2018	Fact Sheet – Small Unmanned Aircraft Regulations (Part 107)

2.2. Reference Documents

The following documents are reference documents utilized in the development of this specification. These documents do not form a part of this specification and are not controlled by their reference herein.

Document Number	Revision/Release Date	Document Title
	2019/04/11	Tarot 650 User Manual
	2018/08/10	DJI MATRICE 100 Product Release Notes
V1.6	2016/03	DJI MATRICE User Manual
V1.0	2015/09	Intelligent Flight Battery Safety Guidelines
MIL-STD-704F	1991/5/1	(D.O.D.) Aircraft Electrical Power Standard
SAE AS1212	2019/12/1	Commercial Electrical, Aircraft Characteristics

2.3. Order of Precedence

In the event of a conflict between the text of this specification and an applicable document cited herein, the text of this specification takes precedence without any exceptions.

All specifications, standards, exhibits, drawings, or other documents that are invoked as "applicable" in this specification are incorporated as cited. All documents that are referred to within an applicable report are considered to be for guidance and information only, except ICDs that have their relevant documents considered to be incorporated as cited.

3. Requirements

In the following section, "Aerial Pathfinding Reconnaissance System" or "APRS" will refer exclusively to the entire system. This will include the power distribution, whether that be for the aerial drone or the land vehicle, the camera to capture ground footage, LiDAR to avoid aerial objects that may be in the drones' path, GPS systems to triangulate the precise location of the drone and to calculate the land vehicle's location as it is following the path, the DJI drone controller which will be used to accommodate for any needed flight path updates, and a ground controller that will act as the bridge from command to the motor of the land vehicle. This definition also includes computer vision and mapping where video data will be consumed in pieces to produce stereographic ground views without moving obstructions which will be analyzed to produce a depth estimation map, pathfinding and drone detection which will analyze the vehicles start and end positions in an environment and the map itself to produce a path and create commands for the land vehicle to follow, and device networking and UI which will route the data through a wireless network and keep the user informed of the system's progress.

3.1. System Definition

The Aerial Pathfinding Reconnaissance System (APRS) is composed of four discrete entities:

Vehicle:
The ground vehicle will be used as a demonstration of the APRS. It contains no visual sensors for avoiding obstacles. Instead, it will receive directions from the application and execute them accordingly.

Drone:
The aerial drone will be used to gather data on the operating environment of the ground vehicle. It will use an onboard transceiver to wirelessly relay video information for further processing on the local machine.

Laptop/Local Machine:
The local machine will run a desktop application that will use video information from the drone to produce directions for use by the land vehicle. To do this, it will stitch video frames together to produce a map and to construct a stereographic view from which to infer depth. Then, a pathfinding algorithm will be used to generate the final directions.

User Interface:
The user interface provided in the desktop application will form the frontend of the system, relaying information about progress and status to the user, and allowing the user to configure pre-flight parameters such as e.g. aerial path, altitude, etc.

Shared among these four elements are four subsystems. Some of these subsystems are spread across multiple entities, while some are limited to one location as shown in Figure 2.

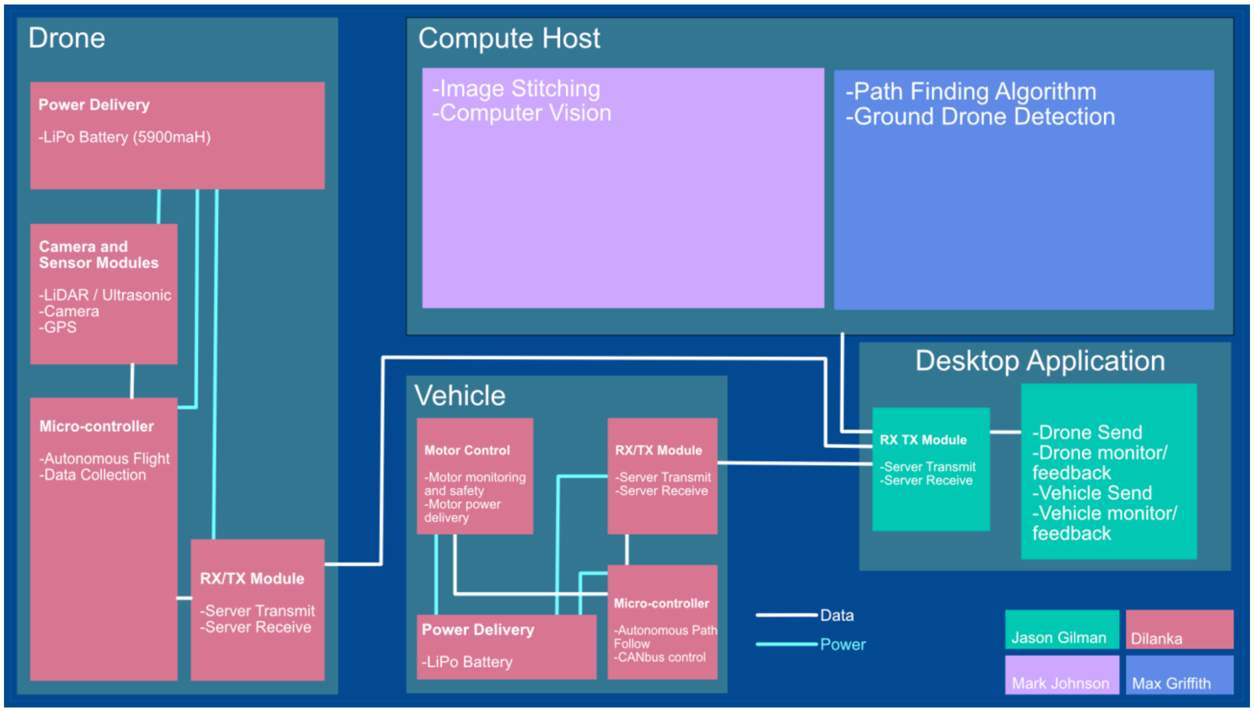


Figure 2. Functional Block Diagram

The color-coded sections of Figure 2 show which person will be managing their development, with each team member on one subsystem according to the following mapping:

• Jason Gilman:	Device Network and User Interface
• Dilanka Weerasinghe:	Data Collection and Drone Control
• Max Griffith:	Pathfinding and Drone Detection
• Mark Johnson:	Computer Vision and Mapping

The diagram also displays points of communication or otherwise shared resources. Where not specified, it is assumed that resources available to one entity are shared by all subsystems within it.

For example, consider how all of these components will be used in the system. First, the drone uses its components (power, cameras, etc.) to collect imaging data of the environment. Then, through its connection to the local machine facilitated by their transceiver modules, this data is relayed for processing. Progress on this task is relayed to the user through the interface, which can also be used to set flight parameters and other options.

The processing of this data occurs in the following fashion: First, the imaging data (in the form of a video) is stitched together to form a complete bird's eye view of the environment. After depth estimation and obstacle detection is complete, the pathfinding algorithm and ground drone detection work together to find a path from its current position to the target. Finally, after a path is found and navigation instructions are created, they can be sent to the ground vehicle (which uses its components to carry out these instructions).

3.2. Characteristics

3.2.1. Functional / Performance Requirements

3.2.1.1. Flight Survey Area

The APRS shall have capacity to survey an area of ~1km^2 in one flight.

Rationale: As required by the sponsor, the initial surveying flight must cover a defined plot.

3.2.1.2. Payload Capability

The APRS shall have capacity to carry 1400 grams in addition to its weight.

Rationale: In accordance with the specifications of the drone it cannot exceed a total weight of 3680g.

3.2.1.3. Movement Characteristics

The APRS aerial drone shall have a maximum yaw angle of 200 degrees and a maximum tilt angle of 35 degrees. This drone shall not fly faster than 15m/s.

Rationale: A characteristic of the Tarot 650 V2.1 drone details the above figures that the programs will not attempt to exceed.

3.2.1.4. Flight Range

The APRS shall have capacity to travel 3.2 m away from the user starting location.

Rationale: The telemetry modules on commercial drones can only communicate at a functional maximum of this distance.

3.2.1.5. Flight and Land Duration

The system aerial drone will be operational for up to 25 minutes in the air. The land vehicle will be able to follow various paths for up to 30 minutes.

Rationale: This aerial drone contains 1 single 5000mah LiPo batteries. Base conditions through Tarot testing show that it can remain in the air stationary for 40 minutes. The Land Drone will have a 4000mah battery that will allow it travel long distances.

3.2.1.6. System Lifetime

The Drone will not be operational after a single flight path is followed. All components of the drone, the flight controller and the Land Drone will require a charge after operation.

Rationale: After each use of the APRS, components of the system require maintenance. The aerial and Land Drones' batteries must be charged before attempting to use again.

3.2.1.7. System Networking

When transferring data between devices, the Aerial Pathfinding Reconnaissance System requires that the devices must be within 30 meters such that the devices can be connected to a shared wireless local network.

Rationale: The APRS will utilize wireless local area networks to communicate between devices. In order to facilitate communication between devices on WLAN, the devices must be in relatively close proximity to each other.

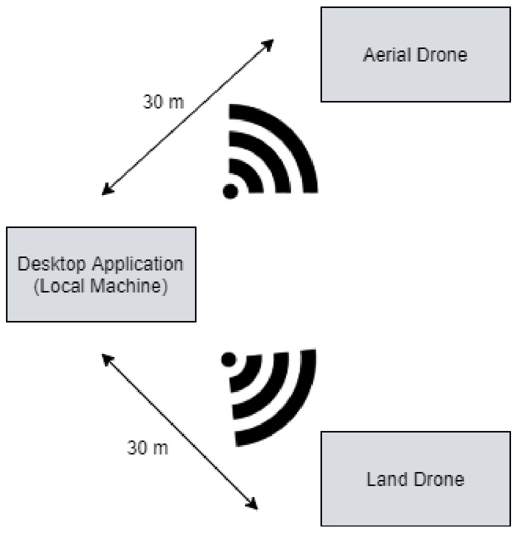


Figure 3. System Networking Diagram

3.2.1.8. Obstacle Detection

The APRS shall be able to accurately detect obstacles as tall as 5% of its altitude.

Rationale: As defined by the sensitivity of the sensors onboard the aerial drone, the APRS will be able to detect objects that meet a height requirement.

3.2.1.9. Optional Internet Availability

The system will provide the user with a mechanism for reporting their current location in the user interface using Google Geolocation Services, given they have an internet connection.

Rationale: In order to fetch the user's current location from Google, an active internet connection is required.

3.2.2. Physical Characteristics

3.2.2.1. Drone Mass

The mass of the drone in the APRS shall be less than or equal to 3 kilograms.

Rationale: This is a requirement of the drone manufacturer to maximize flight time while maintaining all necessary functions of the drone. The drone body weighs 2180 grams and the all addition sensors weighing less than 800 grams.

3.2.2.2. Land Vehicle Mass

The mass of the land vehicle in the APRS shall be less than or equal to 5 kilograms.

Rationale: This is a requirement posed by our customer detailing that the system should be transferable by hand to any location.

3.2.3. Software Characteristics

3.2.3.1. Programming Style

Software subsystems written in Python shall conform to the PEP 8 Style Guide.

Rationale: The software components of the APRS should be written in a consistent programming style for maintainability purposes.

3.2.3.2. Source Commenting

Source code for software subsystems shall contain at least one comment for every 5 SLOC, and a documentation string at the beginning of each function definition.

Rationale: The software components of the APRS should contain sufficient internal documentation to understand and maintain the source code.

3.2.3.3. Error Handling

Upon any unrecoverable error conditions, software subsystems will use the standard exception handling capabilities of their source language.

Rationale: The software components of the APRS should utilize a consistent error handling mechanism to ensure ease of integration.

3.2.4. Electrical Characteristics

3.2.4.1. Inputs

a) The presence or absence of any combination of the input signals in accordance with ICD specifications applied in any sequence shall not place the user in a state of danger, place the APRS in a state of danger, reduce its life expectancy, or cause any malfunction, regardless of whether the unit is powered or not.

b) No sequence of user command shall damage the APRS, reduce its life expectancy, or cause any malfunction.

c) In the event of fatal malfunction, the APRS shall be taken control by a user and may be stopped at any time to prevent such fatal error.

Rationale: By design, should limit the chance of damage or malfunction by user/technician error and by error of acts of god or unpredictable situations.

3.2.4.2. Power Consumption

The maximum peak power of the aerial system shall not exceed overcome the limits of the battery compartment on the drone. The voltage shall not exceed 24.2 V in accordance with a maximum of 4.5 A.

Rationale: A requirement drone systems to regulate power and maintain stable flight.

3.2.4.3. Input Voltage Level

The input voltage level for the APRS aerial drone shall be less than +29 VDC. At system power on the normal voltage limit may be exceeded at battery start.

Rationale: Aircraft bus specification compatibility, MIL-STD-704F

3.2.4.4. Charging and Capacity

The charge capacity of the battery will not exceed 99% of the 22.2 V battery operation. The maximum charging power shall not top 180W. The battery will display a low battery state when under 12.5% capacity.

Rationale: Limitations and safety features of LiPo batteries. By design to maintain battery and drone health.

3.2.5. Outputs

3.2.5.1. User Interface

The APRS will provide a UI encompassed in the system's desktop application that will update the user on the system's progression. When the system progresses from data collection to software analysis, and ultimately to preparation of the land vehicle, the user will receive a notification from the UI.

Rationale: The APRS has many steps and allowing the user to interact through a simple UI will decrease the learning curve for the system.

3.2.5.2. Low Battery Indication

The APRS will provide both an update to the UI and enable a physical notification on the drone that it is incapable of traveling.

Rationale: This requirement is to maintain the safety of the drone and the any nearby individuals.

3.2.6. Environmental Requirements

The APRS shall be designed to withstand and operate in the environments and laboratory tests specified in the following section.

Rationale: This is a requirement specified by our customer due to constraints of their system in which the APRS is integrating.

3.2.6.1. Pressure (Altitude)

The APRS can be operated within 0 to 120 meters above ground level, or 0 to 120 meters above a structure.

Rationale: This operating range is in compliance with FAA Small Unmanned Aircraft Regulations (Part 107).

3.2.6.2. Thermal

The APRS shall be operated within a temperature range of -10° C to 40° C.

Rationale: As per the user manual, the battery that is included with the drone can operate within the temperature ranges of -10° C to 40° C.

3.2.6.3. Inclement Weather Conditions

The APRS shall not be operated in any condition that involves inclement weather such as rain, snow, dust storms, high-speed winds, hail, or thunderstorms.

Rationale: The drone will not operate in any condition of inclement weather to ensure the safety of the aerial drone as it is not waterproof and cannot operate in extreme conditions.

3.2.7. Failure Propagation

3.2.7.1. Aerial Drone Flight Failure

Upon error during flight time, the user should immediately suspend drone operation through use of the remote control provided.

Rationale: Per the requirements to the Texas A&M University ECEN Department, a user must be able to suspend the operation of the aerial drone using the provided remote control upon flight error.

3.2.7.2. Low Battery

In the event of low battery, the drone shall suspend operation and return either to the user or to a location where it can be recovered. The drone will not continue operation in a state of damage or irresponsible condition.

Rationale: To maintain the safety of the drone and the any nearby individuals.

4. Support Requirements

4.1. Materials Provided

The following materials will be provided to the user.

4.1.1. Aerial Drone System

A fully encapsulated aerial drone system will be provided to the user. This will include sensors such as camera, GPS, accelerometer, gyroscope and ultrasonic. A microcontroller onboard the drone will fulfill the interaction between hardware and software and provide functionality to receive and send data from the machine running the system's application. The aerial drone system will contain batteries and power distribution such that all electrical components are able to function properly.

4.1.2. Land Drone System

The Land Drone system will be provided to the user. This system will contain the vehicle's chassis, wheels, and motors. A microcontroller will be attached to the chassis so that software can be implemented to drive the motors and receive data from the local machine running the system's application. The system will contain batteries and power distribution such that all electrical components are able to function properly.

4.1.3. Desktop Application

Software will be provided to the user that will serve as the main interface to the user. The software will provide capabilities such as device networking, data transmission and analysis, and control over the system's processes.

4.2. Local Machine Requirements

The user must possess a local machine such as a laptop that is capable of running the system's desktop application. The recommended specifications are as below:

Processor: Intel Core I5 7th Generation or equivalent
RAM: 8GB DDR3
Operating System: Windows 10
Storage: 5GB

4.3. System Malfunction or Destruction

Upon destruction of the aerial or land drones, a professional will be required to service the affected system. If at some point a portion of the system malfunctions, the user will be notified from within the UI, and will be able to make changes such as restarting or suspending the system.

Appendix A: Acronyms and Abbreviations

APRS	Aerial Pathfinding and Reconnaissance System
GPS	Global Positioning System
GUI	Graphical User Interface
ICD	Interface Control Document
SLOC	Source Lines of Code

Aerial Pathfinding Reconnaissance

Jason Gilman
Max Griffith
Mark Johnson
Dilanka Weerasinghe

INTERFACE CONTROL DOCUMENT

REVISION 2 – Draft
23 November 2020

INTERFACE CONTROL DOCUMENT FOR Aerial Pathfinding Reconnaissance

PREPARED BY:

Jason Gilman	Date
Max Griffith	Date
Mark Johnson	Date
Dilanka Weerasinghe	Date

APPROVED BY:

Jason Gilman	Date
Prof. S. Kalafatis	Date
T/A	Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	9/20/2020	Jason Gilman		Draft Release
2	11/23/2020	Jason Gilman		403 – Final Report Edits

Table of Contents

List of Figures

1. Overview

This document will provide specifics into how the Aerial Pathfinding Reconnaissance System (APRS) will be implemented. It will also describe the APRS subsystems physically and electrically, providing information on how the system will function. The document will define the interaction between the subsystems, and the requirements to enable functionality.

2. Definitions

μC	Microcontroller
REST	Representational State Transfer
APRS	Aerial Pathfinding Reconnaissance System
CV	Computer Vision

3. Physical Interface

3.1. Weight

3.1.1. Weight of Aerial Drone

The aerial drone system will weigh less than 3680g. This is to allow an average person to be able to transport and setup the drone. This weight includes the drone chassis, sensors, battery, power distribution components, and microcontroller.

3.1.2. Weight of Land Rover

The Land Rover system will weigh less than 50kg. The Land Rover system will need to be placed at the user's defined start point, so the Land Rover's weight was defined so that a user could transport and setup the rover, similar to the aerial drone. The weight includes all necessary parts for the land rover's functionality. This Rover is able to carry small loads if necessary.

3.2. Dimensions

3.2.1. Dimension of Aerial Drone

The sensors, battery, power distribution components, and microcontroller onboard the drone will fit within the footprint of the drone's chassis. The drone has a diagonal wheelbase of 66.04cm with a vertical footprint of the drone is 38.1cm.

3.2.2. Dimension of Land Rover

The Land Rover's chassis will have motors, wheels, a motor controller, and a microcontroller mounted to it. The Land Rover can have a volume ranging from 0.03m^3 to 0.07m3.

3.3. System Setup Locations

The following section will define the requirements for where each system will need to be placed in order for the Aerial Pathfinding Reconnaissance System to function properly.

3.3.1. Placement of Aerial Drone

The aerial drone must be placed on a level surface with no obstructing objects within 0.5m. of the zone. The drone must be placed within 30m of the user's local machine running the system's application. This placement will be used for takeoff and landing of the aerial drone, as well as transmission of data to the local machine.

3.3.2. Placement of Land Rover

The Land Rover must be placed within 30m of the user's local machine. The Land Rover should not be placed on a surface with a slope greater than 30 degrees or 58% grade. This placement will be used as the drone's start point.

3.3.3. Placement of the Local Machine

The user’s local machine must be in proximity to the aerial and Land Rovers’ initial locations such that the machine can connect to wireless local area networks hosted by each machine.

4. Electrical Interface

4.1. Primary Input Power

4.1.1. Aerial Drone

The standard power voltage provided to quadcopters and drones is 22.2V. The drone will be using a single 22.2V lithium ion battery with a capacity of 5000mAh. A lithium ion battery was chosen due to its weight and efficiency in comparison to other battery types such as lead acid. A single one of these batteries will power the aerial drone in normal operation and for longer flights it may take two of these batteries in parallel. Below, in Figure 1, is a graph detailing the voltage drop off of a lithium battery versus a Lead-Acid.

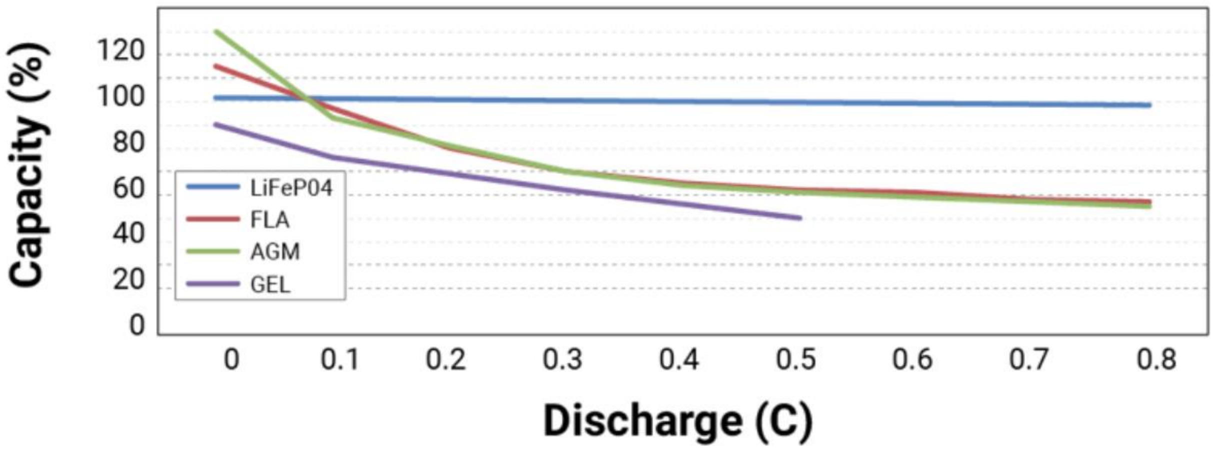


Figure 1. Lithium-Ion vs Common Lead Acid

The batteries will be attached to the drone with a custom fitting bracket that will securely hold the batteries during flight. The batteries must be fitted correctly to ensure no electrical malfunction occurs in the system.

4.1.2. Land Rover

The Land Rover will be using a high capacity lithium ion battery. To see why a lithium ion battery was chosen, see the above description on the benefits of lithium ion vs other battery types. Because of the varying power inputs on the digital controllers attached to this drone a series of boost and buck converters will be used in addition to the 4.2V battery. The batteries will be attached to the drone using a custom-fit bracket that will securely hold the batteries during land travel.

4.2. Signal Interfaces

4.2.1. Aerial Drone Connection

The onboard firmware for the Tarot 650 V2.1 utilizes serial ports on the PCB to communicate with the drone controller. The drone controller has high level methods to communicate with the telemetry module and flight controller. The microprocessor will issue commands to either the TTL UART port or the USB port. Several broadcasts can be received from the drone. Due to the large amount of data generated by the drone this application can record the data to any controller at the user's need. The drone can offload tasks using the OSDK to the controller to perform our autonomous flight and data retrieval tasks.

4.2.2. Aerial Drone µC

In accordance with the requirements of the Tarot, a handful of controllers can be utilized. An STM32, Arduino, or Raspberry Pi can send the required information back and forth from the drone to the server. While the Raspberry Pi does not have the fastest processor, it does have more functionality and less power draw than the other controllers. Because of the low frequency of the broadcasts coming from the drone, either controller will fulfill the project goals. The microcontroller will run the Linux operating system to communicate with the drone. The drone can also be directed with a Mission Controller Program without any object avoidance application.

4.2.3. Land Rover Connection

This drone will be using a Raspberry Pi to handle control of the motor controller and other system connections. The Land Rover will utilize a Sabretooth 2x25 motor controller to handle power distribution and control of the motors. The Land Rover will have a wireless connection to the user's local machine using a WIFI module further outlined in Section 6.

4.3. Sensors

4.3.1. Aerial Drone Sensors

4.3.1.1. Camera

A camera will be used by the µC to record video of the ground. This camera will be directed straight down and will not have any connection to the flight controller on the drone. Only to the 3rd party controller (STM32/Arduino/Pi).

4.3.1.2. Ultrasonic / Lidar

Attached to the front of the drone will be a series of ultrasonic sensors to handle object detection and perform anti-collision movements. Because the drone will only be performing small movements in accordance with telemetry data gathered it will only travel forward and rotate. Therefore, we do not need 360 vision, instead the drone will see the immediate area in front and maneuver around obstacles slowly.

4.3.1.3. GPS

The flight controller will collect telemetry data such as GPS. The GPS is connected via a telemetry connection on the flight controller. The highly accurate GPS location data is what is used to send the drone to specific locations. The µC will send coordinates to the flight controller to follow.

4.3.2. Land Rover Sensors

4.3.2.1. GPS

The µC will collect telemetry data such as GPS. The drone will use the GPS to determine its location in accordance with the map provided and relay this to the user.

4.4. User Control Interface

4.4.1. Physical Interface

Both the land and aerial drones will indicate to the user the effective battery power and ready state. Both systems will include an indication light for power-on and system-ready. The system-ready indicator will show the user that it is prepared for remote instruction and operation.

5. Software Interface

5.1. *Computer Vision and Mapping*

5.1.1. Inputs

The Computer Vision and Mapping subsystem will take one or more pairs of videos. For each section of the potential pathing area that can be fit into a single field-of-view of the camera, there are expected to be two clips which hold the following properties:

- The length of these videos should be long enough for a computer vision (CV) algorithm to have enough visual information to eliminate moving objects from detection.
    - For example, to ignore pedestrians, the clips' length should be at minimum the time it takes for a pedestrian to move a few feet from their original position.
- These clips should be taken sufficiently far apart (depending on the drone's altitude) to provide strong enough visual cues to infer depth with a second CV algorithm.
    - For example, a drone at 10m altitude will likely want to record their two clips at least 1m away from each other for improved depth estimation.
- Each clip's metadata must indicate the correct sequence of video inputs.

These requirements are necessary to infer depth over the entire survey area in a scheme as shown in Figure 2.

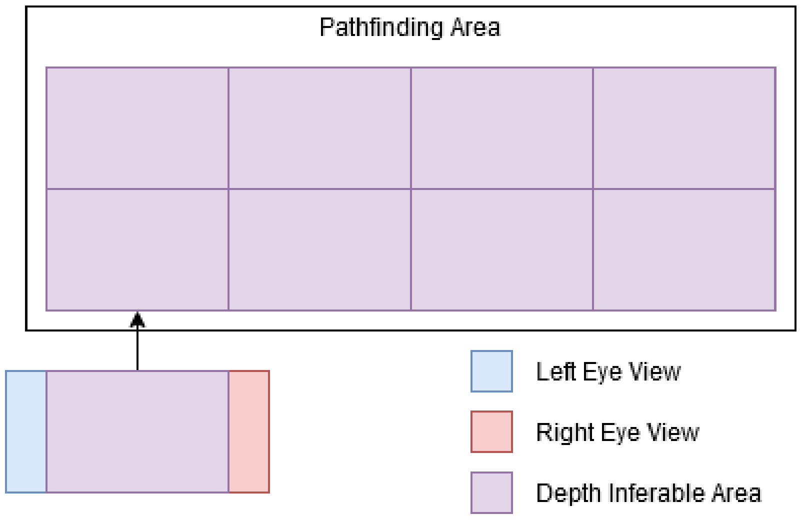


Figure 2. Depth Inferencing Over Area Segments

Furthermore, the drone's flight altitude and land vehicle's properties must be known to determine the scale at which obstacles are to be detected.

5.1.2. Error Conditions

The Computer Vision and Mapping subsystem will report an error through standard Python exceptions on any fatal error, or in any situation where a top-down map of the environment cannot be produced.

5.1.3. Outputs

The Computer Vision and Mapping subsystem will output a two-dimensional Boolean matrix representing the locations of obstacles, with each element representing approximately one pixel of the final top-down image of the environment. In this case, the term “obstacle” refers at least to any region which presents too great of a change in depth, for which a path through that region is not possible for the land vehicle.

5.2. Pathfinding and Ground Vehicle Detection

5.2.1. Inputs

The input of this system will be the two-dimensional Boolean matrix that represents the location of obstacles as outlined above and the land drone starting and ending position

5.2.2. Error Conditions

An error occurs if and only if no possible path is discovered from the Boolean map. The pathfinding and ground vehicle subsystem will report the error though standard Python exceptions and throw the exception to the user interface subsystem.

5.2.3. Outputs

There will be two outputs of this system: one pertaining to the map and the other pertaining to the drone. The Boolean map will be changed throughout a greedy algorithm. The greedy algorithm will work as follows:

- With respect to a final path heading from South to North, the greedy algorithm will choose the straightest path that is valid. Once a west path is deemed non-traversable, the drone will select an east path and repeat this process, next going south, then north.
- If anytime during this calculation a dead end is spotted, the algorithm will mark the current index as false and travel back to the previous position and start the algorithm again. Once the end index is located, the algorithm will be complete, and the path will be sent to the drone.
- If there is no valid path, the Boolean map will be marked as entirely false by the algorithm itself.

The goal of this greedy algorithm is to find the shortest locally optimal path in a quick manner.

The second output will be the path of the drone. Once this new Boolean map is sent to the drone, the drone will move to the end position following the map that has been sent if there is a possible path. If no possible path is found, the error condition is met, and the drone will respond accordingly.

6. Communications / Device Interface Protocols

6.1. Wireless Communications

A combination of communication through wireless local area networks (WLANs) and the internet will be used. Communication between the user's local machine and the drones will be through the use of a WLAN. If the user meets the requirement of internet access on their local machine and wishes to report their current location automatically, the internet will be used to fetch the location from Google Geolocation Services. All wireless communication will follow IEEE specification 802.11.

6.2. Host Device

A local machine, such as a laptop, provided by the user will serve as the host device for the system's desktop application. This device will allow for networking throughout the system.

6.3. REST Architecture Constraints

The system will implement a RESTful API so that software can be utilized within the systems operation. GET and POST application state transitions will be used to send and receive data throughout the system.

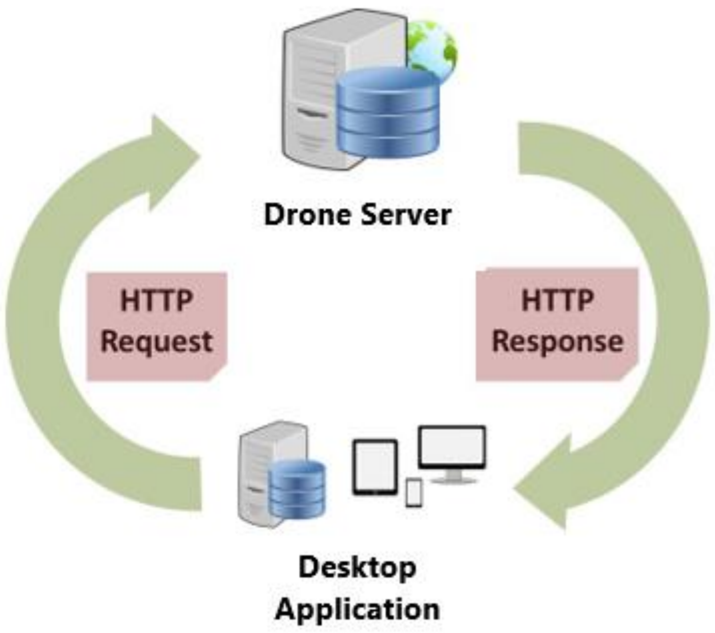


Figure 3. APRS REST Architecture

Execution Plan

Milestone	Assigned To	Week 1 24-Aug-20	Week 2 31-Aug-20	Week 3 7-Sep-20	Week 4 14-Sep-20	Week 5 21-Sep-20	Week 6 28-Sep-20	Week 7 5-Oct-20	Week 8 12-Oct-20	Week 9 19-Oct-20	Week 10 26-Oct-20	Week 11 2-Nov-20	Week 12 9-Nov-20	Week 13 16-Nov-20	Week 14 23-Nov-20
Understand Project	All	🟩													
Initial planning/research	All	🟩	🟩												
Draft Conops Report	All		🟩												
Source Lab Materials (Drone/Sensors)	All			🟩											
Draft FSR Report	All			🟩	🟩										
Draft ICD Report	All			🟩	🟩										
Draft Exection Plan	All				🟩										
Draft Validation Plan	All				🟩										
Prepare Midterm Presentation	All				🟩										
Design Server-Side Architecture	Jason					🟩									
Remove Moving Obstacles from Data	Mark					🟩									
Increase familiarity with Python	Max					🟩									
Implement Sever-Side	Jason					🟩	🟩								
Map Environment from Data	Mark						🟩								
Create algorithm for valid accessible path	Max						🟩								
Research/Order Parts	Dilanka/Jason					🟩	🟩	🟩							
Create error handling for inaccessible path	Max							🟩							
Implement Client-Side/UI	Jason						🟩	🟩							
Estimate Depth on Path Sections	Mark							🟩							
Research Drone SDK / stm32 / Sabretooth	Dilanka					🟩	🟩	🟩							
Add flight mapping functionality	Jason							🟩	🟩						
Research software interaction with drone	Max							🟩	🟩						
Design Aerial/Land Electrical Schematics	Dilanka							🟩	🟩						
Testing Electrical Off Drones	Dilanka								🟩	🟩					
Merge Depth Estimates with Map	Mark								🟩	🟩					
Build Air Drone	Dilanka									🟩	🟩				
Build Land Drone	Dilanka									🟩	🟩				
Prepare Status Update Presentation	All									🟩	🟩				
Implement Server on drone	Jason								🟩	🟩	🟩				
Test Flight of Air Drone	Dilanka										🟩				
Normalize Depth Field for Drone Altitude	Mark										🟩				
Resolution reduction of drone path	Max									🟩	🟩				
Test Travel of Land Drone	Dilanka										🟩	🟩			
Communicate with server on drone	Jason										🟩	🟩			
Derive Gradient Field From Elevation Map	Mark											🟩			
Program: Data Retreival of Air Drone	Dilanka										🟩	🟩			
Test Drone Simulation Programs	Dilanka								🟩	🟩	🟩	🟩	🟩		
Test Data Retreival From Drone	Dilanka												🟥		
Implement Obstacle Delineation	Mark												🟩		
Program drone path from map input	Max											🟩	🟥		
Package Software/Create Installer	Jason											🟩	🟩	🟩	
Prepare Final Presentaion	All												🟩	🟩	
Test and modify Air system	Dilanka													🟥	🟥
Program: Autonomous Flight of Air Drone	Dilanka												🟩	🟩	🟩
Draft Final Report	All													🟩	🟩
Prepare Final Demo	All													🟩	🟩

Legend	
🟩	Completed
🟥	Deadline
🟦	Planned Work Time
🟦	In Progress
🟥	Delayed

Validation Plan

Legend	Complete	Incomplete
	🟩	🟧

Subsystem	Test	Notes	Status
Data Collection and Drone Control	Aerial Drone flies from Point A to Point B	Drone follows waypoint from data file w/ interups	🟩
	Drone stops at midpoint locations on the map and records video and GPS location	Drone Stablize, autofocus, record & save	🟩
	Drone does not takeoff with failed connection/ prearm checks	Failure to lauch without udp / serial connection	🟩
	Low Battery Warning & Exit	Drone RTL on < 40%; Drone Lands on < 30%	🟩
	Add wp during flight and guided travel		🟩
	Drone stops at user shutoff switch	Controller has switch that changes from AUTO to manual	🟩
	Land Drone Functional with Directions	Land drone is built and programmed	🟩
	Drone detects obstacles & Avoides	Dectection 0.05-3m, not operational	🟧
	Electrical Systems Connected	All systems connected and powered	🟩
	System Built & Power On	Flight controller powering systems	🟩
Computer Vision and Mapping	The system produces depth maps with fewer than 1% of negative data points	Invalid points do not appear after the filtering stage. 0%	🟩
	A change in elevation of <2% of drone altitude results in >10 disparity	Measured manually; can not be detected. 1.26%->12	🟩
	The system can identify changes in elevation which account for <0.1% of the image area	Measured manually; object was 1.6% altitude with 15 disparity. 0.003%	🟩
	Depth estimation occurs at a speed of >1MP/s for each image in a pair	Calculated automatically. 1.6 MP/s	🟩
	Depth maps are merged to an accuracy of <0.5% of output map width	Measured manually; average over 14 trials. 0.26%	🟩
	Software components throw an exception when encountering fatal errors	All exceptions in library code trigger correctly. 3/3	🟩
Pathfinding and Drone Detection	Greedy Algorithm given a feasible path	Given a feasible path, if there is a valid path, it will be found 100% of the time	🟩
	Edge cases such as Incorrect Start/End position causes program to end and output no map and error statements	If the start/end cases are not accurate, the program will not run and output a map of 0s	🟩
	Handling Cyclic paths	If a cyclic path is found, the program will solve the cyclic path, retracing its steps once the same point is found twice	🟩
	Changing Resolution of given map given pixel width	Works with 0 even division. 16 / 4, 8/4, 100/50 etc...	🟧
	Updating Algorithm to handle weighted paths		🟧
Device Networking and UI	Fetch user's current location via geolocation api call (internet required) in under 5s	avg time to fetch = 145 ms size of data = 442 B	🟩
	Fetch map images in under 5s (offline assets served by back-end)	avg time to fetch = 253ms avg size of data = 76.3 kB	🟩
	Error handling and message provided in GUI to prevent crash and inform user	Error messages will be presented if the user attempts invalid operation	🟩
	Create drone navigation instructions accurate within 1 Meter	Accurate within .01 meter	🟩
	Communicate with drone's server in under 10s	avg time to post data = 2.91 s avg size of data = 388.38 kB	🟩
	Install software via encapsulated windows installer	create a single application containing all functionality and dependencies	🟩

Aerial Pathfinding Reconnaissance

Jason Gilman
Max Griffith
Mark Johnson
Dilanka Weerasinghe

SUBSYSTEM REPORTS

REVISION 1 – Draft
17 November 2020

SUBSYSTEM REPORTS
FOR
Aerial Pathfinding Reconnaissance

PREPARED BY:

Jason Gilman	Date
Max Griffith	Date
Mark Johnson	Date
Dilanka Weerasinghe	Date

APPROVED BY:

Jason Gilman	Date
Prof. S. Kalafatis	Date
T/A	Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	11/17/2020	Jason Gilman		Draft Release

Table of Contents

List of Figures

List of Tables

1. Introduction

The Aerial Pathfinding Reconnaissance System (APRS) is designed to navigate a blind ground vehicle using only data collected by a prior drone pass. First, the user will select their survey area and start/end points. Then, the drone will autonomously travel to and survey the area. It surveys the area by taking a series of downward videos of the area. Once the drone returns, the data will be collected by the drone for analysis. This analysis begins by converting the videos into images, removing all non-stationary obstacles as per our sponsor's requirement. These videos are then used to infer elevation data through stereoscopy, which is then analyzed further to produce an obstacle map based on the ground vehicle's capabilities. Once analysis is complete, the generated instructions can be relayed to the ground vehicle and it can begin its traversal of the path from its initial starting location.

The APRS is divided into four subsystems:
- Drone Control & Data Collection
- Computer Vision & Mapping
- Pathfinding & Drone Detection
- Device Networking & UI

Each of these four subsystems has been independently developed by one of our four team members since the beginning of Fall 2020, and all subsystems have reached the final stages of their development. Since each of the subsystems is either complete or mostly complete, the development of the APRS is on track for completion by the end of Spring 2021. The following sections report on the status of each individual subsystem and have been authored by their sole developers.

For reference, all source code is available on the team GitHub repository at:
https://github.tamu.edu/jasongilman/Aerial-Pathfinding-Reconnaissance-System

2. Definitions

API – Application Programming Interface
APRS – Aerial Pathfinding Reconnaissance System
FOV – Field of View
UI – User Interface
RaspPi – Raspberry Pi

3. Data Collection and Drone Control

3.1. Subsystem Introduction

The purpose of the Data Collection and drone control subsystem is to interface the physical components of the APRS system to the UI and other subsystems. Because the purpose of the APRS system is to navigate and survey an area, it is critical to confirm that the physical components of the system are functional. For this subsystem there are three major components. An aerial drone that will fly and survey an area, a land rover that will navigate the area surveyed by the aerial drone and a senor system that will collect all the data that will be sent to the computer vision system.

3.2. Aerial Drone

3.2.1. Description

The purpose of the aerial drone is to survey an area. Given this requirement the drone needs to move from location to location and gather data from each point. To accomplish this task the APRS system is using a Tarot 650 v2 drone. This drone is capable of carrying large loads and has a long flight time. Two elements that are vital to making sure the drone can navigate through a large area. The Tarot is using an mRo AUAV x2.1 flight controller with the ArduPilot firmware. Connected to the drone is a Raspberry Pi running Raspbian. The flight controller acts as a median between the microcontroller which controls the logic of the flight path and the sensors and motors that move the drone.

3.2.2. Communication

Because the mRo is a flight controller that utilizes the ArduPilot firmware it can be simulated to show the exact commands that the flight controller will follow. To send packeted commands to the flight controller the ArduPilot family of controllers uses the MAVproxy and MAVlink protocols to decode and encode serial commands. These commands can be sent literally through the serial port on the Pi or can be decoded by a third-party software such as Drone Kit. In order to have the drone follow the required commands the APRS uses a mix of both methods.

In case of aerial emergency, the drone control can be overridden via a handheld controller. The handled radio controller connected to the drone is a Tannaris X9 lite. This controller has the necessary functionality to change the flight controller from its auto mode where it will follow specified mission commands to the manual mode where all high-level movement commands are completed with the Tannaris.

3.2.3. Input Data

From the UI subsystem the Drone will read in coordinates to follow the UI subsystem provides an altitude above ground and an orientation to follow the program that controls the Drone will read in the attitude and orientation (yaw) information to initialize the flight of the drone.

The Pi will read in the initialization information and begin the read in latitude and longitude point that the program will turn into mission commands. These commands are then encoded into MAVproxy messages that are sent to the flight controller that the drone will follow.

3.2.4. Physical Characteristics

In order to be operational, the drone must have a body, four quad motors, a flight controller, a battery, and a controller. In order to be functional for the APRS the drone has the additional requirements of a Pi, a GPS and other sensors outlined in (3.5). All these systems have been attached to the drone body and are functional separately.

3.2.5. Aerial Obstacle Avoidance

In order to perform obstacle avoidance stunts while in the air the drone uses a series of ultrasonic sensors mounted to the drone (3.4.2). While the drone is following mission, points outlined in the program the drone will always fly straight forward. The groundspeed (speed of the drone over the ground) is determined by the command sent. The ultrasonic sensor located in the front of the drone will detect an obstacle. If an obstacle exists within 3 meters the ultrasonic sensor will place the drone in a state of obstacle avoidance.

In this state of obstacle avoidance, the drone will orient itself straight up and then move in the cardinal directions given the input from the ultrasonic sensors. It will move half a meter at a time to remain far from the obstacles. Once all obstacles are out of the way it will return to the autonomous mode where it will follow the mission points.

3.2.6. Validations

To validate the programming of the drone several simulations were used. For any flight controller in the ArduPilot family a software in the loop program can be calibrated to act as both the flight controller and the drone. All messages sent to the drone can display their operating capacity through the simulator.

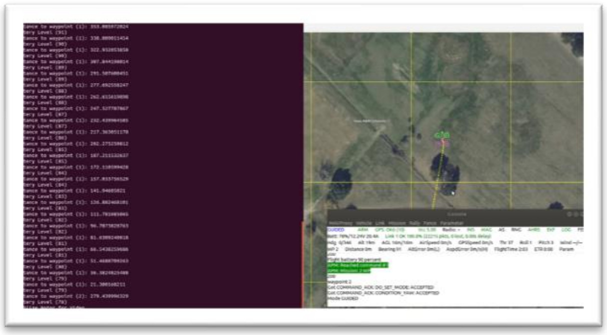


Figure 1. Orientation and Video Capture

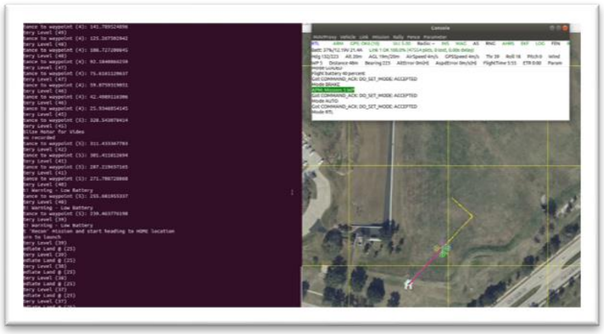


Figure 2. Return Home / Battery

This simulator was used to test the following sub-validations of the Drone subsystem.

- Coordinates Received – From the UI (6.4).
- Flight activated – Drone arms itself and begins liftoff.
- Mission points Followed – Drone will follow all mission point even after interrupted and resumed.
- Battery warning and landing – Drone will return to launch site at a given battery level and in case of dire emergency will land straight down immediately to prevent catastrophic failure.
- Vehicle Mode changing – To account for manual override, obstacle avoidance and ground video.
- Camera Wait time and operation – The Drone will change its orientation to match north perform a BRAKE and wait required amount of time for video.
- MAVlink messages decoded – To send serial packets to flight controller.

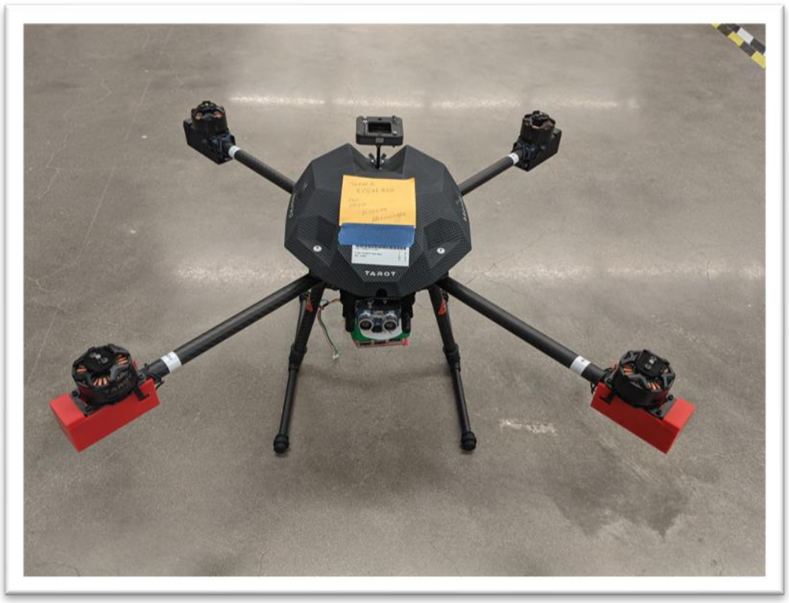


Figure 3. Physical Drone

Validations performed on the physical state of the drone.

- Lipo charges the motors and flight controller.
- Flight controller powers the Pi.
- Pi powers logic and data collection sensors.
- GPS receives power with GPS port on flight controller.
- Accelerometer and Gyroscope show correct orientation on flight controller.
- Tannaris x9 lite powers on and bus connection to RC receiver.

3.2.7. Calculations

Vlipo	:	23,46	V
Vout-pix	:	5,11	V
Vout-tel	:	5,11	V
Vout-pix	:	4,78	V
Vin-pi	:	5,1	V
Imax-pi-draw	:	1,28	A
Weight	:	2828	g

Table 1. Physical Drone Characteristics

3.3. Land Rover

The Land Rover subsystem is the final section of the APRS design. Referenced previously in the Introduction (3.1) the Land Rover will navigate the given area using simple directions from the Path Finding and Drone Detection subsystem (5.0). In this subsystem the goal was to provide a simple interface between drone commands that Path finding algorithm can follow. The land rover/drone will use a Raspberry Pi to communicate with the Communication subsystem (6.2.2). Here it will receive a python script to follow specified commands. The main driver for the rover is a Sabretooth 2x25 motor controller that outputs to all 4 DC motors on the rover. This Sabretooth controller can take several different types of command inputs to move the rover. The two main inputs used in the APRS are RC control and serial command.

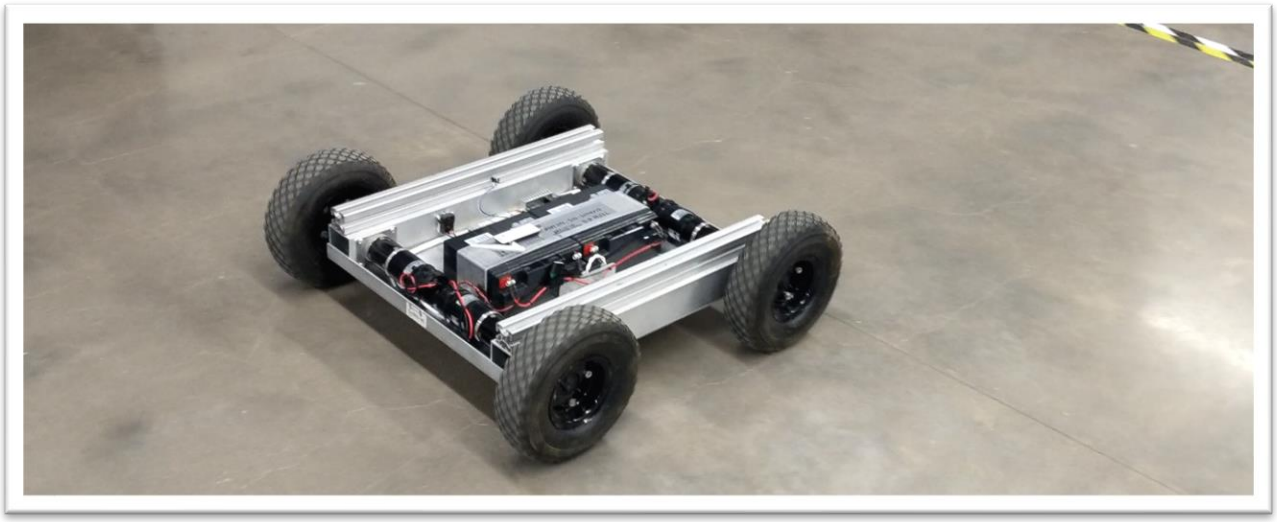


Figure 4. Rover Build

3.3.1. Radio Transmission

One aspect of controlling the land drone is to get it to its starting location. Because the land Rover weight 31 kg it is not easily movable by hand. To combat this the Drone will instead use an RC controller to get to its initial start location. The radio transmission is powered by the sabretooth controllers 5V output and received very simple gimble commands from a controller. In radio mode the rover is easily moveable from location to location and can even climb in rigorous climates.

3.3.2. Microcontroller Transmission

The rover is capable of moving using serial commands from a microcontroller. The Raspberry Pi sends serial packets to the sabretooth controller which interprets speed data and emergency stops. For the Pathfinding subsystem this subsystem uses a simple command to send the rover to specified locations. Because of the restrictions placed on sensor data applicable to the land rover it is not capable of detecting obstacles on its own or knowing its location. Therefore, the system has to be calibrated for the environment that it is navigating. Without the restriction of no sensors the rover could be used with a Pixhawk or other telemetry module with an accelerometer and gyroscope to perform internal navigation. The accuracy of the simple movements only lowers as the length of the course increases.

3.3.3. Validations

To validate the movement of the rover physical components were tested for power and efficiency. All components were able to be powered correctly and received voltages within the given range of valid power draw for the components. The rover is able to move linearly in any path with simple commands for turning, stopping, and traversing a certain distance. Like mentioned in 3.3.2, the system needs to be calibrated for its area and therefore its accuracy lowers as the length of the course increases. While this accuracy has not been tested on a large scale the rover performs within 3% of the required distance traveled while in the microcontroller transmission mode. This was tested by measuring the required distance and the time the rover takes to move that distance. The simple class for simple movement is complete and the rover easily moves in a grid like pattern.

- Rover moves in a grid like pattern
- Simple class created for low- and high-level movement

3.3.4. Calculations

Vout-rover	:	5,016	V
Vlipo1	:	12,45	V
Vlipo2	:	12,26	V
VlipoT	:	24,72	V
Vin-pi	:	5,016	V
Vout-pi	:	4,899	V
Imax-pi	:	1,113	A
Weight	:	31,8	kg

Table 2. Rover Characteristics

3.4. Sensor System

The sensor subsystem handles the data collection and obstacle avoidance on the APRS. Because the Land rover lacks the sensors required to navigate on its own it needs a way of finding out what obstacles exist in its way. To manage this a sensor system is placed on the aerial drone that will gather the data necessary to perform navigation on for the land rover.

3.4.1. Camera

The camera system uses the raspberry pi camera module to record video. In order to gather the data necessary, the computer vision subsystem (4.0) requires a left and right eyesight of the ground. Therefore, the camera, when observing the ground from its mounting location on the drone, will take two videos from slightly different locations in the air in order to grab a left and right eye video to create a depth map. The reason the drone grabs video instead of images of the ground is to remove any moving obstacles from the ground. As part of the design constraints the APRS will remove any moving obstacles from the area. To do this a video is required. The camera will take a full resolution 1080p image at 15 frames per second. The camera is capable of taking all kinds of different image sizes. For calibration purposes this can change given additional requirements from the Computer Vision subsystem. One important note is that the FOV of the camera is deterministic of the altitude and distance that the drone needs to travel while in the air. Using a simple geometric equation, the altitude of the drone can be determined by the number of steps required to take the drone. If the drone is flying higher in the air it will need to take less steps to collect the information in a large area. The RaspPi camera module has a floored FOV of 56 degrees. This makes it difficult to survey a large area given the amount of flight time necessary to record stationary video. Although this is a give and take scenario with the computer vision aspect of the APRS a balance can be found once more testing is put into determining the optimal altitude of the drone.

The camera stores its videos given the timestamp and the GPS location. This is subject to change as the Computer vision subsystem makes additional changes to the how it needs to interpret the video data. The RaspPi records in the .h264 format. This is similar to an .mp4 without the addition of .mp3 audio. While this subsystem does not make any changes to the format of the video this is capable in the future. The video is hardware encoded on the RaspPi.

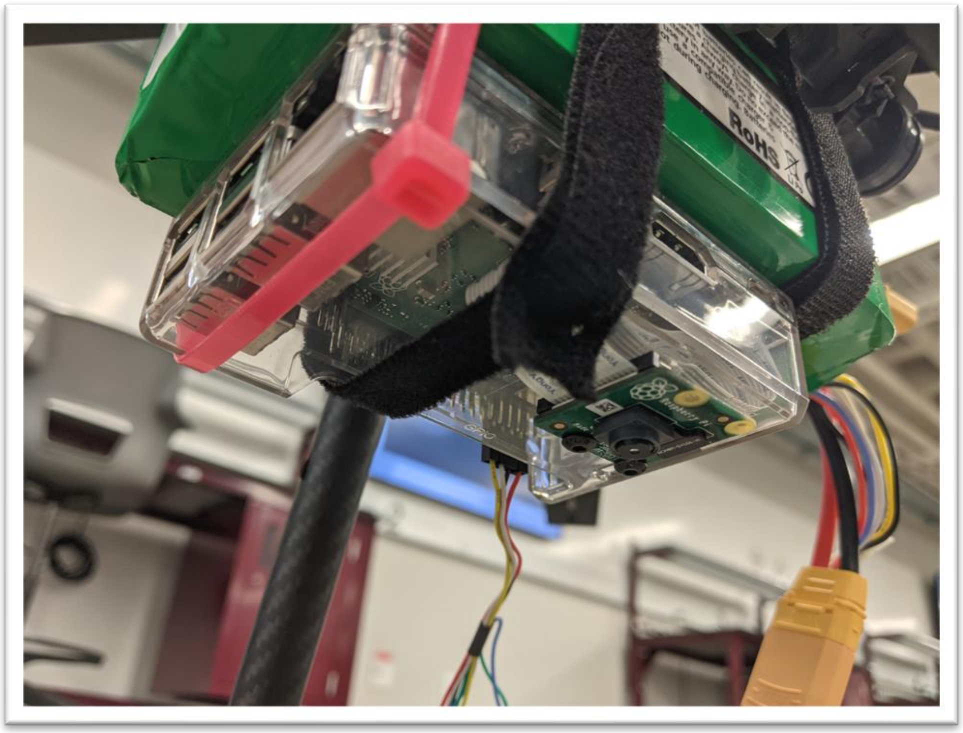


Figure 5. Drone Camera Mount

3.4.2. Ultrasonic Sensors

Like mentioned in Aerial obstacle avoidance (3.2.5), in order to perform obstacle avoidance stunts while in the air the drone uses a series of ultrasonic sensors mounted to the drone. While the drone is following mission, points outlined in the program the drone will always fly straight forward. The groundspeed (speed of the drone over the ground) is determined by the command sent. The ultrasonic sensor located in the front of the drone will detect an obstacle. If an obstacle exists within 2 meters the ultrasonic sensor will place the drone in a state of obstacle avoidance.

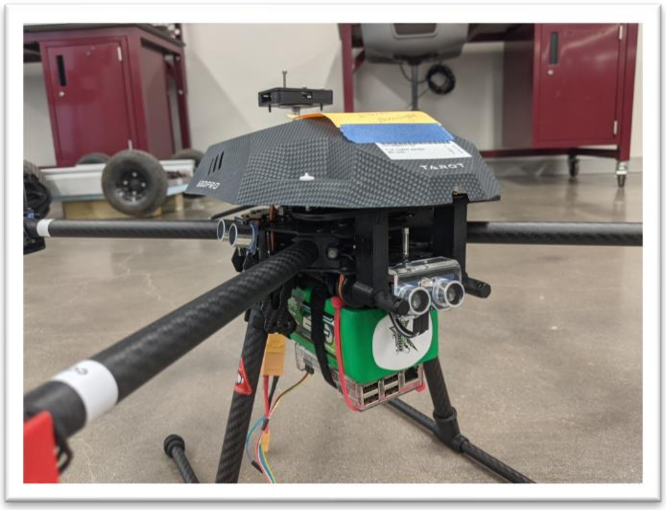


Figure 6. Ultrasonic Mount

The ultrasonic sensors are mounted to the drone in all 4 cardinal directions. They are capable of detecting an obstacle within 0.02m to 4m this is within the range calculated in validations. A distance of 2 meters was chosen as the cutoff distance because in testing it was shown to be reasonably accurate in terms of how the sensors performed.

3.4.3. Validations

To validate the sensors for the APRS several tests were performed to check their accuracy and specifications. All sensors came working and perfboard designs were created to mount and connect the sensors to the microcontroller on the drone.
The validations for the camera connected to the Raspberry Pi are as follows.

- Video is encoded in the .h264 format
- Video is stabilized and calibrated for the FOV of the camera.
- System is powered through the RaspPi and the Pixhawk.
- Labeled with the printout for coordinates and timestamp.

The validations for the ultrasonic sensors are as follows.

- Sensors detect with mild accuracy < 3m.
- Sensors are mounted in the 4 cardinal directions of the drone.
- Sensors are all powered by the RaspPi.

3.4.4. Calculations

V-ultrasonic	:	5	V
V-transmit:		3,21	V
V-echo	:	2,97	V
I-ultrasonic-max	:	0,06	A

Table 3. Sensor Characteristics

3.5. Subsystem Conclusion

While not all of the validations have been truly proven due to the lack of equipment, significant progress has been made on the functionality of the major smaller subsystems of the Drone control and Data collection system. The drone performs its target commands perfectly in simulations and the physical and simulated drone components have been calibrated to match each other completely. The Land Rover is fully established and ready to move on to the final stages of pathfinding and networking. The Aerial drone has not had a flight yet but with all the preparation done in simulations it will not be difficult to test its functionality. Data collection using the sensors on the drone is finalized with small changes necessary to integrate with other subsystems required in the APRS. The APRS team now has access to all components of a fully functioning drone and is well on its way to being a complete system.

4. Computer Vision and Mapping

4.1. Subsystem Introduction

The Computer Vision and Mapping (CV&M) subsystem is responsible for using the collection of video clips collected by the aerial drone to produce an overhead obstacle map of the environment. As prescribed in the Interface Controls Document, this obstacle map is implemented as a 2-dimensional matrix of Boolean values.

The process of creating this obstacle map from the videos is organized as a pipeline of stages progressing from video input to obstacle map.

4.2. Moving Object Removal

4.2.1. Description

The moving object removal step solves the requirement that obstacles in motion not be considered in pathfinding. It takes an OpenCV video object and returns a 2D NumPy array representing the image.

4.2.2. Algorithm

The implementation of this step uses a simple median of frames algorithm. A specified number of frames, 5 by default, are sampled over an even distribution of the video, starting at the first frame and ending with the last frame. Once extracted from the video as 2D matrices, a median value is calculated for each pixel, forming a new image. With the process complete, the resulting matrix is then returned.

Because multiple algorithms can accomplish this task, some experimentation was necessary early on to decide which algorithm to use in the subsystem. The alternative was a weighted feedback algorithm applied over the entire length of the video. This approach was more complex, slower, and less effective at producing a clean output. The median of frames algorithm, on the other hand, is simple, fast, and much better at avoiding blurry areas where a moving object once was.

4.2.3. Validation

Because of the simple nature of this processing stage, speed and accuracy are not at issue. The remaining validation is with regards to error handling. The Functional System Requirements state that unrecoverable error conditions should be met with a standard exception, and all such failure cases are accounted for in this stage. Note that because the function does not load the videos itself, the burden of error handling video loading is pushed to the caller. However, there are a few potential failure conditions internal to this function:

- If the inputs are not of the correct type, an exception will be thrown implicitly, satisfying the requirement.

- If fewer than two frames are requested, the function will be unable to complete its task, and a value error will be raised indicating the issue.
- If a frame cannot be retrieved for any reason, it will be skipped. This error is recoverable, but if fewer than two frames could be fetched, a runtime error will be raised indicating the issue.

4.3. Image Stitching

4.3.1. Description

The image stitching stage is intended to produce a color image of the overhead view of the surveyed environment. It takes many images as input and produces a single image as output.

4.3.2. Algorithm

The algorithm in use is OpenCV's image stitching pipeline set to "scan" mode.

Unfortunately, this algorithm (even in the other mode) is not suitable for the purpose we had originally intended. Ideally, this step would provide an overhead map as if taken from far above, but because the stitcher appears to prioritize the information in later images over earlier ones, the perspective often does not appear to be from overhead for much of the output image. Instead, a large portion of the image appears to be taken above but from one side, with much of the sides of landmarks visible on the map.

This issue is shown in Figure 7, with regions of perspective issues marked below the map. About half of the image appears to have a rightward perspective, with the sides of objects visible. A small portion to the side exhibits the desired perspective, while the rightmost portion exhibits some undesirable perspective but only from reaching the end of the data set.

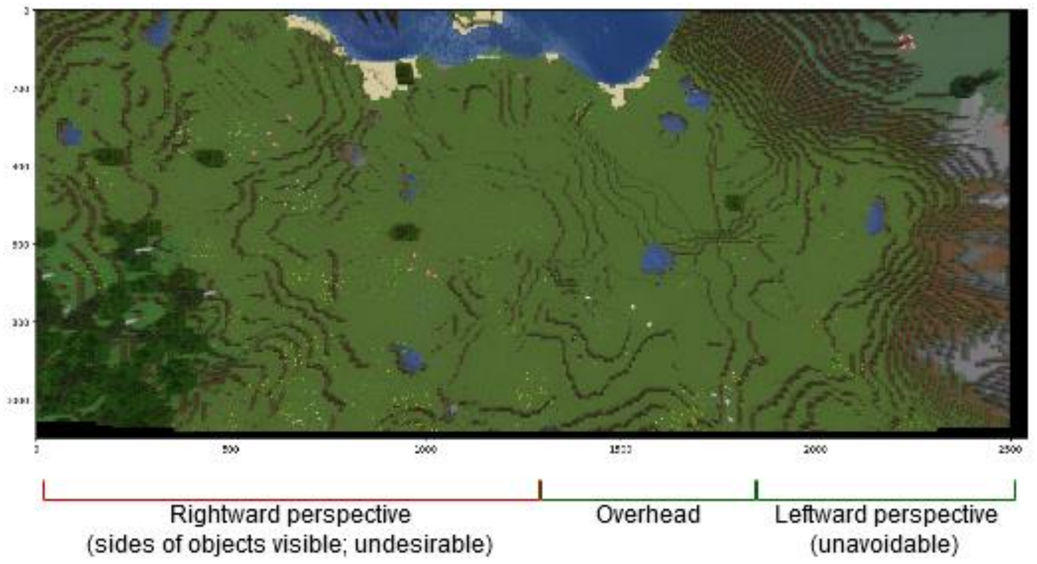


Figure 7. Image Stitching Perspective Problem

Because OpenCV's stitching pipeline does not handle perspective in the way we require, we have reevaluated our needs and discovered that this step is not truly necessary to run the system as a whole. Because the starting location of the land drone is expected to be in the first overhead view, a full color overhead map of the environment, while potentially useful for other configurations, is not vital to the system’s integration. As such, this stage of the subsystem may remain unused in the final product.

4.3.3. Validation

Because the performance and accuracy of this step is beyond our control, only error handling can be evaluated. If OpenCV's stitcher returns a failed status or throws an exception, an exception will be catchable by the caller, satisfying the requirement that unrecoverable errors throw an exception.

4.4. Depth Inferencing

4.4.1. Description

The depth inferencing stage embodies the core idea behind our method of detecting obstacles, as an "obstacle" status in most cases is caused by a difference in height. This step takes an image for the "left eye" and for the "right eye" positions and produces a two-dimensional matrix of depth values inferred through the stereoscopic perspective given by the image pair.

4.4.2. Algorithm

The core of the algorithm used is provided by OpenCV's “StereoSGBM” class, which itself implements a modified variant of Heiko Hirschmüller's semi-global matching algorithm. The results from this algorithm are then filtered using the WLS filter provided by OpenCV.

This algorithm has numerous parameters, which a user of the system will struggle to understand, and some of which have so great an impact on the quality of the output that they may need to be tuned by hand for each run of the system for best results. To simplify the calling interface, the parameters with the greatest effect have been determined through experimentation and only these must be provided by the caller. The remaining parameters are set to sensible defaults.

Unfortunately, because the best values for parameters with the most impact cannot be decided automatically, the user may instead have to tweak these by hand. To accommodate this requirement, a new user interface element will be included in the final product for this purpose. The user will be able to adjust sliders to change these values interactively and see their effects in real time. Once these values are decided, the rest of the subsystem will perform without issue.

4.4.3. Validation

The depth inferencing stage is subject to precision, sensitivity, and performance tests. For each test input, the parameters are tweaked to be as effective as possible to measure the capabilities of the algorithm and not the effectiveness of a single set of parameters. These measurements are performed as follows:

- To measure precision, test the system over an area with many assorted objects decreasing in size and preferably with many of each object and at different angles. Find the smallest obstacle that can be consistently detected and measure its bounding box in the image. Divide this measurement by the size of the original image to acquire the final measurement in percent image size. Post-process filtering may cause the system to ignore small objects, so this measurement is useful to ensure that this has not occurred to an unreasonable degree.
    - The system has been observed to reliably detect objects <0.003% the size of the source images, passing the desired 0.1% threshold.
- To measure sensitivity, run the system over any area with a region for which the difference in elevation is known. Measure the difference in disparity values between these two regions and divide by the ratio of their real elevation difference to the drone's altitude. This yields a value of disparity per altitude and is helpful for understanding the limits of stereoscopic depth inferencing.
    - The system’s sensitivity has been measured at approximately 952 disparity per altitude, passing well over the desired value of 500.
- Processing speed is measured automatically at each runtime, in megapixels of source images per second (MP/s). This metric is to understand the effect of each source image’s resolution on processing speed. For true processing speed, the measurement can be doubled to take into account that two source images are consumed at a time.
    - The system’s self-reported processing speed has not been observed to fall below 1.6 MP/s, passing well over the desired 1 MP/s.

4.5. Depth Map Merging

4.5.1. Description

The depth map merging stage is responsible for combining the information from the depth maps of each area segment into one cohesive whole.

4.5.2. Algorithm

For lack of an existing solution to this problem, a new algorithm was designed. The goal of this algorithm is to find a way of overlapping the two depth maps that introduces the least error. When these maps are overlapped poorly, an obvious "seam" can be observed where the two maps meet and their values differ. For simplicity, the word "seam" herein refers to the *difference* in pixel values between columns in two images.

To understand this algorithm, first consider the methods one might use to combine two images with a horizontal overlap in real life. Firstly, as with physical pictures, it is possible to simply lay one photo over the other. While simple, this has the disadvantage of merging the two images at the edge of one image, where the lens distortion and effects from perspective will be at their worst. Therefore, we want to "cut off" a portion of the overlapping space in the image to move the edge closer to the center, where distorting effects are lessened. There, the two images will agree spatially more easily, and the overlapped result should therefore be more accurate.

To find a good value for how much to cut off, the first section of the algorithm analyzes the seams that would be created by equally removing columns from the right and overlapping columns on the left. It chooses the seam with the least standard deviation along it, and then uses this value to decide how much to remove from the right image. The output from this hypothetical result would be unsuitable as a solution, however, because the change in perspective means that landmarks in one image will not be equally far from the edge in the other. The result is that this method of overlapping could never produce an overlap that lines up correctly. However, it will result in an amount of the overlapping map removed which should adequately avoid distortion and other local errors. This half of the process is roughly visualized in the left half of Figure 8, which shows the first stage of the merging algorithm.

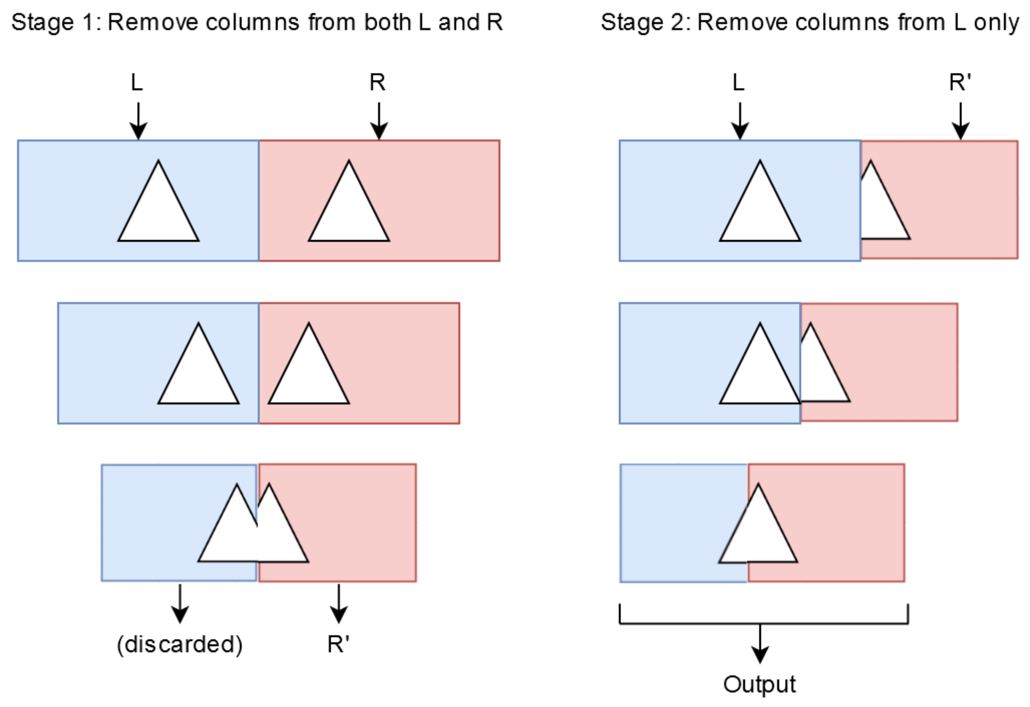

Stage 1: Remove columns from both L and R
Stage 2: Remove columns from L only
L
R
L
R'
(discarded)
R'
Output

Figure 8. Depth Map Merging Algorithm

The second section of the algorithm uses the number found in the first section to test another set of hypothetical seams. This time, the same amount is removed from the right map in each case, but a different amount of the left map is overlapped each time with the resulting seam evaluated. The seam with the least standard deviation is chosen, this time finalizing the overlap between the left map and "chopped" right map. This half of the process is visualized in the right half of Figure 8, which shows the second stage of the merging algorithm.

A few potential effects are also accounted for in the algorithm. In case the right map differs in value from the left by a consistent amount, the right map is corrected by the mean difference before overlapping. If the maps are not of the same height when merged, the algorithm pads the bottom of the shorter one with zeroes. Lastly, to avoid further issues caused by lens distortion, the seams that are analyzed ignore a fraction of each map from the top and bottom.

Finally, even though this algorithm was designed to merge two maps along a single axis, it can be used with a reduction to merge several maps, and with transpositions to merge maps vertically, with a few caveats. Rotation is not considered in this algorithm, and the method of padding the end of the shorter map with zeroes arises from the assumption that the tops of each map are already vertically aligned. For merging maps over a strictly rectangular area, no camera rotation is necessary, and the alignment assumption is correct. However, these limitations mean that more complex map schemes are not possible. For the purposes of the system as a whole, however, we do not consider this limitation to be restrictive.

4.5.3. Validation

The primary concern with this subsystem is accuracy, and for this reason it is tested with an accuracy measurement. The accuracy of a merge cannot be determined automatically because the ability to do so would imply having a perfect solution to the problem. Instead, accuracy must be measured manually. This process is performed by analyzing seam locations in the output and counting how many pixels away the result is from being "correct". Determining what merge would have been more correct is very challenging, however, so measuring the accuracy of this stage is difficult. In some cases, the original source images must be used to aid measurements, and in other cases there are no obvious discrepancies along a seam that can be used to measure its error.

Over 14 trials, the algorithm deviates by 5px on average for a source image width of 1920px, placing its error at 0.26%, which passes under the desired threshold of 0.5%.

4.6. Gradient Derivation

4.6.1. Description

The gradient derivation step produces a map of the environment where every pixel represents the "steepness" at that location.

4.6.2. Algorithm

The algorithm used to implement the gradient derivation step is known as a Sobel filter. The implementation used in this subsystem uses Euclidean distance instead of simpler approximations but does not suffer from any noticeable lack of performance as a result. Furthermore, because the edges of the depth map are meaningless regarding the slope of the environment, the single step along both axes is implemented by a roll operation for the sake of simplicity. This technically treats the depth map as a torus, in that each edge of the map is treated as wrapping around to the other side. However, because the edges of the map are meaningless either way due to the lack of data, this simplified approach does not present any serious consequences.

4.6.3. Validation

Due to the simple and mathematical nature of this subsystem, there are no addressable performance or accuracy metrics. Likewise, there are no real failure conditions other than type errors. In this instance, the Functional System Requirements dictate that an unrecoverable error such as this shall result in an exception, but this requirement is automatically satisfied by the Python interpreter, which automatically raises a type error exception in this circumstance.

For completeness, the source code for this stage is included in Appendix A.

4.7. Obstacle Delineation

4.7.1. Description

The obstacle delineation stage is the final stage of the subsystem, in which a depth map of the environment is used to construct a Boolean map of obstacle boundaries.

4.7.2. Algorithm

Currently, these values represent areas which are too steep for the land vehicle to traverse. This is implemented very simply as an implicitly element-wise comparison to a threshold value. The simplicity of this algorithm comes as a direct result of the many stages before it.

The logic behind the current implementation of this stage is that large objects which would prove to be an obstacle to the land vehicle are likely so due to their height, which can be detected using the stereoscopic overhead view of the environment produced by earlier stages. Because the output representation of the environment has been intended for use over land areas only, it therefore does not account for obstacles that may cause other issues depending on the land vehicle, such as bodies of water, which are non-trivial to detect.
To account for changing requirements in the future, it is possible to modify the simple obstacle delineation stage to account for other variables or obstacle detection criteria. These changes would be internal to the subsystem and would not require any change to its interface.

4.7.3. Validation

The extraordinary simplicity of the algorithm in use for this step means that no complex validation steps could be decided upon. Because this algorithm is implemented by the use of a single comparison operator (though abstracted nonetheless), there are no accuracy or performance issues that can be addressed. As with the gradient derivation step, a type error is the only failure condition, and is handled implicitly by the interpreter.

For completeness, the source code for this stage is included in Appendix A.

4.8. Subsystem Conclusion

All of the individual stages of the computer vision and mapping subsystem are complete. Despite a couple of minor hiccups in some of the less important stages, the subsystem's components pass all their respective validation tests and are demonstrably capable of performing their necessary tasks. Furthermore, the set of subcomponents comprising this subsystem has already been integrated successfully into the full pipeline, and has been shown to perform the full transformation successfully on many sets of test data, an example of which is shown in Figure 9.

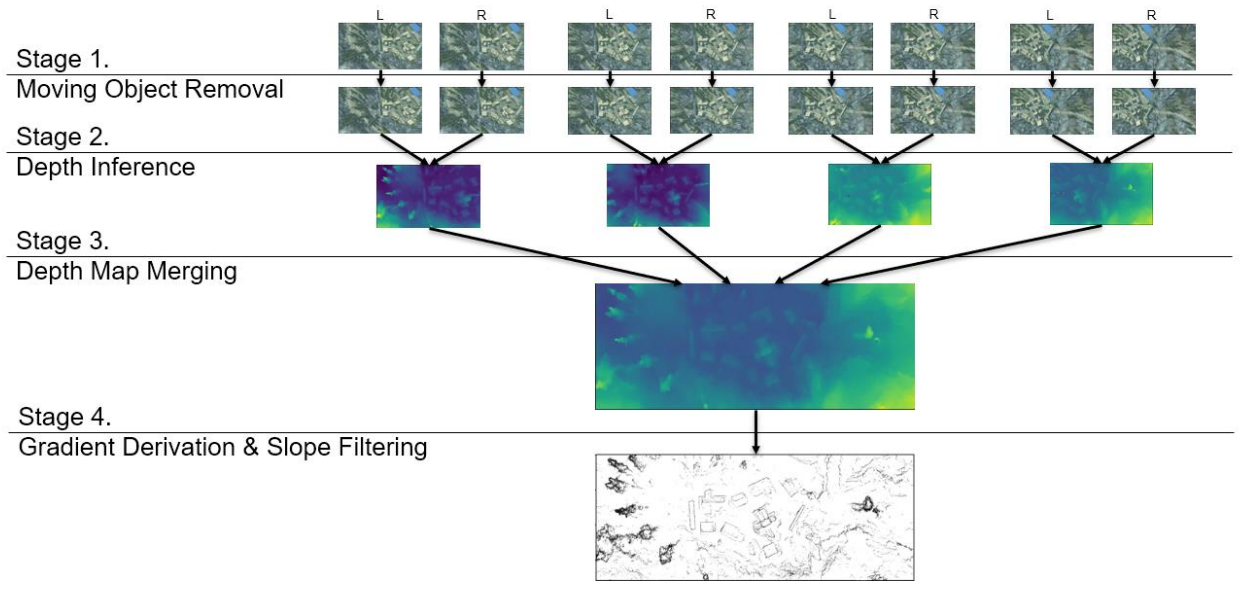


Figure 9. Full CV&M Pipeline Example

5. Pathfinding and Drone Detection

5.1. Subsystem Introduction

This is the Pathfinding and Drone Detection system design for the pathfinding of the APRS. This includes the resolution decreasing algorithm of the input map, and the pathfinding algorithm to output a proper map that will then be used to control the ground drone. For the resolution decreasing algorithm, I will be responsible for decreasing the resolution of the map to match the size of the land drone so that there will be no error in making sure that every spot traveled by the drone will be acceptable for the complete area of the drone. Currently, this program can minimize maps based on any range of pixels as long as the pixels are evenly divisible within the map's dimensions. Attached to this system, is the pathfinding system, which will input the map at its initial resolution and the dimensions of the drone for the resolution decreasing algorithm. Below is a small diagram to show how the two programs interact with each other.

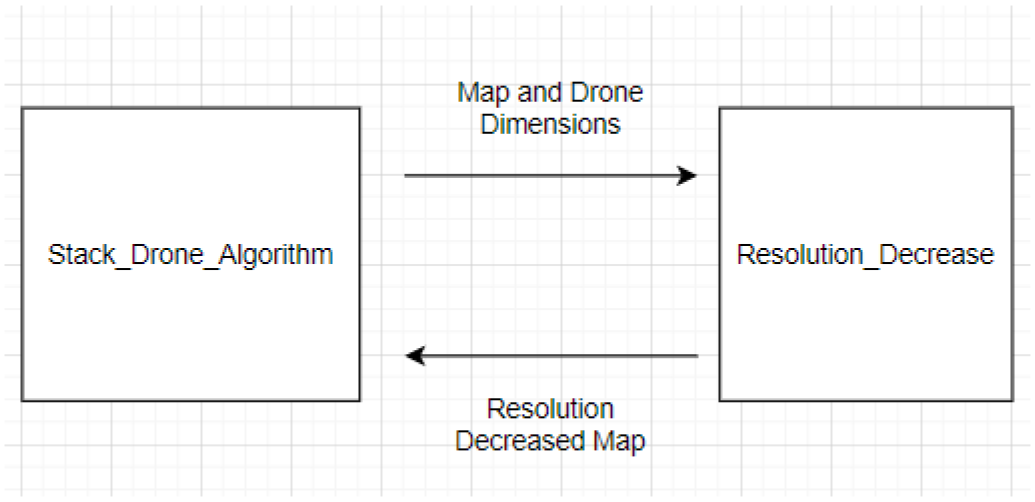
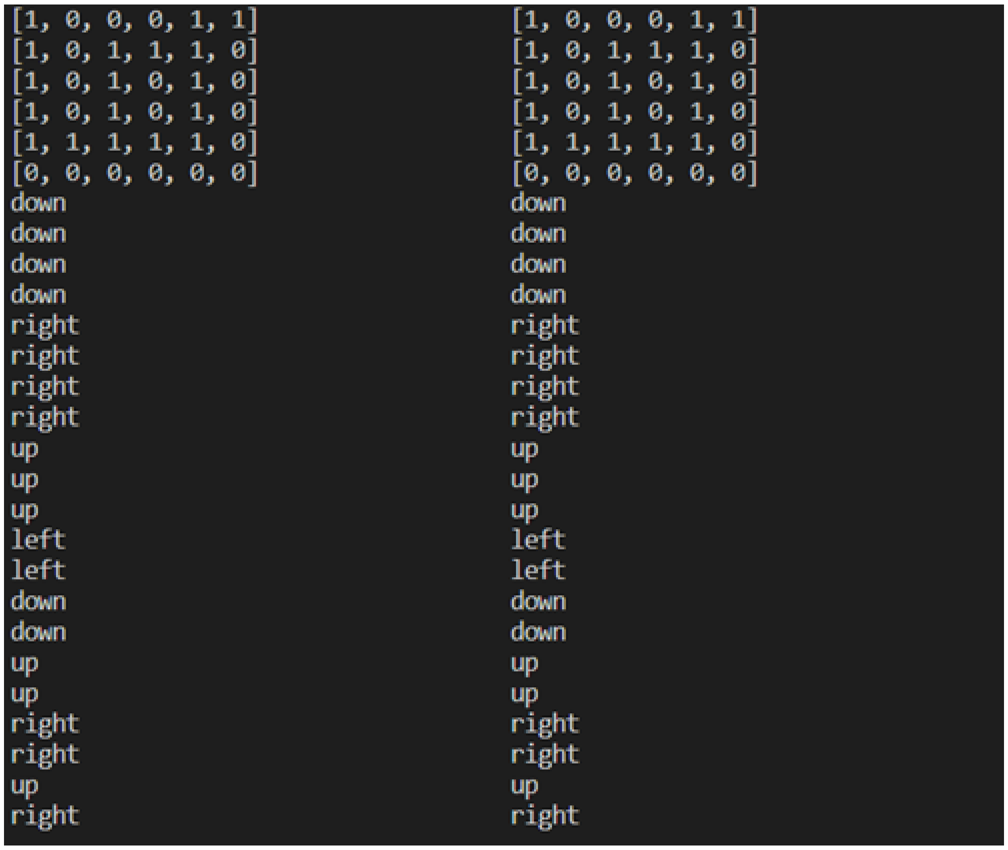


Figure 10. Software Interaction

5.2. Stack Drone Algorithm

5.2.1. Description

When deciding an algorithm to use for the drone, at first, I wanted to use recursion to recursively travel throughout a map so that any previous values can be looked over if there is a dead end. Once a dead end was found, we would then choose another path that was not taken. This idea seemed great at first, but I realized that when we had to backtrack, instead of keeping track of the data throughout recursion, it would better be kept within a stack so that it would be easier to match cases and keep storage of the data. Eventually, I settled on using a stack to keep track of the most recent paths taken, the most recent path being the coordinate on top. With the inclusion of the stack, it was easy to verify a correct course and to fix an edge case where the program would run into an infinite loop or never reach the ending location. Below is the dump of the trace of the path for a sample input as well as the final path map, and the path dumped by the stack.



```
[1, 0, 0, 0, 1, 1]                    [1, 0, 0, 0, 1, 1]
[1, 0, 1, 1, 1, 0]                    [1, 0, 1, 1, 1, 0]
[1, 0, 1, 0, 1, 0]                    [1, 0, 1, 0, 1, 0]
[1, 0, 1, 0, 1, 0]                    [1, 0, 1, 0, 1, 0]
[1, 1, 1, 1, 1, 0]                    [1, 1, 1, 1, 1, 0]
[0, 0, 0, 0, 0, 0]                    [0, 0, 0, 0, 0, 0]
down                                  down
down                                  down
down                                  down
down                                  down
right                                 right
right                                 right
right                                 right
right                                 right
up                                    up
up                                    up
up                                    up
left                                  left
left                                  left
down                                  down
down                                  down
up                                    up
up                                    up
right                                 right
right                                 right
up                                    up
right                                 right
```

Figure 11. Sample Map

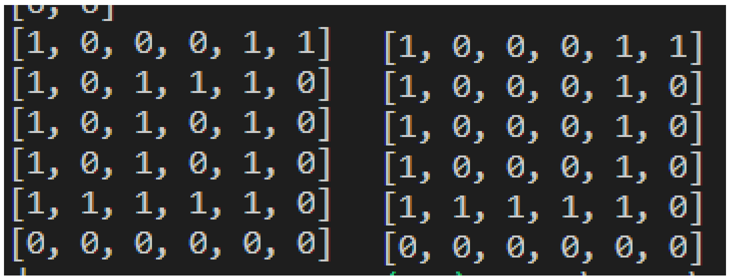
5.2.2. Algorithm

The algorithm works by having a starting position, ending position, input map, and the dimensions of the land drone. The dimensions for the land the drone and the input map are sent to the Resolution_Decrease algorithm, and a new map is handed back to this algorithm. Next, this algorithm will enter a function that will start finding the path from the starting position to the ending position. The algorithm first starts with checking if the next pixel (right) is a 1 and isn't already included in the path. If it passes these tests, it will call a function called stack_push that will add to the stack the new point and set as the current condition. This will happen for the next pixels (left, down, up) if the previous path is unavailable. In this function, after the value is added to the stack, the path finding algorithm will be called with the new point.

If all of the four options (right, left, down, up) aet exhausted, then the stack_pop function will be called. This function will pop the top value from the stack if there is a value and set the current location to the popped value, while setting the current location's value to a 0 instead of a 1 so it knows that the path is invalid. After a value is popped, this function will also call the path finding algorithm with the new point. However, if there are no valid paths, this means that there is no valid location from start to finish and the program will end and will output a map of all 0s.

5.2.3. Validation

To validate this algorithm, I first tested some basic inputs that gave a definitive answer, such as a straight line, and then traversing other directions, and then also finding dead ends. After these all worked, I tried to create a different map and ran into an infinite loop. Eventually I realized that the map I was using contained a cycle, or a path that never ends. This is where I introduced a function to check if the next point that we are going to be using was already in the stack specified earlier (Appendix B). If this function returned true, then I would treat it as a dead end and try a different path, while backtracking until there was another path found.



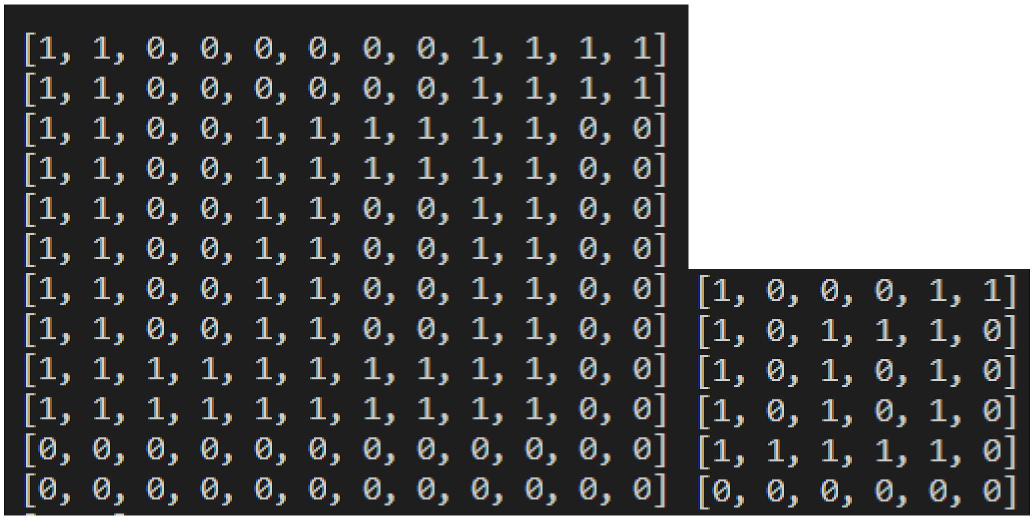
```
[1, 0, 0, 0, 1, 1]   [1, 0, 0, 0, 1, 1]
[1, 0, 1, 1, 1, 0]   [1, 0, 0, 0, 1, 0]
[1, 0, 1, 0, 1, 0]   [1, 0, 0, 0, 1, 0]
[1, 0, 1, 0, 1, 0]   [1, 0, 0, 0, 1, 0]
[1, 1, 1, 1, 1, 0]   [1, 1, 1, 1, 1, 0]
[0, 0, 0, 0, 0, 0]   [0, 0, 0, 0, 0, 0]
```

Figure 12. Cyclic Path Solved

5.3. *Resolution Decrease Algorithm*

5.3.1. Description

After the completion of the Stack Drone Algorithm, there needed to be a way for the drone to choose a proper path by 1 pixel. Since the input map pixels are going to be smaller than the dimensions of the drone, the input map would have to have pixels that would fit the size of the drone to ensure that every square of the drone would take an accurate path. Using a few while loops, it became possible to shrink a matrix down, with the shrunken matrix showing a 1 if all the spaces in the old map to be merged were 1s. Below is a figure of a resolution decrease of a matrix by the dimensions 2x2. As shown below, the spaces where there are 1s in a 2x2 grid starting at an even starting and ending location ((0,0), (0,2), (2,0), (2,2) for examples of the start positions) would be selected to have a 1 for the newly created pixel.



```
[1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
[1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
[1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0]
[1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0]
[1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0]
[1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0]
[1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0] [1, 0, 0, 0, 1, 1]
[1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0] [1, 0, 1, 1, 1, 0]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0] [1, 0, 1, 0, 1, 0]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0] [1, 0, 1, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] [1, 1, 1, 1, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] [0, 0, 0, 0, 0, 0]
```

Figure 13. Decreased Resolution Map

5.3.2. Algorithm

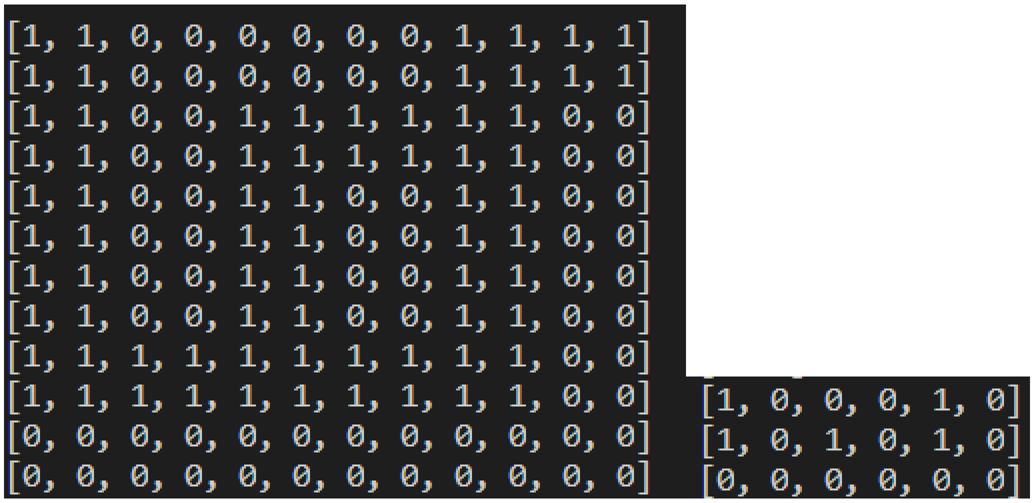
This algorithm first takes in an input of a map and the drone dimensions. Next, the algorithm creates a new matrix of the new size (current map dimensions divided by the pixels).

Next, 4 while loops are used in order to go through the two map dimensions and the two pixel dimensions. A variable is set as the value 1 (since we want to assume that the values are a 1 until we find only one 0) and then the dimensions of the pixel are searched within the map updating the location of the map to check every pixel for its value. If any one of the values is a 0, then the pixel is automatically set to a 0. This moves on throughout the entire matrix until it is filled.

After the matrix is filled, it is then returned to the Drone Stack Algorithm

5.3.3. Validation

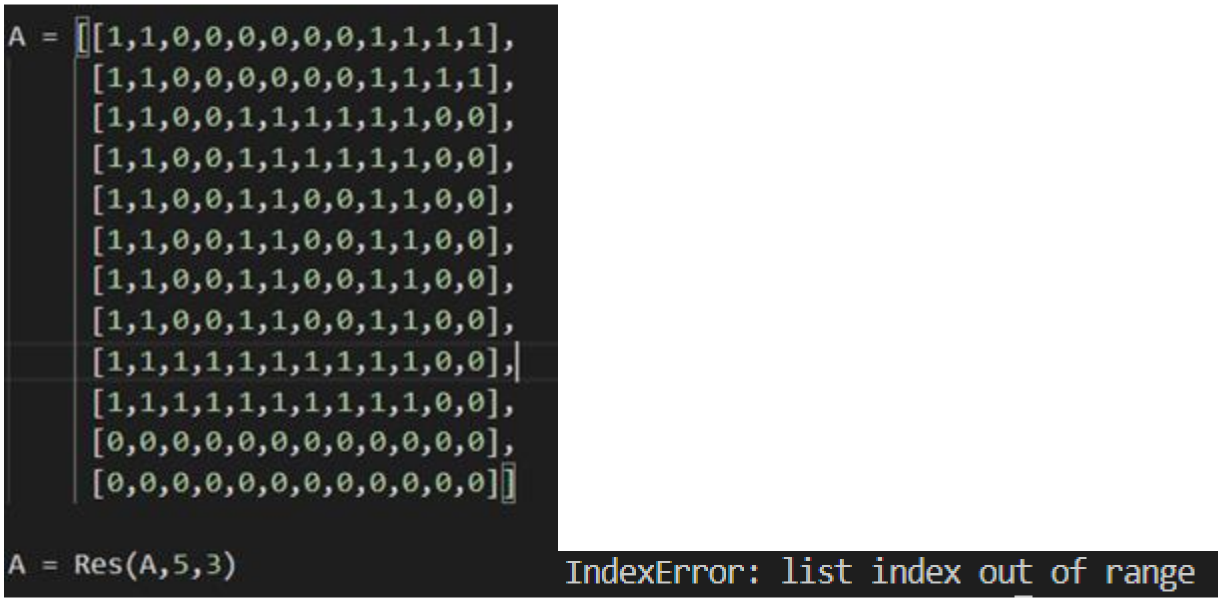
To validate this algorithm, I tested multiple sized resolution decreases among different sized maps. As shown above, the resolution decrease of a 2x2 matrix works as intended, and so does evenly divisible map dimensions to pixel dimensions as shown below in figure 6 (with a decrease of 4x2). However, anything that does not evenly divide will produce an error which has yet to be handled, and this is shown in figure 15. There are two ideas as of now: one is to take away the westmost and southmost portions of the map if there is runoff of the pixels, however this could cause some potential errors if the starting/final ending location is specified in those locations. The other idea is to try to move the drone depending on if the next pixel will fit the drone, where the next pixel will just have to match whichever dimension the drone will want to move (forward then width of drone. If downwards, then I will have to take into account the rotation of the drone and the width).



```
[1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
[1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
[1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0]
[1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0]
[1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0]
[1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0]
[1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0]
[1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
[1, 0, 0, 0, 1, 0]
[1, 0, 1, 0, 1, 0]
[0, 0, 0, 0, 0, 0]
```

Figure 14. 4x2 Decreased Map



```
A = [[1,1,0,0,0,0,0,0,1,1,1,1],
     [1,1,0,0,0,0,0,0,1,1,1,1],
     [1,1,0,0,1,1,1,1,1,1,0,0],
     [1,1,0,0,1,1,1,1,1,1,0,0],
     [1,1,0,0,1,1,0,0,1,1,0,0],
     [1,1,0,0,1,1,0,0,1,1,0,0],
     [1,1,0,0,1,1,0,0,1,1,0,0],
     [1,1,0,0,1,1,0,0,1,1,0,0],
     [1,1,1,1,1,1,1,1,1,1,0,0],
     [1,1,1,1,1,1,1,1,1,1,0,0],
     [0,0,0,0,0,0,0,0,0,0,0,0],
     [0,0,0,0,0,0,0,0,0,0,0,0]]

A = Res(A,5,3)
```

```
IndexError: list index out of range
```

Figure 15. Error With a 5x2 Decreased Map

5.4. *Subsystem Conclusion*

In these two subsystems, they have working parts and parts that can be improved on. In the stack algorithm, there are some efficiency issues which would result in many changes to the algorithm. For instance, in the algorithm, there are priorities in the directions, so if the ending location is in the direction of the least prioritized value, the drone may take a lengthy path and could potentially run out of battery. However, this algorithm does work and will always provide a path to an ending location granted that there is a valid path between the starting and ending positions.

For the resolution decreasing algorithm, the only issue is the sizing the pixels given a map. This can be solved by giving the drone a larger area so that there will not be a problem trying to ensure that the ending position does not get cut off by taking away pixels from the map. The downside of this is that there could be an optimal potential path that will not be taken because of this.

6. Device Networking and User Interface

6.1. Subsystem Introduction

The purpose of the device networking and user interface subsystem is to facilitate communication between all of the APRS devices, including the aerial drone, land rover drone, and the user's computer. All APRS device communication, inputs, outputs, and error reporting will occur through a user interface which will be delivered as a desktop application and run on the user's computer. Development of this subsystem included designing a network infrastructure, drone server software, and user interface.

6.2. Network Infrastructure

6.2.1. Description

Several times throughout the APRS's operation, the systems devices need to transmit data between each other. To facilitate this, a wireless local area network approach was used. The network was configured to standard infrastructure mode, where client devices connect to wireless access points over a Wi-Fi signal.

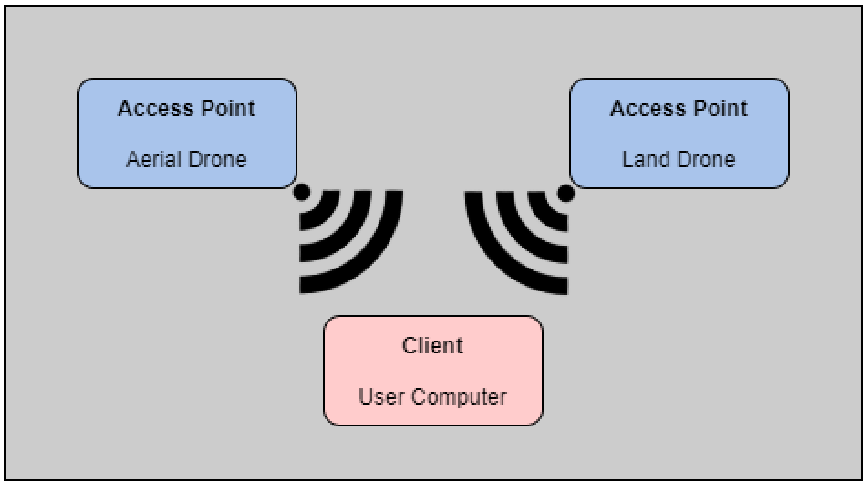


Figure 16. Overview of Network Infrastructure

For the APRS, both the aerial and land drones will act as access points in the network. This means that they will each broadcast a different Wi-Fi signal. This approach was used so that the user's computer, or client, which is running the desktop application, can connect to the correct device for the transmission of data.

6.2.2. Access Points

Onboard each drone is a Raspberry Pi 3B+. This device serves purposes defined in the Data Collection and Drone Control subsystem, but it will also be configured to allow the drone to act as an access point. To accomplish this, the TP-Link TL-WN722N wireless module was configured into access point mode using the RTL8188eus version 5.3.9 driver and various network configuration files.

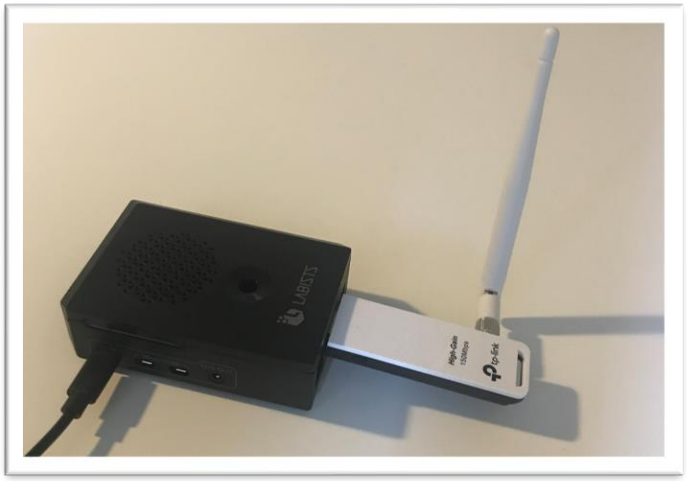


Figure 17. Raspberry Pi 3B+ with TL-WN722N

The TL-WN722N wireless module allows each drone to be connected to from up to 30 meters.

6.2.3. Client

The user's computer running the APRS desktop application will act as the client in the network. No configuration was needed to make the user's device act as a client, meaning that any device that meets the system requirements of the APRS desktop application can be used to operate the system.

Specification	Minimum Requirements
Processor	Intel Core I5 7th Generation or equivalent
RAM	8 GB DDR3
Operating System	Windows 10
Storage	5 GB

Table 4. Client System Requirements

6.3. Drone Servers

6.3.1. Description

While the APRS network infrastructure allows the system devices to connect to each other, the Raspberry Pi 3B+ devices still require software to receive, send, and interpret data onboard the drones. To allow for this, drone server software was developed.

6.3.2. Software

Using Python and the Flask web-framework, software was designed to allow the onboard Raspberry Pi device to act as a web-server, so any client devices connected to the microcontroller can send HTTP requests for data. The Flask framework defines API endpoints that receive HTTP requests from any connected client. For the purposes of the APRS, several endpoints were required on the aerial and land drone servers. Whenever one of the endpoints is requested by a client, a function within the drone server software is allowed to run.

Drone	Endpoint	Method	Function
Aerial + Land	"/uploadInstructions"	POST	Save navigation instruction file from client request in drone storage.
Aerial	"/download"	GET	Send collected video data from flight to requesting client.
Aerial + Land	"/launch"	GET	Invoke function to start aerial flight or land excursion.

Table 5. Drone Server API Endpoints

The drone server software was configured as a Linux service on the Raspberry Pis so that the process can run in the background. This allows all of the endpoints to wait until the client requests, effectively creating a RESTful API onboard each drone. The Linux service also allows the server software to be started when the drone is powered.

6.3.3. Communication

All communication from the user's computer that is running the APRS desktop application to the drone servers is done through HTTP methods. Since the drone systems will not be connected to the internet, an established connection between the client and drone must be made before any requests can be sent.

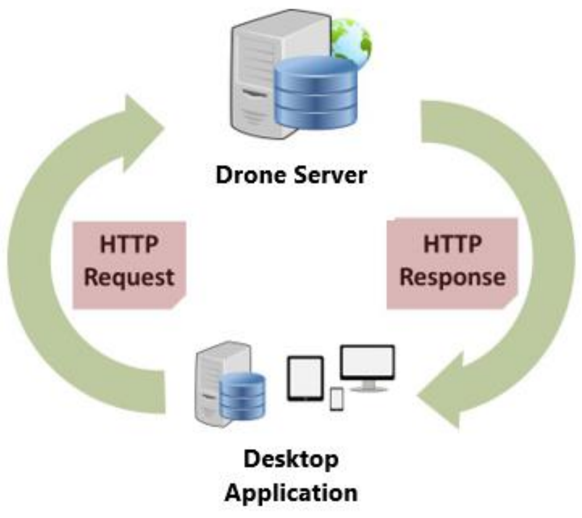


Figure 18. HTTP Request Architecture

Due to the range of the onboard Wi-Fi module, all communication between the drones and the user's computer will happen when the drones are stationary. This is to say that the aerial drone only transmits its flight data once the entire flight has ended, and the drone has returned to its starting position. Communication between each drone and the user's computer will occur at several points throughout the APRS's operation:

1. Uploading of aerial drone navigation instructions from client to aerial drone
2. Initiating launch of aerial drone flight
3. Downloading aerial drone flight data to client
4. Uploading of land drone navigation instructions from client to land drone
5. Initiating launch of land drone excursion

All communication is initiated by the client. The process for the client to interact with either drone is the same. First, the client must connect to the Wi-Fi signal broadcast by the desired drone. Second, the client must send an HTTP request with the required data. Third, the client must handle the returned data. All of these steps are handled automatically by the user interface in the desktop application.

6.4. User Interface

6.4.1. Description

The user interface for the APRS will be offered as a desktop application and will serve as the sole controller for the system. Within the UI, the user will be able to define all inputs, including their current location and the system area of operation, as well as transmit the inputs to the aerial and land drones. The software that is described in the Computer Vision and Mapping and Pathfinding and Drone Detection sections will be invoked through the UI. Any outputs, and error handling will be reported to the user through the UI. Full functionality of the APRS and UI is achievable with and without an internet connection. The major components of the UI are the system area of operation mapping tool, communication tasks, and error handling.

6.4.2. Application Structure

The user interface for the APRS is contained within an executable desktop application. This desktop application was developed using the following libraries, frameworks, and languages.

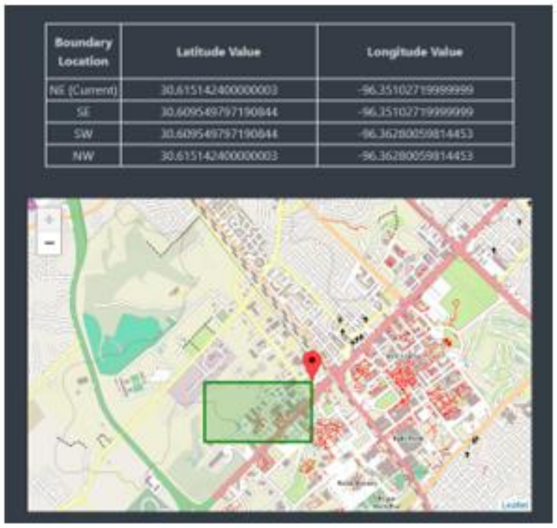
Library/Framework/Language	Notes
NodeJS	Used to install dependency packages, and setup application structure
JavaScript/TypeScript	Main language used in developing front-end application logic
Python/Flask	Used to develop back-end of application
ReactJS	Used to develop front-end visualization and event handling
Leaflet	Used to develop mapping tool
Node-WIFI	Allows application to connect to available wifi connections automatically
Child_process	Allows application to spawn child processes
CSS	Used to style front-end
Bootstrap	Used to style front-end
ElectronJS	Used to compile HTML/CSS/JS into a desktop application

Table 6. Desktop Application Stack

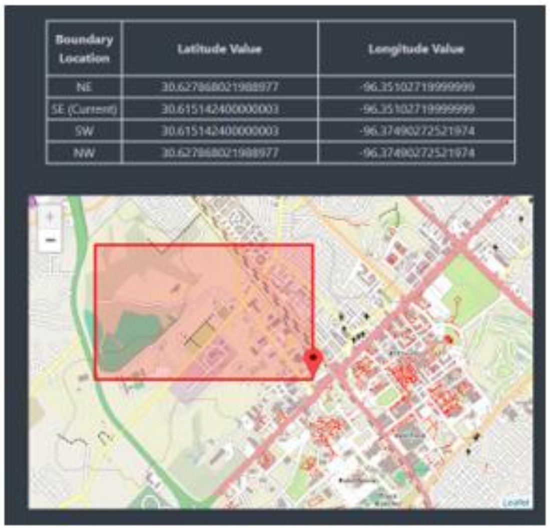
All of the dependencies listed above have been compiled into a single portable application. The application can be opened by running an executable that is contained in the portable application.

6.4.3. Mapping Tool

The mapping tool contained in the user interface allows the user to define the area of operation for the APRS. The area of operation creates the boundaries of the aerial drone's flight, and the potential land drone excursion routes. There are two steps to the mapping tool, including reporting current location and selecting a valid area of operation. The user must report their current location in terms of a latitude and longitude coordinate. Obtaining the user's current location is important because it restricts the possible areas of operation. If the user has an internet connection, they do not have to manually input their current coordinates, we can fetch their location for them using the Google Geolocational Services. Once the user enters a valid current location, the UI will show a map that allows the user to select the area of operation.



Boundary Location	Latitude Value	Longitude Value
NE (Current)	30.615142400000003	-96.35102719999999
SE	30.609549797190844	-96.35102719999999
SW	30.609549797190844	-96.36280059814453
NW	30.615142400000003	-96.36280059814453



Boundary Location	Latitude Value	Longitude Value
NE	30.627868021988977	-96.35102719999999
SE (Current)	30.615142400000003	-96.35102719999999
SW	30.615142400000003	-96.37490272521974
NW	30.627868021988977	-96.37490272521974

Figure 19. Mapping Tool: Valid and Invalid Selection

On the map, the user can move their mouse to define the area of operation. Per the APRS functional requirements, the area of operation must be no greater than 1 km^2. If the user attempts to select an area that does not meet this requirement, the rectangle will appear red in color, and not allow the user to lock-in their selection. Once the user has a valid selection, which appears green in color, the user can click to lock-in their selection.

Once the area of operation selection has been locked-in, an algorithm is invoked to generate the aerial drone's navigation instructions. The algorithm breaks the area into a grid made up of equally sized quadrants. The quadrant size is based on the F.O.V. limitations of the aerial drone's camera and the altitude at which the drone will fly.

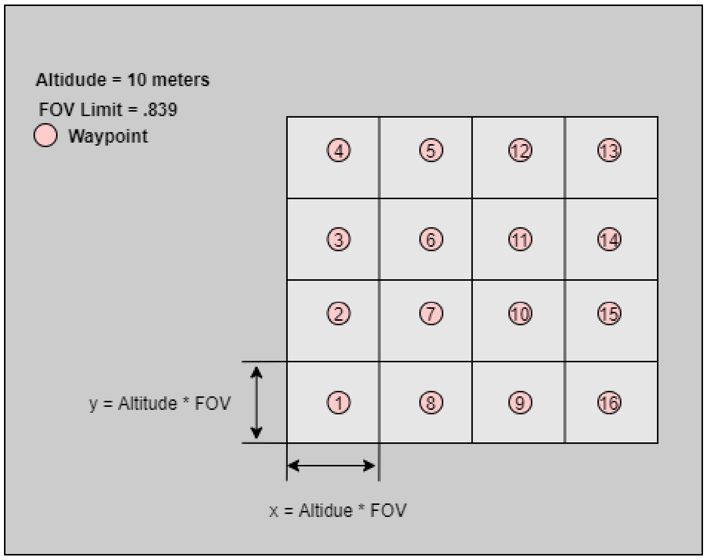
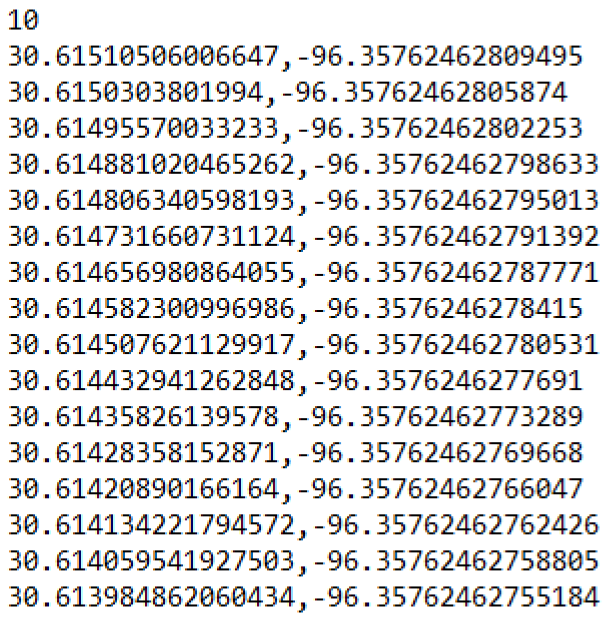


Figure 20. Map Quadrant Size

In the center of each quadrant, a waypoint made up of a single latitude and longitude coordinate is placed. This waypoint will serve as the instructions for where the aerial drone will fly to. Once all of the waypoints are placed, a navigation instruction file is created by compiling all of the waypoints. The navigation instruction file also contains a number representing the altitude at which the drone should fly at.



```
10
30.61510506006647,-96.35762462809495
30.6150303801994,-96.35762462805874
30.61495570033233,-96.35762462802253
30.614881020465262,-96.35762462798633
30.614806340598193,-96.35762462795013
30.614731660731124,-96.35762462791392
30.614656980864055,-96.35762462787771
30.614582300996986,-96.3576246278415
30.614507621129917,-96.35762462780531
30.614432941262848,-96.3576246277691
30.61435826139578,-96.35762462773289
30.61428358152871,-96.35762462769668
30.61420890166164,-96.35762462766047
30.614134221794572,-96.35762462762426
30.614059541927503,-96.35762462758805
30.613984862060434,-96.35762462755184
```

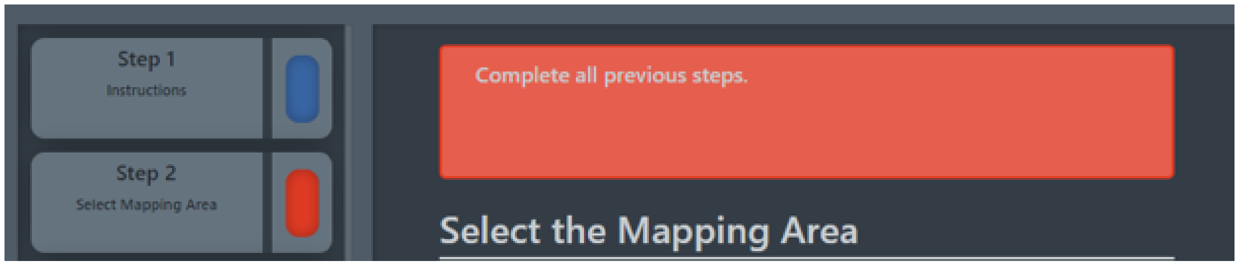
Figure 21. Navigation Instruction File

6.4.4. Communication Tasks

After selecting the area of operation, the majority of the user interaction with the APRS desktop application involves transmitting data between the client and drone servers. For each of these tasks, there is a separate upload or download button to ensure that each task is completed before moving to the next. The UI provides the user updates on the data transmissions progress, such as showing a loading bar and the duration of the aerial drone's flight.

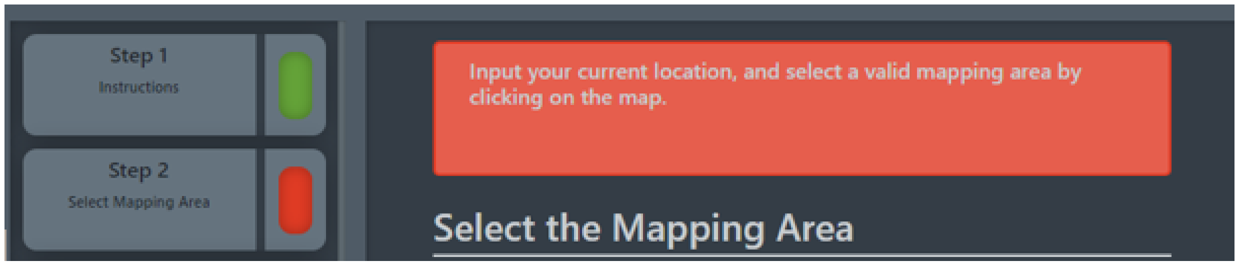
6.4.5. Error Handling

The user interface handles all errors by showing a notification with details of the error. The desktop application is organized into sequential steps, where one step cannot be started if the previous step has not yet been completed. This ensures that no race conditions occur during the system's operation, such as attempting to send navigation instructions to the aerial drone before they are generated.



Complete all previous steps.

Select the Mapping Area



Input your current location, and select a valid mapping area by clicking on the map.

Select the Mapping Area

Figure 22. User Interface Error Messages

Errors are handled for the following cases:

- Attempting to start a new step with incomplete previous steps
- Attempting to complete a step with incomplete processes remaining
- Inputting invalid latitude or longitude for user current location
- Attempting to get current location from internet without an active connection
- Attempting to select an invalid area of operation
- Attempting to communicate with a nonoperational drone

6.5. Subsystem Validation

In order to validate the Device Networking and User Interface subsystem, various tests were run to ensure the proper operation of the system was occurring. The tests ran can be broken into three categories, including networking, UI logic, and accessibility.

6.5.1. Networking Validations

To complete the networking validation, tests were setup to capture the packet size and latency time when communicating between client and drone server. Additional tests were setup to validate the packet size and latency time when fetching assets from the back-end of the desktop application. The following values were obtained from the desktop application's developer console.

Test	Results
Fetch user's current location via geolocation api call (internet required) in under 5s	avg time to fetch = 145 ms size of data = 442 B
Communicate with drone's server in under 10s	avg time to post data = 2.91 s avg size of data = 388.38 kB

Fetch map images in under 5s (offline assets served by back-end)	avg time to fetch = 253ms avg size of data = 76.3 kB

Table 7. Network Validations

6.5.2. UI Logic Validations

To complete the UI logic validation, tests were setup to assess the error handling of the application and the accuracy of the generated navigation instruction file. Through the testing of edge cases on the user interface, it can be seen that the application prevents any invalid operations from occurring. The error messages provided in the UI prevent the application from crashing and inform the user of their errors. According to the defined validations for this subsystem, the generated navigation instructions should be accurate within 1 meter. To test this, a latitude and longitude distance calculator was used. The following results were obtained from the test.

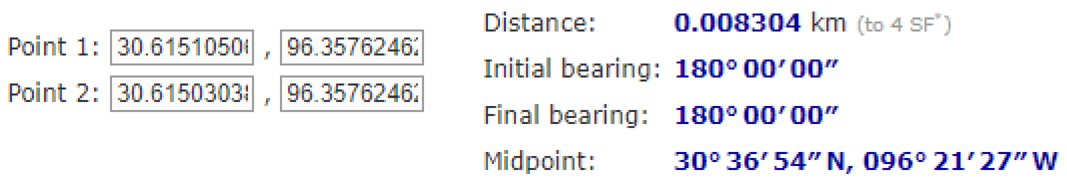

Point 1: 30.61510501 , 96.3576246
Point 2: 30.6150303 , 96.3576246
Distance: 0.008304 km (to 4 SF*)
Initial bearing: 180° 00′ 00″
Final bearing: 180° 00′ 00″
Midpoint: 30° 36′ 54″ N, 096° 21′ 27″ W

Figure 23. Coordinate Distance Calculation

The results from the calculation shows that the two waypoints from the navigation instruction set are spaced by 8.304 meters. According to the APRS mapping tool's algorithm, they should be spaced by 8.39 meters. This falls below the threshold of 1 meter of error. Based on these tests, the following validations are passed.

Test	Results
Error handling and message provided in GUI to prevent crash and inform user	Error messages will be presented if the user attempts invalid operation
Create drone navigation instructions accurate within 1 Meter	Accurate within .01 meter

Table 8. UI Logic Validation

6.5.3. Accessibility Validations

To complete the accessibility validation, tests were setup to ensure that full functionality of the application could be achieved when the source code is built into an executable. This was proven to be true by testing the operation of the system.

Test	Results

Install software via encapsulated windows installer	create a single application containing all functionality and dependencies

Table 9. Accessibility Validations

6.6. Subsystem Conclusion

As it is now, the Device Networking and User Interface subsystem is setup well for the next phase of our project. Much of the functionality is complete, such as the area of operation mapping tool. Other functionality, like the communication between the desktop application and the drone servers is mostly complete, although some details may need to be fine tuned based on my team's results from their subsystems. In these cases, the infrastructure is already setup, so fine tuning the details should not be a troublesome task. As mentioned, next semester's work will mostly involve integrating my teammates hardware and software into the desktop application.

Appendix A: Minor Computer Vision & Mapping Stages

```
def gradient_map(dispmap):
    """Gradient field derivation stage.
    Implements a simple Sobel filter.
    For simplicity, edges are rolled over as on a torus.

    dispmap -- Disparity map to derive a gradient from
    """
    # Collect vertical and horizontal differences
    dx = np.roll(dispmap, 1, axis=1) - dispmap
    dy = np.roll(dispmap, -1, axis=0) - dispmap
    # Return Euclidean differences by sqrt(dx^2 + dy^2)
    return np.sqrt(np.clip(np.square(dx) + np.square(dy), 0, None))


def mark_obstacles(grad, height):
    """Obstacle delineation stage.
    For each element, set to 1 if less than threshold, 0 otherwise.

    grad   -- gradient field to operate over
    height -- threshold to compare against
    """
    # Consider any slope too steep an obstacle boundary.
    return grad < height
```

Appendix B: Reused Coordinate Checker

```
##cycle_finder will check if current is already in the list (meaning
that a cycle has been found)
##this is useful in pathfinder because the conditions will include this
function which
##will make sure that a cyclic path will not be taken
def cycle_finder(current):
    for u in mystack:
        if (current == u.current):
            return True
    return False
```