Human-UAV Collaborative Search with Concurrent Flights and Re-Tasking

Alexander T. Broz

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Mechanical Engineering

Kevin B. Kochersberger, Chair

Alexander Leonessa

Ryan K. Williams

August 3rd, 2022

Blacksburg, Virginia

Keywords: UAVs, Human UAV interaction, Multiple UAVs, UAV search

Human-UAV Collaborative Search with Concurrent Flights and Re-Tasking

Alexander T. Broz

(ABSTRACT)

This thesis discusses a system that allows an operator to use two unmanned aerial vehicles (UAVs) to search an area. Prior work accomplished this in separate survey and search missions, and this work combines those two missions into one. The user conducts a search by selecting an area to survey, and the first drone flies above it, providing up to date information about the area. Points of interest (POI) are then marked by the user and investigated by the second drone. This system assumes a static and known obstacle map, and segmenting the environment during the missions leaves potential for future work. Both drones are equipped with cameras that stream video for the user to observe. A custom graphical user interface (GUI) was created to allow for the drones to be controlled. In addition to marking a search area and POI, the user can pause the drone and delete or add new POI to change the mission mid-flight. Both drones are commanded remotely by a ground station (GCS), leaving only low-level control to the onboard computers. This ground station uses a nearest neighbor solution to the travelling salesman problem and a wavefront path planner to create a path for the low altitude drone. The software architecture is based on the Robot Operating System (ROS), and the GCS uses the MAVLink messaging protocol to communicate with the drones. In addition to the system design, this paper discusses UAV human interaction and how it is applied to this system.

Human-UAV Collaborative Search with Concurrent Flights and Re-Tasking

Alexander T. Broz

(GENERAL AUDIENCE ABSTRACT)

This thesis discusses a system that allows an operator to use two drones to search an area. Prior work accomplished this in separate survey and search missions, and this work combines those two missions into one. The user conducts a search by selecting an area to survey, and the first drone flies above it, providing up to date information about the area. Points of interest (POI) are then marked by the user and investigated by the second drone. This system assumes that obstacles in the environment are static and already known. Both drones are equipped with cameras that stream video for the user to observe. A custom graphical user interface (GUI) was created to allow for the drones to be controlled. In addition to providing the initial mission for the drones, the user can also change the mission mid-flight. Both drones are commanded remotely by a separate computer, leaving only very basic control to the drones. This ground station uses a simple path planner to create a path for the low altitude drone to avoid obstacles. The software architecture is based on the Robot Operating System (ROS), and the GCS uses the MAVLink messaging protocol to communicate with the drones. In addition to the system design, this UAV human interaction and how it is applied to this system.

Acknowledgments

I want to thank my advisor Dr. Kochersberger for his guidance of my research and support in starting my career. Dr. Kochersberger helped me navigate requirements for my project and introduced me to my future employer. I would also like to thank everyone in the Unmanned Systems Lab for their help in adapting my robotics knowledge to drones and for piloting during test flights, especially Manav Gandhi, Avery Sebolt, Jonathan Keller, and Caleb Adams. Finally, I want to thank my family and friends for their love and support.

Contents

List of Figures

List of Tables

List of Abbreviations

GCS Ground Control Station

GUI Graphical User Interface

HAS High Altitude Surveyor

LAI Low Altitude Investigator

POI Point(s) of Interest

ROS Robot Operating System

HAS is the drone flying at high altitude with a downward facing camera, used to survey the search area.

LAI is the drone flying at low altitude with a forward facing camera, used to investigate the search area.

ROS is an open-source framework commonly used in robotic and autonomous systems development.

GUI is a graphical way for a user to interact with a computer, in this case used to control drones.

GCS communicates with drones remotely from the ground.

POI are points chosen by the user for the LAI to investigate.

Chapter 1

Introduction

Drones, formally known as unmanned aerial vehicles (UAVs) are becoming increasingly popular as their capabilities improve and costs decrease. This is leading to more uses for drones, including military, search and rescue, industrial inspection, and recreation. With all of these uses, autonomy and human interaction are key features in drone systems. For use in search operations, a drone must be able to navigate through it's environment without crashing, part of the system must decide where to search, and the findings of the search must be communicated to a user. Emergency responders often manually control drones, which covers the navigation and decision requirements but requires a constant attention [7]. This workload can be reduced by using autonomous control for the drone. However, the computer processing hardware necessary for this automation takes up valuable space and weight on the drone, leading to shorter flight times and less room for other payloads or require a larger, more expensive drone. Utilizing a distributed control architecture can overcome this problem by handling some or all computations on a separate computer wirelessly networked with the drone, with the added benefit of making UI integration easier [12].

This paper discusses the design and use of a distributed system using two autonomous drones to search an environment with user input, expanding on previous work done in the Unmanned Systems Lab (USL). A system created by a prior student at USL used two drones and a base station to search an area in three separate phases [6]. First, one drone surveyed a search area with a downward facing camera and stored the images. These images were then

stitched together on the base station, and finally, the user marked points on the stitched image for the second drone to investigate with a front-facing camera. In this original system, the drones flew at separate times. The new system follows a similar pattern but allows the user to fly both drones at the same time and re-task the investigating drone during flight. The focus of this work is on the simultaneous flight and re-tasking, so obstacles are assumed to be static and known. A system like this can keep the cost of drones low because of the distributed control architecture, and could be used for a variety of search operations by adding sensors to the drone. Adding thermal cameras might improve the effectiveness of rescue search missions, while adding a radiation detector could be used to search for leaks or weapons. Current work at USL is using microphones mounted on a drone to detect and locate nearby drones. Parts of the system described here are being investigated as a potential basis for searching an area for UAVs in this manner.

The software of the system could also be extended to use probabilistic methods to aid the user in selecting points of interest (POI). One example of this is the preference elicitation algorithm created by Ray et al. [16], discussed in 2.2. This algorithm learns a user's preference for POI and applies that preference to the entire map, reducing the time needed for an operator to mark POI for a drone. A simpler solution to aiding the user in selecting POI is explored in this work, where the computer highlights the intersections of roads and buildings to clarify high priority search areas.

Chapter 2

Review of Literature

2.1 Human-UAV Interaction

Human-robot interaction is an established field, and human-UAV interaction is a relatively new field very similar to and in some ways extending human-robot interaction. Tezza and Andujar [19] explain different types of interaction, user interfaces (UI), and what the interaction should achieve. In any interaction with a drone, a human can take on one of four main roles. An *active controller* has direct control of the drone, such as a pilot with a radio transmitter controlling the drones with joy-sticks. A *recipient* has no control of the drone but benefits from the interaction, such as a customer receiving a delivery from a drone. The role of *social companion* does not have any requirements for user control of the drone, but the user has some sort of social interaction, like with the Joggobot running companion. Finally, a *supervisor* has indirect control of a drone and supervises the entire flight. All of these types of interaction require some kind of UI, and the main goal of any UI is to be intuitive. An imitative UI interprets user actions to control the drone, like raising a hand to control altitude or head turning to control yaw. The simplest UI is instrumented, in which a user interacts with a physical, digital, or imaginary object to control the drone. Lastly, an intelligent UI interprets high-level commands from the user.

A novel example of a user acting as an active controller using an imitative UI is the system created by Maher et al, which allowed a user to use gestures to control a micro-UAV in a GPS

denied environment [13]. A modified version of the Tiny-YOLO object detection framework was used to find the user's face and hands. Gestures were then differentiated by the ratio of the heights of the hands and face, with a certain range of ratios corresponding to a certain gesture. These gestures were designed to be intuitive and mean a simple command for the UAV, such as take off, hover, yaw, or fly in a direction. All of the image processing was done on a local computer which sent interpreted commands to the drone, allowing gestures to be processed at a rate of 21 frames per second.

Another method for humans to interact with drones is by physically interacting with the drone istself, rather than controlling it through another device. Rajappa et al. created a way to do this by adding a sensor ring around a UAV [15]. Robots that aren't UAVs already interact with humans in this way, but this is uncommon with UAVs because they are perceived as unsafe for physical interaction and it is more difficult to implement. The implementation difficutly comes from the fact that UAVs are more susceptible to outside forces, like wind, than other robots. Both of these limitations were mostly overcome by adding the sensor ring to the drone. This ring was mounted onto buttons past the range of the propellers, allowing the drone to sense where the force of an interaction was coming from and shielding the user from the propellers. Using the response of the drone and the input from the ring, Rajappa et al. were able to separate the forces and torques of human interaction from other disturbances for continuous, sudden, and multiple contacts. Based on the interaction, the drone would modify it's behavior to account for the interaction, such as resisting the pull resulting from a human taking a tool that the UAV delivered without landing.

One of the main advantages of human-UAV collaboration is that humans have a reduced workload due to the autonomy while the shortcomings of autonomous intelligence can be overcome with human intelligence. Bevacqua et al. [5] proposed a framework to seamlessly

transition drones between different levels of autonomy and human control, based on the idea that an operator may not be fully dedicated to drone operation. The framework divided human interaction with the system into three different levels of abstraction. An operator could give high level commands, like defining new search areas or strategies, make a navigation adjustment, like modifying the path of a drone, or take over the control of a single drone entirely to inspect difficult environments. Due to the different levels of interaction, the proposed system would also use different kinds of input. These varying levels of interaction also served to allow the user to update the mission using new information. The system was testing on simulated alpine search and rescue and found to be better than fully autonomous or fully human controlled search.

With all human-UAV interaction, the system designer must choose how the user collaborates with the drone. Angrawal et al. [2] defined a model for human-on-the-loop interaction and a process to elicit requirements for the interaction. Interactions are split into four groups. Since the drones can act autonomously, the must provide rationale to the user to improve the user's situational awareness. Similarly, the drone must provide information behind that rationale, and the user might provide the drone with information to improve it's awareness. If necessary, the user can intervene through commands or feedback. Finally, the user can also intervene by changing the level of autonomy for a drone. In all of these interactions, the interaction requirements are defined by what the interaction is, when it occurs, and why it occurs. For the cases of user intervention, how the drone responds to a command and determining additional system requirements with changing autonomy levels are also important.

2.2 UAV systems for Search Missions

Regardless of the method of interaction, systems involving both humans and drones have uses across multiple industries. One common use for these systems is search missions, including industrial, military, and civilian purposes. With all of these types of search, controlling a vehicle manually requires all of the operators attention. When the Boulder Emergency Squad uses a UAV for search, an additional operator is needed as an observer to help the pilot with the search [16]. In search and rescue missions, the first few hours of rescue are the most critical because survival rates drop quickly, so speed and accuracy are extremely important in search and rescue search [4]. Ray et al. [16] used a Partially Observable Markov Decision Process (POMDP) to make it easier for an operator to mark POI for a drone, making a search faster and require only one operator. The algorithm learns user preferences for POI and marks similar locations across the entire search area. First, the user provides some POI on a map of a segmented environment, which are represented as an array of percentages of the surrounding segmented objects (eg: road, building, grass). The algorithm then suggests points that it believes match the users preference, and the user can accept or reject those points. Using the information on which points the user accepted and rejected, the algorithm refines its belief of the users preference and propagates that belief across the entire search area to create a mission for the drone. The POMDP was evaluated against a greedy preference selection algorithm and performed better for three different levels of complexity.

Another way to speed up a search by using multiple drones, and Cleland-Huang and Angrawal [7] proposed a framework for emergency responders to use multiple semi-autonomous drones for search and rescue. Currently, emergency responders often manually control or pre-program waypoints to control the drone. Instead, the proposed framework allows a user to choose a mission type and search area for the drones to investigate. When the drones

find a victim, they notify the user, and if the user confirms the find, the drone tracks the victim until they are rescued. Due to the chaotic environment often found at these scenes, Cleland-Huang and Angrawal suggest using a graphical user interface over gestures or voice control.

A powerful tool to improve UAV search and rescue by better controlling multiple drones is swarming. Arnold et al. proposes a search and rescue framework using swarming drones, and defines swarming as a combination of simple behaviors performed by multiple drones at the same time yielding an intelligent system. In this framework, drones in the swarm are assigned one of three roles, each with different priorities and behaviors. "Relay" drones provide network infrastructure, allowing other drones to travel further. "Antisocial searcher" drones prioritize expanding the search area and increasing the spread of the swarm. Finally, "social searcher" drones search near previously identified survivors, based on the reasoning the survivors tend to congregate and move towards safer areas together. After testing various swarm sizes and compositions in simulations, Arnold et al. found that these parameters for a drone swarm performed better than similar existing swarms. It was also found that the swarm generally functioned best with more drones in total and a low number of relay drones, but this varied from case to case.

Alotaibi et al. [3] created a different method to task a swarm of drones based on dividing the environment into layers. These layers expand outward from the center of the disaster, and one drone is assigned to each layer and searches for survivors in that layer. If there are more drones than layers, additional drones are assigned to each layer starting at the center. This prioritizes searching in the center of the search area because the algorithm assumes more victims will be located near the center of the disaster. Additionally, when more than one drone is in a layer and a survivor is found, the drone that found the survivor will suspend its search until the survivor is rescued.

Most literature on using drones for search and rescue focuses on land-based solutions. Lomonaco et al. [1] discuss a use for a drone swarm to rescue migrants crossing the Mediterranean Sea. Currently, these rescue operations use ships and manned aircraft, both of which are very expensive to operate. The main problems with using drones in sea rescue are the limitations of the battery, network connectivity, and hostile environment. Lomonaco et al. suggest using a drone swarm to overcome limitations due to battery and connectivity while reducing operating cost compared to manned search vehicles.

2.3 Distributed Systems

With all UAVs, payload space, power, and weight are very limited, and computing power necessary to control the drone can take up a lot of it. To reduce drone payload and increase flight time for a search mission, Hummel et al. [12] used a distributed control architecture. As small UAV equipped with a camera streamed flight data and video over WiFi to a cell phone, with only low-level flight control performed on the drone. The phone had a digital model of the drone, and used that alongside image processing to find a path for the drone in real time. This system was used to scan books for library inventory, so in this case the phone also used computer vision to identify book titles. The distributed control architecture used allowed the system to do much more complicated computing than if all computing was done on the drone, made integrating a GUI much easier. The downside to this architecture is that the control algorithm must consider the delay from sending and receiving data over WiFi.

2.4 Prior USL work

Chourey [6] previously created a system at the Unmanned Systems Lab at Virginia Tech that used a distributed control architecture and GUI to allow a user to search an area using two drones. First, QGroundControl (QGC) was used on the base computer to command a high altitude drone to survey an area. Once complete, the survey images were stitched together into a 3D reconstruction which was used to generate a 2D occupancy map of the area. Next, the user selected search areas on a top-down view of the reconstruction and the computer used the occupancy map to generate a path for a low altitude drone to investigate those areas while streaming video back to the user.

3D reconstruction using multiple images can take a long time, so Chourey attempted image mosaicking and using computer vision to extract features like roads and buildings. Mosaicking was done by warping and stitching the images together. Off-line mosaicking, meaning done with all pictures at once, was successfully done using OpenCV, but was not used because the images from the survey were returned from the drone sequentially. On-line mosaicking, meaning continuously expanding the stitched image as new data is received, was attempted using OpenCV and a new solution. OpenCV was unable to process all of the images in a simulated on-line experiment, and while the new solution was able to create an output, the result was disjointed and distorted. Building and road detection for the purpose of creating an obstacle map was attempted using color and texture based segmentation, but the results were not reliable enough to use.

2.5 Path Planning

For any robot to navigate autonomously through an environment, some sort of path planning is required to tell the robot how to move in order to reach its goal without colliding with obstacles. When there are multiple goals for the robot to reach, the additional problem of the order in which to visit the goals must be addressed, which is known as the Travelling Salesman Problem. Solutions for creating a path for a robot to reach a single target and choosing the order in which a robot will travel to targets are presented below.

2.5.1 Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is a difficult problem in optimization, especially with a high number of locations to visit. Goyal [11] presents a collection of solutions to this problem, which can be split into deterministic and non-deterministic algorithms. The simplest solution is a brute force method, checking every possible path between the points to find the shortest, but this is extremely inefficient for more than a few targets. Deterministic solutions like the brute force method are exact, but are time-inefficient, space-inefficient, or both. Non-deterministic algorithms have the potential to drastically reduce run time, but do not always guarantee the best solution. A simple non-deterministic algorithm is the "Nearest Neighbor" algorithm, in which the next target is the closest to the current location. Chourey [6] used a genetic algorithm, which is non-deterministic, implemented by the MLRose Python library to solve the TSP.

2.5.2 Path Creation

Wavefront and A* planning algorithms are commonly used path planners that are both fast and relatively simple. These two algorithms are explained by Zidane and Ibrahim [22], and they compare the algorithms performance on mazes of varying complexity.

The wavefront search algorithm is limited to a grid map with obstacles and the goal location marked, commonly by 1 and 2, respectively. The algorithm then expands out from the goal location either to neighbors that share an edge, known as 4-sector connectivity, or neighbors that share edges or corners, known as 8-sector connectivity. In both cases, neighboring spaces that are empty are marked with a number one higher than the current space. Limiting the expansion to empty spaces ensures that the obstacles will not be part of the path and that spaces that are already part of the expansion are not overwritten. This expansion is repeated from every space with the current value, and then with the cells of the next value. Once the starting location is reached, the path is found by always transitioning to the neighboring cell with a value one lower than the current cell until the goal is reached.

While the wavefront algorithm is limited to a grid, A* finds a path through a graph, which is a type of data structure that includes grids. Beginning with the starting position, the algorithm uses a cost function involving the distance from start to current location and current location to goal to make the best move to reach the goal. Along the way, it records locations that have been visited and locations adjacent to visited ones in separate lists. Locations that have already been visited are recorded along with the index of the previous location that has the lowest cost. Once the goal is reached, the path is determined by following indices of previous locations.

Zidane and Ibrahim found that the order of increasing code complexity and run duration for path finding in a grid was 4-sector wavefront, 8-sector wavefront, then A*. The order

of shortest resulting path was the opposite. They also found that down-sampling the grid improved computation time at the cost of returning a less optimal path.

2.6 CameraTransform Python Module

Drone imagery has a lot of uses, but survey imagery can be difficult for a computer to interpret if the image plane is not parallel to the ground. CameraTransform [10] is a Python module for manipulating the perspective of an image and can therefore change the plane of an image. This is done using the intrinsic and extrinsic properties of the camera. Intrinsic properties are the internal the camera, including focal length, image size, sensor size, and sensor center. The extrinsic properties of the camera are its position and orientation in the world, including both local position centered on the ground directly below the camera and global position in latitude, longitude, and altitude. When the extrinsic parameters are unknown, CameraTransform can find them through fitting known information such as object height or global position. Once determined, the camera parameters can be used to project the image into any plane. For any picture with a known extrinsic matrix, CameraTransform can find the local and global position of any pixel.

Chapter 3

System Design

The following sections discuss the hardware and software used for the two-drone flight. The hardware is broadly divided into the two drones and the base station (base), and the software is divided into the drones, the ground control station (GCS) and user interface (GUI), as shown in Figure 3.1. The High Attitude Surveyor (HAS) provides nadir imagery to the user and the Low Altitude Investigator (LAI) shows a horizontal view of the environment as it flies.

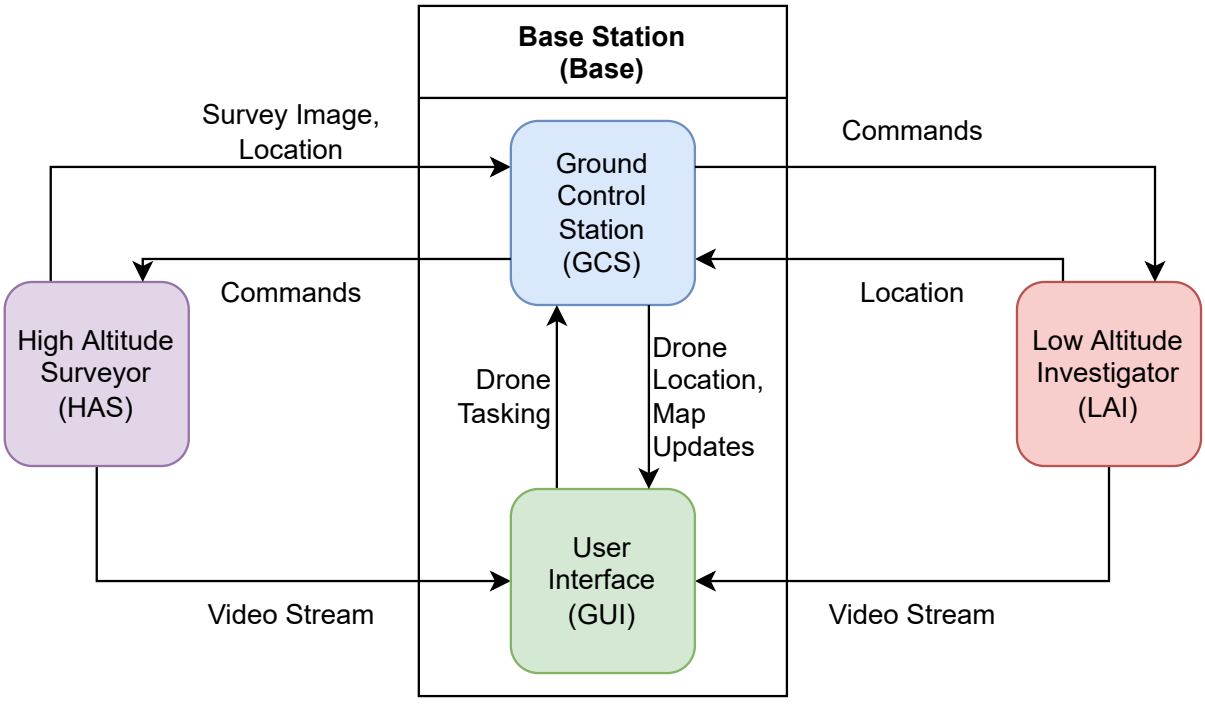


Figure 3.1: Overview of hardware and software system architecture.

3.1 Target Environment

This system requires a partially-known environment to operate, meaning there must be some sort of initial map that the user can use to select an area to survey. When this system was tested, previous survey imagery encompassing a larger area than a single mission was used. Creating an updated obstacle map from 2D imagery is a difficult problem, and 3D reconstruction of an environment would make this trivial but takes a very long time for imagery from a single camera [6]. To allow the survey and investigation missions to run at the same time, the system assumes a known and static obstacle map, and survey data is used only to show the user updated information. Additionally, it is assumed that the survey height is higher than any obstacles in the area. Finally, the survey height is limited to various heights depending on location, which limits the size of the search area.

3.2 Hardware

This section describes the various hardware components that make up the subsystems in 3.1. All of this hardware was chosen because it was readily available, easy to use, and sufficient to prove that this system functions as intended. A diagram of the major hardware components and how they are connected is shown in Figure 3.2.

3.2.1 UAVs

Both the LAI and HAS are Holybro S-500 quad-copters with almost identical equipment and payload, as seen in Figure 3.3. They use Pixhawk 4 Mini autopilots with ArduCopter firmware to control the drone. The drones are equipped with GPS, Ubiquiti long range WiFi, and radio antennas. Each drone also carries a Raspberry Pi 4 model b microcomputer and

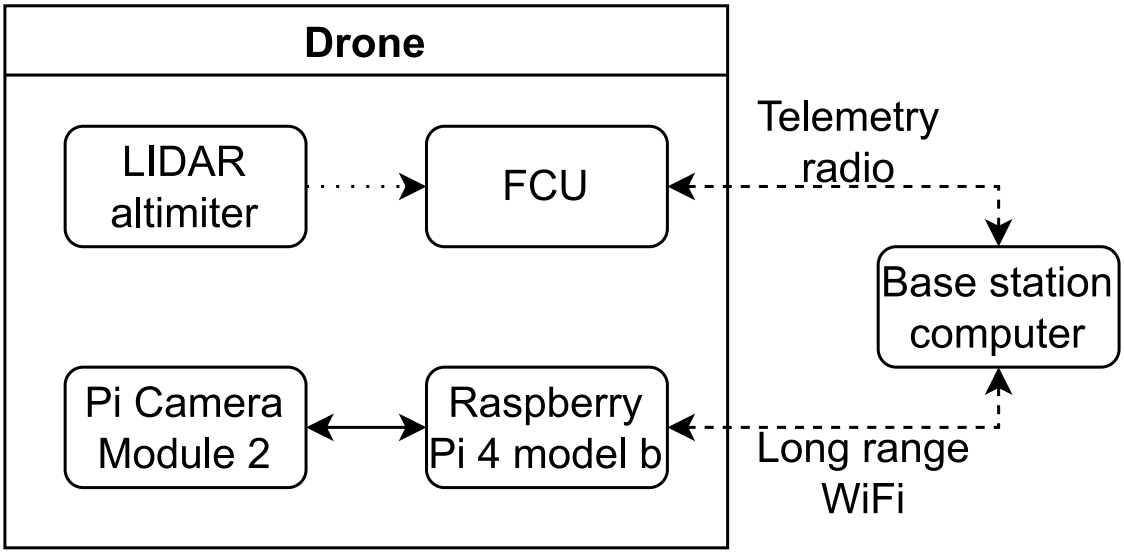


Figure 3.2: Hardware schematic for the system. Only one drone is shown since the hardware is identical, with the exception of the LIDAR altimeter only on the LAI.

Raspberry Pi Camera Module 2 (Picam v2). The Raspberry Pi and Picam v2 are attached to the drone via a 3D printed mount designed by Colin Fox, an undergraduate student at USL. All of this is powered by a 4s lithium polymer battery with voltage converters to step the voltage up and down for various components. There are two differences between the LAI and the HAS. The HAS has a downward facing camera while the LAI has a front facing camera, and the LAI has a LIDAR altimeter. With all of the payload and the battery, the drones have a flight time of at least 15 minutes.

The Picam v2 can record video at up to 1080p (1920 x 1080 pixels) and is easily integrated with any Raspberry Pi computer. The Raspberry Pi 4 model b is the most powerful Raspberry Pi computer available, with sufficient computing resources and I/O ports for this system.

Figure 3.3: Holybro S-500 drone with payload. The LAI is shown, but HAS is nearly identical.

3.2.2 Base Station

The base station is a computer with two telemetry radios are plugged in, each paired with one of the drones. Separate from the computer but also on the ground is another long range WiFi antenna. The WiFi antennas on the drones acted as access points to the WiFi on the round, ensuring that the Raspberry Pi's shorter WiFi range did not limit the aircraft range.

3.3 Software

This section explains the software tools and architecture that make up the subsystems in Figure 3.1. The Raspberry Pis on both drones run the same video streaming software (3.3.5), and the GCS and User Interface both run on the base station computer. The GCS software is split into the drone interface (3.3.3), map creation (3.3.7), and path planning (3.3.6). The

User Interface is split into drone tasking (3.3.2) and video streaming (3.3.5). All computers use the Ubuntu Linux operating system and all of the code is written in Python 3.

All of the software was written to be modular, so pieces can be replaced or added to better suit a particular use case without having to re-create the entire system as long as the new software uses the same inputs and outputs as the old. For example, a different kind of path planner could be used for either drone as long as it uses the same format for input and output locations. Figure 3.4 shows all of these parts and their interactions.

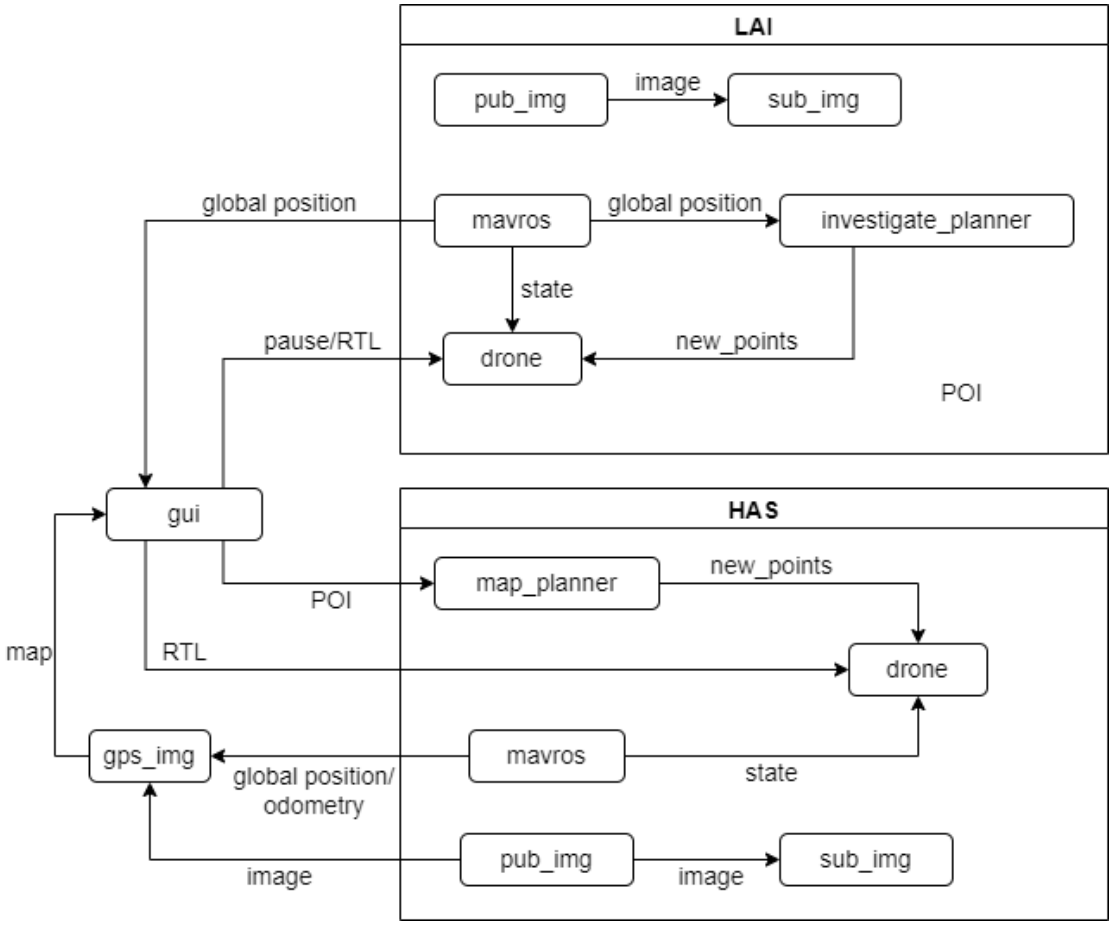


Figure 3.4: Individual programs and their interactions in the software of the system, displayed as ROS nodes and topics.

3.3.1 Commonly used tools

Below are brief explanations of software tools and protocols that are used throughout the code.

Robot Operating System

The Robot Operating System (ROS) is an open-source framework often used in robotics development [17]. It provides a way to run multiple programs, called nodes, at the same time and facilitates communication between them. Nodes communicate by publishing and subscribing to topics, which act like a bus for transmitting messages. Nodes and topics can have namespaces, similar to folders in a file system, which isolate sections of the ROS system. Nodes can only publish and subscribe to topics in their namespace, allowing for duplicate nodes and topics. ROS was used in this system to communicate between the user interface, path planners, cameras, and drone interface. The LAI and HAS used many of the same nodes and topics to function, so each drone had a corresponding namespace, shown as boxes in Figure 3.4. Outside the namespace were parts of the system that interacted with both drones, mainly the GUI. This use of ROS architecture allowed the parts of the system for one drone to be reused for the second drone.

MAVROS and MAVLink

ROS has many packages available online that can make ROS development faster by reducing work, and one such ROS package is MAVROS, which was used in this project to communicate between the GCS and the drones [8]. The ArduCopter firmware on the drones uses the MAVLink communication protocol to send and receive commands from the GCS. MAVROS functions as a bridge between ROS and MAVLink by converting commands from other

ROS nodes into MAVLink messages and sending them to a drone, and performs the inverse operation with data received from a drone

Coordinate Conversion

Determining the location on the globe of a point in a 2D image is a difficult problem since the Earth is round and an image on a computer screen is flat. To overcome this difficulty, many different coordinate systems exist to represent position on the Earth, and many projections exist to convert between coordinate systems [9]. This project used the pyproj Python 3 module [21] to transform coordinates between coordinate systems and the Rasterio Python 3 module [14] to store coordinate system data with pictures in a GeoTIFF image as well as convert between pixel location and location on the globe. Pixel coordinates are mapped to world coordinates in a specific frame by an affine matrix.

3.3.2 Drone Tasking

The user acts as a supervisor in this system because they continuously monitor the drones and are responsible for tasking the drones (2.1). When the system is started, the user is presented with an old aerial image of the mission area and several buttons to control the drones, shown in Figure 3.5. The user first selects a survey area by dragging the mouse and launches the HAS to survey this area by clicking the ”Map” button. Once HAS reaches its survey location the GUI is updated with a new image of the chosen search area. The user can then select points of interest (POI) on the updated image and click the ”Investigate” button to launch the LAI. At any time during the mission, the user can click the ”Pause” button to make the LAI stop, and the ”Resume” button to continue the mission. The user can also re-task the drone during flight or pause by adding and removing POI from the image

and clicking the ”Investigate” button again. If the user chooses to re-task the drone during a pause, clicking ”Investigate” will also un-pause the drone. When the mission is complete, the user can click the ”Land” button to have both drones return to their initial position on the ground. Figure 3.6 shows the sequence of actions a user can take during a mission, starting with selecting a search area.

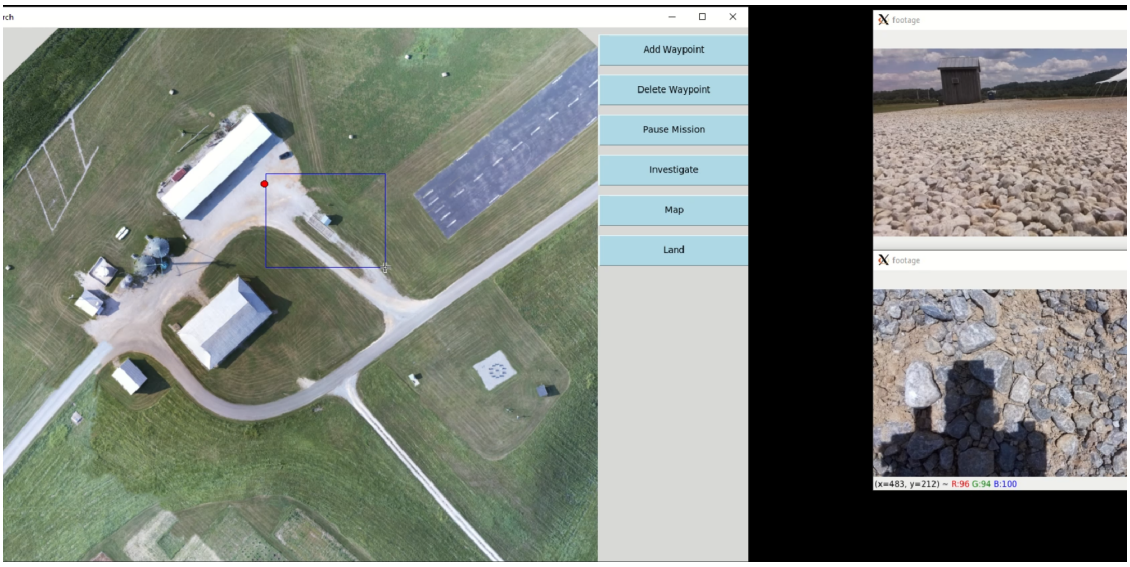


Figure 3.5: User interface at startup.

The ”Land” and ”Pause” button commands are sent directly to the drone interface since they require no further processing. However, the search area and POI sent by clicking the ”Map” and ”Investigate” buttons are first sent to path planners to convert them into waypoint missions for the drone to follow. This information is all converted from pixels to latitude and longitude before being sent to ensure the coordinates are understood properly by the path planners.

The GUI was built using the Tkinter python package and based off of the GUI made by Chourey for his work at USL. Additional buttons were added and some functionalities were changed to adapt the GUI to the new system in addition to integrating it with ROS.

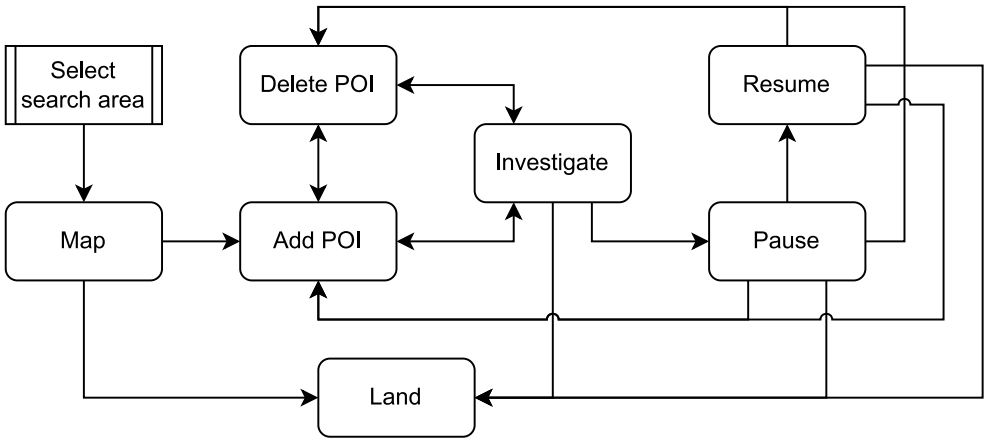


Figure 3.6: Possible user actions on the GUI, starting with "Select search area." The "Land" button can be used at any time and is excluded from the diagram for clarity.

Despite this system only using two drones, it can still be considered a swarm. In this case, the roles of drones are surveyor and investigator. The current system is limited to one drone in each role due to the path planning, but adding steps to separate the survey and investigation goals between drones could adapt the system to a larger swarm of drones.

3.3.3 Drone Interface

The main node in the software architecture interfaces between the drone, via MAVROS, and the rest of the programs running on the base station. This node modifies parameters, transitions between various modes, arms and disarms the motors, and uploads missions in the form of waypoints according to user input. The node is also designed so that is can be used independently of the rest of the software to ease future development work.

Radio Problems

As mentioned in section 3.2, each drone has a telemetry radio and the base station has two radios, one paired with each drone. The radios operate on the same frequency but

have different ID numbers, meaning they should be able to operate side-by-side without any interference. However, it was found that packet loss increased significantly with this setup. This packet loss often caused uploading missions to the drones to fail, meaning the drones would never fly.

Several solutions to this problem were investigated and tested. First, the base station was changed to have only one radio to communicate with both drones, which was accomplished by giving all of the radios the same ID. The drones were then distinguished by a unique system ID number. The first problem with this arose from the fact that only one client can connect to a serial port on Linux, meaning only one instance of MAVROS could connect to the port, leaving one drone with no communication to the base station. A workaround for this was found by creating a virtual serial bus on the base station, but the extra messages from an additional drone still caused MAVROS to malfunction.

Another potential solution using only one radio on the base station was creating a program that would listen to the radio traffic and split the message stream according to the system ID number. This solution was not tested due to time constraints and uncertainty about its potential success.

The solution that was ultimately used in the final system was to continue using one radio for each drone, but to send the missions in sections, called tasks, and retry sending those tasks if packet loss caused it to fail. This solution allowed for some tuning in the form of changing the number of waypoints in a task and the minimum distance between waypoints.

ArduCopter

As mentioned previously, the Pixhawk flight controller onboard the drone is responsible for low level control of the vehicle, in this case meaning moving to the location commanded by

the GCS. This is handled by autopilot firmware called Arducopter, which is a version of the open source UAV controller Ardupilot specific to multi-rotor craft. This firmware on the follows strict requirements for uploading and carrying out missions. The autopilot has many modes, all of which have different flight characteristics and allowed actions, but this system uses only *stabilize*, *auto*, *brake*, and *RTL*. When in *auto*, the drone flies autonomously between waypoints uploaded to the autopilot, but the drone cannot arm while in *auto*. The *stabilize* mode does allow arming, and levels roll and pitch during flight. *Brake* brings the drone to a stop mid-air, and *RTL* returns the drone to the launch position by bringing the drone to a pre-programmed height before flying to the launch coordinated and descending.

Uploaded waypoint missions are only activated once the drone switched into *auto*, so when already in *auto* the drone switches to *brake* mode and then back to *auto*. The autopilot can only store one waypoint mission at a time, so the drone must come to a stop and switch out of *auto* when a new set of waypoints is uploaded. Attempts were made to upload new waypoints while the drone was flying between existing ones, but this resulted in multiple errors and often led the drone to stop in mid-air due to a corrupted or incomplete mission. *RTL* can operate a variety of ways by changing parameters on the autopilot. For this system, the drone flies to a height known to be above all obstacles and maintains that height until descending directly over the launch position.

At startup, the GCS switches the drone to *stabilize*. Once the first task is uploaded, the drone is armed and switched to *auto*. At the end of each task while waiting for the next task or during a pause, the drone is switched to *brake*. Upon resuming the mission or after the next task is uploaded, the drone switches back to *auto*. Once the mission is finished, the GCS switches the drone to *RTL* and the drone automatically disarms when it lands.

3.3.4 System Modeling and Validation

The interaction requirements for the firmware discussed above combined with the workaround for the radio problems made the drone interface complicated. To ensure that everything worked as expected, interactions between the user, GCS, and drone are modeled and verified in UPPAAL. UPPAAL is a tool for modeling and validation of real-timed systems as timed automata, which are and extension of finite state machines that include time and continuous components in the form of differential equations. States in UPPAAL are represented as blue circles, with the initial state having a black circle inside. States can have an invariant, which is a condition that must hold for the system to be in that state. Transitions between states are called edges, which consist of selection, guard, synchronization, and update. Selection in an edge allows the model to choose a value nondeterministically from a given range. The guard is a condition that must be true for the edge to trigger. Synchronization allows edges across multiple state machines to fire at once. Finally, an update in an edge modifies model variables.

Once the model of a system is complete, UPPAAL can simulate the system by running the model and displaying the system state at each time-step. Models are validated in UPPAAL using the verifier, which evaluates logical statements provided by the user.

Several assumptions are made to reduce computation time, model a human interacting with the system, and model data that would normally come from the drone. First, a mission could theoretically be endless if the user simply keeps sending the drone locations to visit, so the maximum length of a mission is limited and the model terminates when the drone lands for the first time. Time is also not considered in this model in an effort to reduce computation time, because the transitions are no different at or between mission points and the drone flies autonomously between points. This turns the timed automaton into a finite

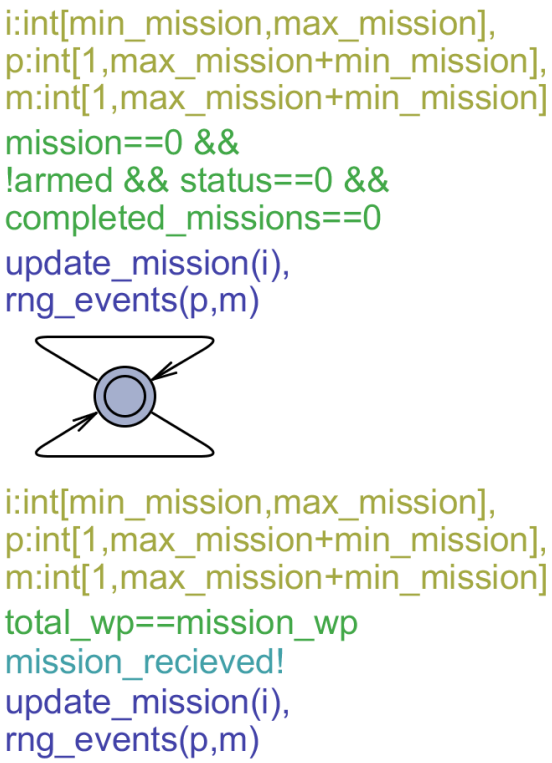
state machine. Next, human interaction is modeled as a nondeterministic choice with events happening at a specific point in the mission. These events can happen outside of the mission, causing them not to occur during simulation and modeling the scenario of the user never triggering an event. Finally, the drone sends data to the GCS specifying if it is in the air or on the ground, so the model includes additional variables as counters to determine this information. The models of the subsystems in the interface are shown in Figure 3.7.

Human Interaction

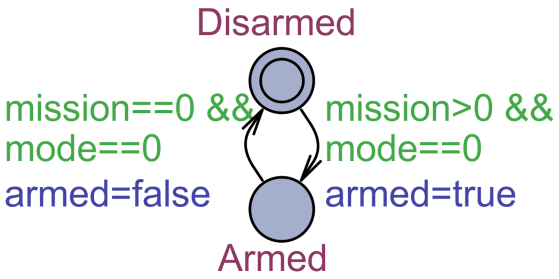
The human interaction in this model is represented by a state machine with a single state and two different edges looping back into that state, as seen in Figure 3.7a. Both edges contain the same nondeterministic selection for mission length, pause location, and re-tasking location. The possibility of events not happening during the mission is added by choosing event location from a larger set than mission length is chosen from. Both edges also call the same update functions, which apply the nondeterministic selections to the model variables. The differences between the edges come in the guards and synchronization.

The top edge is triggered only when the drone is on the ground, so the guard checks for the appropriate mode, motor state, and that the drone is in fact on the ground. The "completed_missions" guard statement was added to give the model a finite duration, since the drone could land and take-off repeatedly without it. This edge represents the user's initial choice of locations for the drone to visit. It does not require any synchronization because the drone is at rest and the only possible actions at the next time-step are to initiate flight.

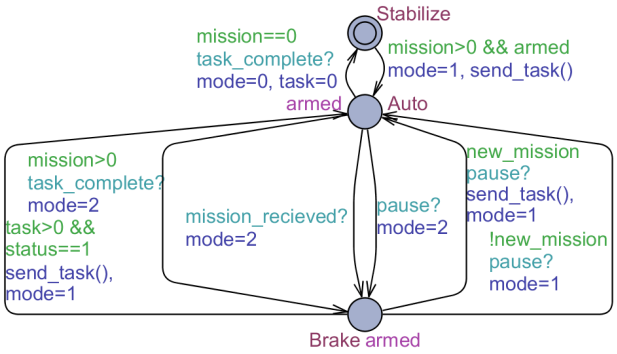
The bottom edge represents the user re-tasking the drone during flight, and the timing of this action is controlled by the guard checking if the current location is the same as the chosen location for re-tasking. In this case, synchronization is needed to interrupt the flight



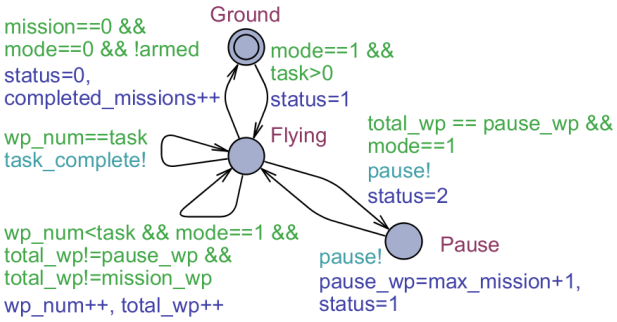
(a) Human interaction model



(b) Arming/Motor status model



(c) Drone mode model



(d) GCS tracking flight status

Figure 3.7: UPPAAL model of the interactions between the drone, GCS, and user, split into four subsystems.

and ensure that the drone responds to the new mission as quickly as possible.

Arming/Motor Status

Figure 3.7b shows the state machine for the arming status of the drone. The drone starts with the motors *disarmed*, and the GCS can tell the drone to switch to *armed* if two conditions are satisfied. First, the drone must be in the *stabilize* mode, since the drone's operating system will not allow the motors to arm in the other states. Second, the GCS will only arm the motors if the user has chosen a mission since arming the motors without one is a waste of energy and potential safety hazard. The GCS can only tell the drone to return to *disarmed* if the mission is complete and the mode has returned to *stabilize*. This ensures that the drone is on the ground again after completing its mission, preventing the motors from turning off during flight, which would lead to a crash.

Drone Mode

Figure 3.7c shows the model for the mode of the drone. The most critical part of this model is that the drone will not start flying to waypoints until it changes from a mode that is not *auto* to *auto*. Because of this, every transition into *auto* calls the "send_mission" function, which models sending a mission to the drone, activating it, and updates related variables.

The drone starts in *stabilize* when turned on, and the GCS will only change the mode to *auto* if the drone is *armed* and the user has specified a mission. These guards are in place for the same safety and efficiency reasons as the guards required for arming the motors. There are several possible transitions between *auto*, where the drone flies autonomously between waypoints and stops at the last, and *brake*, where the drone stops and hovers. These transitions can be grouped into normal flight, dealing with a requested pause, and dealing with a requested re-tasking. Another thing of note is that the *auto* and *brake* states both have the invariant *armed*, meaning the motors must be on.

The two leftmost edges deal with normal flight. The transition from *auto* to *brake* is triggered when the drone reaches the end of the current task if there are still points remaining for the drone to visit. From there, the drone is set back to *auto* to send and activate the next task.

The third edge from the left is triggered when the user chooses a new mission during flight. In this case, the drone is set back into *auto* as in normal flight but the new task comes from the new mission.

The three rightmost edges handle the user requesting a pause. Triggering a pause puts the drone into *brake* so it stops moving. From here, the user can resume the current mission, represented by the rightmost edge, or choose and new mission and then resume. If a new mission is chosen during a pause, the transition back into *auto* includes sending and activating the first task in that mission.

Flight Status

As mentioned previously, the GCS uses data from the drone to keep track of its flight status, shown in Figure 3.7d. The drone begins on the ground, and the GCS considers the drone to be *flying* once in *auto* mode and it has a task to follow. During flight, the drone notifies the GCS ever time it reaches a waypoint. This is modeled by the bottom-left edge incrementing both the index within the task and the overall mission. For modeling purposes, this edge can only trigger when the drone is not at the waypoint where the user requests a pause or re-tasking. When the drone reaches an event location in the simulation, the event is handled and it's trigger location is set to outside the range of the mission so the drone can continue with these guards no longer affecting the mission.

One of two things can happen when the drone finishes it's current task. If there are still waypoints remaining in the mission the left edge is triggered, resulting in the GCS sending

and activating the next task as described in the mode section. If the completed task was the final task, the GCS transitions back to the *ground* state once the mode and arming machines transition to *stabilize* and *disarmed*.

When the user requests a pause, the GCS transitions into a paused state, and returns to flight when the user terminates the pause. These transitions are synchronized with transitions to and from *auto* and *brake* to make the drone stop, resume flight, and set a new mission if given.

Validation

UPPAAL's built-in simulator and verifier are used to debug and validate the system model. This is done by providing the verifier with a query, which UPPAAL then evaluates. Four queries were used to validate this model, all relating to ensuring the drone actually does what the user requests. In the language used for the verifier, $A[]$ means invariantly or always, and $A <>$ means eventually.

Four queries were used to validate this model, all relating to ensuring the drone actually does what the user requests. In the language used for the verifier, $A[]$ means invariantly or always, and $A <>$ means eventually. The verifier queries used are listed below. (1) ensures that the drone has visited all waypoints when it lands. (2) checks that the model is only in deadlock when on the ground after completing a mission, again ensuring that the mission is completed by making sure the drone never gets stuck. (3) and (4) check to see if the pause and re-tasking events take place within the mission, and if they do, ensures that they happen at the correct time. All of these queries were valid, meaning the model met these requirements.

(1) A[] ((status==0 && completed_missions>0) imply total_wp==

total_mission)

(2) A[] (not(status==0 && completed_missions>0) imply
not deadlock)

(3) A<> ((pause_wp <= total_mission) imply
(flight.Pause && total_wp==pause_wp))

(4) A<> ((mission_wp <= total_mission) imply
(new_mission && total_wp==mission_wp))

The verification exposed some short-comings and edge cases in the high-level control algorithm. The verifier and simulator made finding and debugging these problems faster and easier than doing so in code, making the final interface more robust once these changes were implemented.

3.3.5 Video Streaming

Both drones streamed video over WiFi to the base station. This was handled by sending individual images compressed to a jpg through ROS messages, which were then displayed in separate windows for the user. The video stream from the HAS was also used to to update the map used for drone tasking with a current aerial view. ROS is designed to run on a single computer, but can be modified to run on multiple systems connected by WiFi. Using the ROS functionality to stream the images was chosen because when the computers clocks are synchronized, the ROS time is the same on both machines and can be used to match images to odometry data for use in rectification, as described in 3.3.7. If the system was expanded to use more drones, the traffic would eventually use all of the available bandwidth in the network, meaning the videos would need to have lower frame-rates or resolution. ROS does

introduce some overhead in the communication so finding another communication method would help, but not eliminate this problem.

3.3.6 Path Planning

The path planners used for the HAS and LAI were designed based on the assumptions in 3.1, with the most relevant for this section being static obstacles and no obstacles at survey height. Because of this, the path planner for HAS was extremely simple. The search area selected by the user is represented by a rectangle with latitude and longitude coordinate pairs representing two opposing corners. These coordinates are converted into position in meters. From there, the camera field of view is used to calculate the height and location required for the HAS camera to capture the entire area. A slight buffer is added to the height to ensure the search area is always in view.

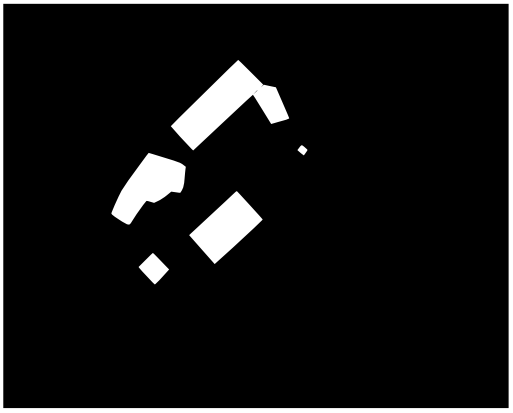
The LAI path planner converts the POI sent by the user from world coordinates into pixel coordinates on the obstacle map, shown in Figure 3.8a. The obstacle grid is down-sampled to reduce computation time, and was set to 8 foot grid squares for the testing environment. Due to uncertainty in the drone's position from GPS errors and weather, a buffer is added to the obstacle map, shown in Figure 3.8b. The obstacle map includes an area marked as obstacle that is not a building but reserved for parked cars. These pixel coordinates are then processed by a wavefront path planner based on code written by Hunter Ray, a PhD student at CU Boulder. This path planner chooses the next POI to visit based on lowest linear distance, ignoring obstacles, and uses a wavefront planning algorithm to find a path for the drone between those points, the results of which can be seen in 3.8c. The wavefront planning algorithm (2.5.2) was chosen due to its low run duration and the small size of the map, and the Nearest Neighbor TSP solution (2.5.1) was chosen for speed and simplicity.

The most likely problem to arise from these choices would be if POI were located on either side of a narrow corridor. This could lead to a zig-zagging path when a sweep along each side of the corridor individually might be more desire-able. Chourey [6] solved this problem by calculating contours around objects and biased the path planner to favor travelling along these contours. This solution was not implemented for this system because the main focus was on concurrent flight and re-tasking the LAI.

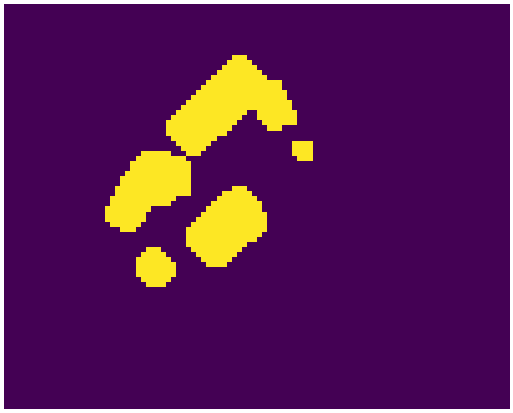
Despite down-sampling the obstacle grid, the calculated paths can still contain a large number of waypoints. To reduce the amount of waypoints uploaded to the drone the final step in LAI path planning is removing waypoints within a certain distance of each other along a straight line. Not all waypoints on the straight line are removed to allow the planner to add yaw commands along the way. The effect of this can be seen in the irregular spacing of the waypoints in Figure 3.8c.

There is a possibility that the user selects POI located inside obstacles. To ensure the LAI can investigate all POI, the path planner moves all points inside objects to the edge of the object. This is accomplished by finding the edges of all obstacles in the obstacle map, finding the closest edge to the POI by minimizing squared distance, and moving the POI just to the other side of the edge.

Both path planners assign headings for the drone to follow as it flies along the path. For the HAS, this is always set to north, but the LAI's headings always looks towards the nearest obstacle. The nearest edge of an obstacle to the point on the path is found the same way as above, and the system calculates the yaw required for the drone to look at this obstacle for each point in the path. Previously, a pilot controlled the yaw of the aircraft, and this was added to further automate the search. However, this does mean that the drone may not always look where the user intends to search. This can be mostly overcome but choosing the POI with this in mind.



(a) High resolution obstacle grid with white as obstacles and black as free space.



(b) Down-sampled obstacle grid with a buffer around obstacle, with yellow as obstacles and purple as free space.

(c) Drone (green dot) path, where red dots are the waypoints and blue dots are POI.

Figure 3.8: Various stages of the LAI path planner for visiting four POI. The area next to the uppermost building is reserved for parked cars on the obstacle maps.

3.3.7 Map Creation

As mentioned previously, the HAS supplies the user interface with an updated image of the search area, but this image is first processed by the GCS to rectify it with the Earth's surface and georeference it. The image is rectified using the CameraTransform Python package and the drone's roll, pitch, and yaw. Global position is used as the extrinsic properties of the camera and the Picam v2 properties are used as the intrinsic properties, as explained in section 2.6. Once rectified, CameraTransform is used to convert 3 pixel coordinates in the image to world coordinates, and Rasterio is used to georeference the image based on these points. Both the picture and HAS position data are timestamped, so the position data recorded closest to when the picture was taken is used.

3.3.8 Probabilistic Search

As discussed in the Introduction (1), probabilistic techniques could be used to aid the user in conducting a search. This section discusses how the preference elicitation algorithm created by Ray et al. [16] and a simpler visual overlay could be implemented into this system. Both of these solutions require a segmented environment, but segmenting 2D imagery is difficult, as discussed in Section 2.4. To demonstrate how these probabilistic algorithms could be used, a 3D reconstruction was used to segment buildings from the rest of the environment based on height. The 3D reconstruction was made using geotagged survey images and OpenDroneMap. From there, OpenCV python functions were used to segment roads and process the results. Roads were segmented from 2D imagery using the HSV (hue, saturation, value) color model. This segmentation detected parts of buildings as well as the roads, so the building segmentation result was used to remove the building from road segmentation. This yielded a mask with a lot of noise, false positives from objects with similar HSV values,

and false negatives from discolorations in the road. A series of eroding the image, or taking away outer parts of objects, and dilating, or expanding the outer part of an object, helped fix these problems without warping the shape of the road. Eroding followed by inflating removed noise and small false positives like hay bales, and then inflating followed by eroding closed holes in the road segmentation. The final results of segmenting roads and buildings are shown in Figure 3.9. While eroding and dilating the road segmentation improved the results, there are still some false positives and negatives.

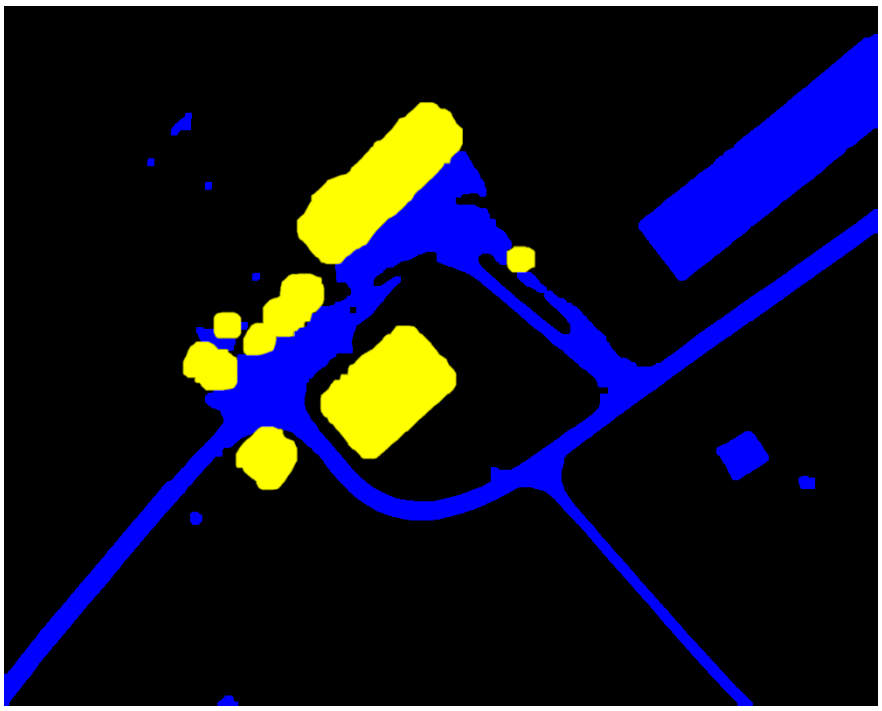


Figure 3.9: Segmented environment of Kentland Farms. Roads are blue, buildings are yellow, and everything else is black.

Using the segmentation results as masks, intersections between roads and buildings were found by dilating, or expanding, each and finding where the resulting masks intersected. These intersection areas were then highlighted on the map in red, shown in Figure 3.10.

Integrating the preference elicitation algorithm into the system would require changes to the GUI to allow the user to interact with the algorithm. An example of this based on the

Figure 3.10: Region of interest overlay, showing intersections between buildings and roads highlighted in red on a map to help the user choose POI for investigation.

GUI for the existing system is shown in Figure 3.11. In this example, the functions of the "Add Waypoint," "Delete Waypoint," "Pause Mission," "Investigate," and "Map" buttons would remain the same. To teach the algorithm the user's POI preference, the user would add some POI and get the algorithm's initial guess of preference by clicking the "Suggest Points" button. From there, the user could update the suggestion by adding and deleting POI and click "Propagate Points" to have the algorithm mark POI across the entire search region. Finally, the user could add or remove points again and click "Investigate" to start the mission.

Figure 3.11: Example of a GUI allowing the user to interact with the preference elicitation algorithm.

Chapter 4

Testing

4.1 Test Environment

The system was tested at the Kentland Experimental Aerial Systems (KEAS) Laboratory, located at the Virginia Tech Kentland Farm agricultural research facility. The area has several buildings, some farm equipment, a field, and a small runway. As mentioned in section 3.1, previous survey data was used to create an obstacle map in the form of a grid. In addition to the buildings present, a section of the grid often used for parking cars was also marked as an obstacle.

4.2 Risk Management and Safety

As described in Vierhauser et al. [20], operating UAV systems creates a variety of potential hazards. While the methods discussed by Vierhauser et al. were not used here, several steps were taken to ensure the system operated safely, which can be broken down into operations and system safeguards.

USL has implemented a tracking system to help ensure safe drone operation, which covers both the drones and pilots. Pilots are FAA part 107 and TOP level 2 (Trusted Operator Program, an industry standard) certified. Both of these certifications require a pilot to pass

a knowledge exam, and the TOP level 2 certification also includes a flight test. In addition to the certifications described above, all crew members, including pilots and other crew, must pass a quiz on basic drone operation and the USL tracking system, which is described in detail below.

For each flight, a pilot must fill out a flight request form with information about the location, weather, goal, risks, and consequences of the flight as well as the pilot and aircraft. All identified risks must have mitigation techniques that are implemented, and if a risk has too high of a consequence or severity extra precautions must be discussed with the chief pilot. The mission's overall risk level is evaluated using a risk matrix, shown in Figure 4.1, by matching the rows and columns to the risk severity and likelihood. This is done first without and then with risk mitigation techniques. If the risk level is orange after mitigation, the risks must be discussed with the chief pilot before the flight is approved. A risk level in the red is deemed unacceptable and is not allowed to fly.

Risk Likelihood	Risk Severity				
	Catastrophic A	Critical B	Moderate C	Minor D	Negligible E
5 - Frequent	5A	5B	5C	5D	5E
4 – Likely	4A	4B	4C	4D	4E
3 – Occasional	3A	3B	3C	3D	3E
2 – Seldom	2A	2B	2C	2D	2E
1 – Improbable	1A	1B	1C	1D	1E

Figure 4.1: Risk matrix used to determine overall mission risk level.

Once the flight is approved by the Flight Operations Manager or Chief Pilot, the pilot will receive an approval code. Using this code, the pilot must fill out a pre-flight form that requires the same information as the flight approval, but also includes an inspection

checklist. This checklist is filled out using the ”point and call” method, in which the item being inspected must be pointed to, helping to ensure proper inspection [18].

While at the test site, a few more rules are followed to ensure a safe testing environment. The crew member in charge of the test must give a briefing discussing the mission plan, risks, and crew member roles. For this system, clarifying the pilot roles was vital because two drones were flown at once and confusing which drone a pilot is controlling would almost certainly lead to a crash. During flight, pilots must announce important flight actions, like arming and takeoff. Since the drones are controlled by the computer for this system, the computer operator called out actions so the pilots were always aware of what should happen next.

Once the flight is complete, the pilot must fill out a post-flight form listing the results of the flight and any incidents that occurred. The post-flight form contains resources to help determine what qualifies as an incident or accident, and the quiz all crew members take also ensures everyone knows what qualifies as an incident or accident and what to do for each. All of these steps are in place to keep a detailed log of each pilot's and aircraft's activities and ensure all flight operations a USL are as safe as possible.

In addition to the operations level safety steps above, the system described in this paper has some additional steps specific to the aircraft used and the human interaction. The aircraft have an onboard ”hardware safety switch” that pilots must press before the drone can be armed, ensuring that the pilots are ready for the flight. In addition to this, the autopilot conducts several internal pre-am checks that must pass for before the drone can arm, ensuring that the system will respond properly in flight. Once the hardware switch is pressed and pre-arm checks are complete, the drone can be armed. This can be done automatically on the GCS, but this system is a prototype so the pilots were required to arm the drones manually. Similarly, the GCS could change the autopilot mode to auto, starting

the mission, but the pilots did so manually. Both of these steps were implemented manually instead of automatically on the GCS to ensure the pilots were ready for the mission. Pilots were additionally required to maintain visual line of sight with the drone at all times and keep the throttle on their transmitters on so they could take over control of the drone if the system failed in any way.

Within the system software, some parameters on the drone and GUI characteristics were chosen to mitigate risks. The drones were set to fly slowly, giving pilots more time to react if collision seemed imminent, and the RTL altitude was set to be well above the buildings in the area. To ensure that the GUI was easy to read, the markers for POI and the drone's location are shown in red and pink, which were chosen to contrast with the grass in the map. The color of the markers in the GUI is easily changed, and might need to be updated depending on the environment and user color-blindness. As a final safety measure, the system was tested in simulation, then over the runway and field at Kentland Farms to ensure everything worked as expected without the risk of colliding with buildings or cars.

4.3 Test Procedure

The objective was to ensure that all user commands were properly carried out. Once all safety checks are complete, the user conducts a search mission. An example of the plan for such a mission is shown in Table 4.1. For each test, communication between the drones and GCS was verified manually via ROS commands as a safety precaution. Once complete, the pilots pre-armed the drones so the drones could fly. The mission started when the user selected a search area on the GUI, pressed "Map," and waited for the HAS to reach its target position. Once the GUI was updated with current imagery, the user selected a few points for the LAI to investigate and clicked "Investigate." At some point during the LAI flight before

the all POI were visited, the LAI was paused and re-tasked. Once the mission was complete, the "Land" button was pressed and both drones returned to their takeoff locations.

Table 4.1: Mission plan for two-drone search at Kentland Farm.

Step Number	Action
1	Select search area for HAS and start HAS mission
2	Mark POI on updated map and start LAI mission
3	Pause LAI
4	Add/delete LAI POI
5	Resume LAI with new mission
6	Terminate mission

4.4 Results

Table 4.2 shows the results of a mission with timestamps. These tables show that the drones responded appropriately to user commands throughout the mission. Figure 4.2 shows the GUI during flight, with the video from the drones, the location of the LAI, and POI visible.

Table 4.2: Mission timeline for two-drone search at Kentland Farm.

Mission Time (s)	Event
0	Mission starts
23	Search area selected and HAS takeoff
50	HAS reaches target and GUI is updated 10 seconds later
90	POI chosen, planned path section uploaded to LAI, LAI takeoff
128	Next path section uploaded to LAI
154	LAI paused
190	LAI resumed with new POI and path
226	RTL commanded
270	LAI landed
285	HAS landed, mission terminated

The path the drones took is shown in Figure 4.3, including the POI marked by the user. The path is split into two sections, with the first showing the HAS travelling to the survey location

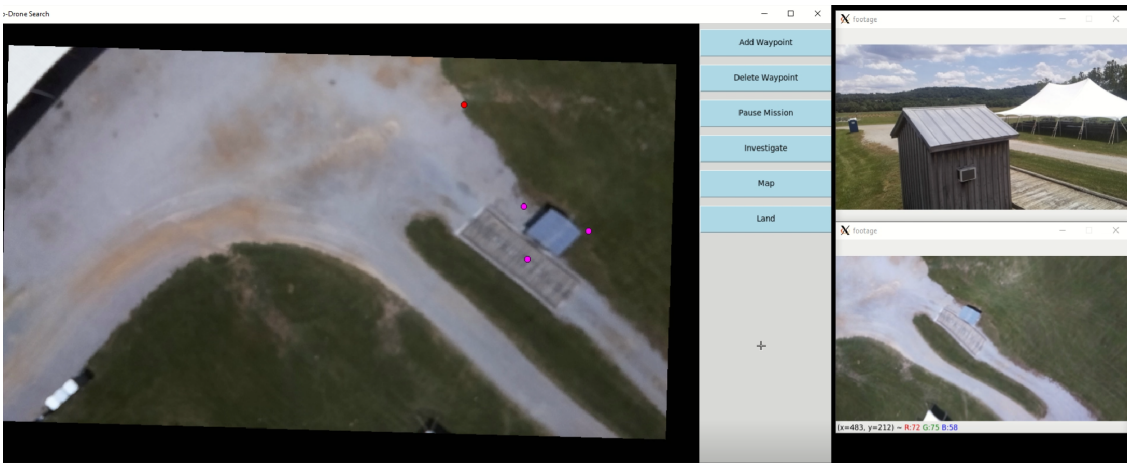


Figure 4.2: GUI during a mission, showing the live video from both drones, the POI, and the location of the LAI

and the LAI investigating the initial POI, and the second showing the LAI investigating the second set of POI. The times each drone reached the positions is shown in Table 4.3.

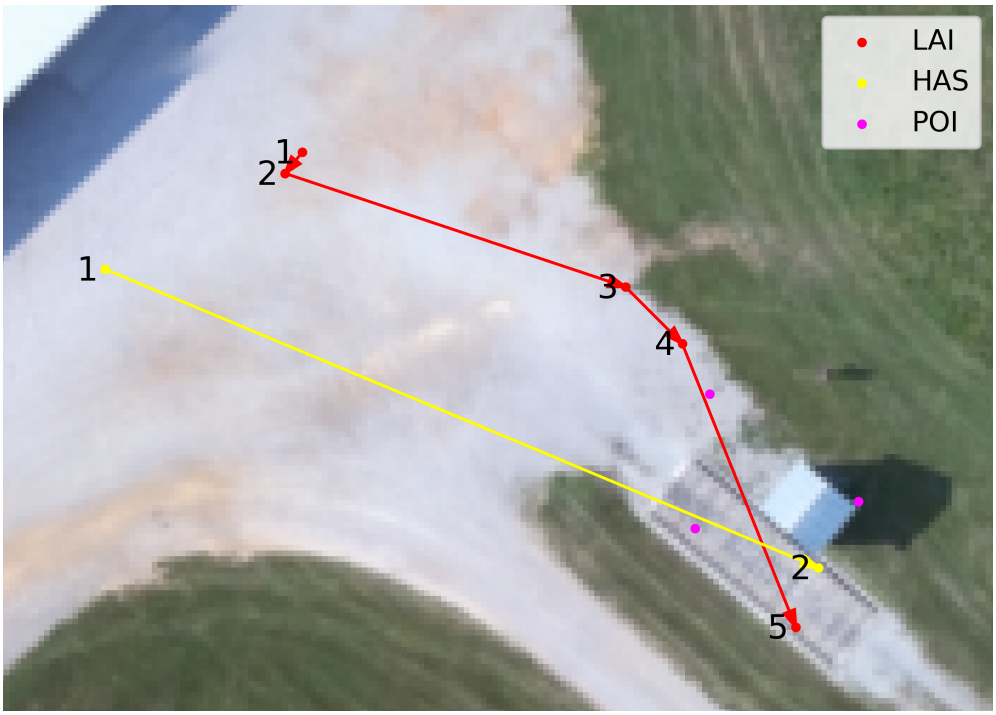
Table 4.3: Times the drones arrived at waypoints during the mission.

Waypoint	1	2	3	4	5	6	7	8
LAI Time (s)	0	102	119	124	153	195	202	215
HAS Time (s)	0	51						

Another test was completed on the same day, with the addition of a re-tasking without pausing the LAI. During this test, the HAS returned to launch prematurely, but examining the flight data showed that this was due to a connection error with the pilot's transmitter triggering a fail-safe and not due to the GCS or drone malfunctioning. In both tests, all user commands were carried out successfully. Re-tasking the LAI mid-flight allowed the user to abandon POI deemed unimportant and revisit previous locations that required review.

The area experienced relatively high and gusting winds during the test shown in the tables and figures above, which had three major effects on the results despite the wind being within the drones' operating capabilities. First, the size of the safety buffer around objects was set to 16 feet to ensure the LAI did not collide with buildings or cars. This resulted in the LAI

not actually visiting the POI marked on the map since those POI were inside the buffer, but the LAI still visited the points adjusted for safety. Second, the HAS was not steady due to the wind gusts. This motion warped the high-altitude video slightly due to the Picam's rolling shutter and made the georeferencing of the updated GUI image less accurate. Another problem that would likely occur with the LAI even without wind was unnecessary turning due to discrepancies between commanded and actual yaw angle. The drone must be told what direction to rotate to reach a commanded yaw, and the direction must be predetermined so it can be uploaded along with the rest of the mission. The GCS uses the previous and yaw command to determine the direction to rotate for each yaw, so when a small angle change is requested and the drone is not at exactly the previously commanded yaw it will sometimes spin almost 360 degrees instead of a few degrees to reach the target angle. A minimum angle change requirement was added to limit this, but the minimum could not prevent required heading adjustments so it could not be very large. To the user, this behavior can be seen in the LAI footage when the camera rotates away from obstacles, but the drone spins back to look at the object quickly so there is not a very large effect on the mission.



(a) Waypoints and POI for the initial tasking.

(b) Waypoints and POI after the user re-tasked the LAI.

Figure 4.3: The waypoints for each drone and the POI marked by the user. The times the drones arrived at the points are shown in Table 4.3.

Chapter 5

Summary and Conclusions

This thesis discussed the design of a system allowing an operator to search a partially known environment with two drones, continuing previous work at the Virginia Tech Unmanned Systems Lab. This work allows a user to search an area with a high altitude survey drone and a low altitude investigator drone. This collaboration between a human and drones utilizes the operator's perception and intuition to ensure a search mission is exhaustive and relevant while automating the drone flight to reduce workload and increase efficiency. The previous work surveyed and then investigated the search area in two separate missions and used a 3D reconstruction to segment buildings from the environment between the missions. The new system discussed here instead uses a preexisting obstacle map and allows the survey and investigating drones to fly at the same time. This allows the user to have a closer and updated aerial view of a section of the search area, and use that updated information to task the investigating drone. In addition to flying both drones at once, the user can pause and re-task the investigator at any time, or even re-task without pausing. All of the commands from the operator are entered through a GUI that first shows the entire operating area, then updates with current imagery from the high altitude drone. The operator can view live video from both drones throughout the mission. The low altitude drone is programmed to always look at the nearest obstacle, removing the need for the user to control the drone's view.

Before testing the physical system, the logic behind the interactions was modeled as a finite state machine in UPPAAL. This required some assumptions to be made, but validating

the model showed some edge cases that may not have been found otherwise, making the final product more robust. This system was tested with adherence to the USL safety and tracking procedures to minimize risks. These procedures, in addition to safeguards in the system design, ensured that the testing was safe and proceeded without any notable incidents. The testing showed that the system responds as expected to user commands from the GUI and allowed a user to search around cars and a building at the KEAS lab at Kentland Farm, including re-tasking the drone using updated information. Testing also exposed some shortcomings, mainly inaccuracy in georeferencing survey imagery and discrepancies between planned and actual yaw angles causing the drone to spin more than necessary. Neither of these problems inhibited the success of the search mission and could be improved in future work.

The system uses ROS to communicate between different programs and communicated with the drones via the MAVROS package, which uses the MAVLink messaging protocol. This ROS based architecture allowed the system design to be modular, making future development on the system and adapting to other uses very simple. The drones are currently equipped with cameras for the search, but adding other sensors like thermal cameras, radiation detectors, or microphones would allow the system to be used for other kinds of search missions, using the modular architecture to speed integration. Additionally, this system could be expanded to allow for more than two drones by adding an additional path planning step for swarm control. Currently, path planning for the low altitude drone was done using a nearest neighbor solution to the traveling salesman problem and a wavefront path planner to create a path between points, which could be used for individual drones in a swarm or be replaced entirely.

While not implemented on the system, this work also discusses probabilistic methods to aid the user in planning a search. The first method was created by a team at the University

of Colorado Boulder, using a partially observable Markov decision process to learn user preference for points of interest. The second method uses simple computer vision techniques to find were buildings and roads are close to each other, and highlights these regions on the GUI for the user. In both cases, computer vision is needed to segment the environment. This is currently done using a 3D reconstruction, but that takes a long time so it cannot be done live during a mission. Future work could look into solving this through more advanced computer vision techniques to segment using only 2D imagery, or find a way to create a 3D reconstruction faster. In addition to enabling probabilistic mission planning aids, a fast way to segment buildings from the environment would remove the need for a predetermined obstacle map, allowing the system to fly anywhere.

Similarly expanding the capabilities of the system, future work could also focus on on-line image mosaicing. Existing methods were explored by Chourey [6], but were all found to be too slow or inaccurate. Improving this would allow the user to select a search region that is not limited to a single camera frame, and allow the investigation flight to start once only a part of the region is surveyed instead of waiting for the entire survey. The user could then use the existing re-tasking process to expand the investigation and the survey is completed.

Bibliography

[1] Intelligent Drone Swarm for Search and Rescue Operations at Sea, November 2018. URL http://arxiv.org/abs/1811.05291. Number: arXiv:1811.05291 arXiv:1811.05291 [cs].

[2] Ankit Agrawal, Jane Cleland-Huang, and Jan-Philipp Steghöfer. Model-Driven Requirements for Humans-on-the-Loop Multi-UAV Missions. In *2020 IEEE Tenth International Model-Driven Requirements Engineering (MoDRE)*, pages 1–10, August 2020. doi: 10.1109/MoDRE51215.2020.00007.

[3] Ebtehal Turki Alotaibi, Shahad Saleh Alqefari, and Anis Koubaa. LSAR: Multi-UAV Collaboration for Search and Rescue Missions. *IEEE Access*, 7:55817–55832, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2912306. Conference Name: IEEE Access.

[4] Ross Arnold, Jonathan Jablonski, Benjamin Abruzzo, and Elizabeth Mezzacappa. Heterogeneous UAV Multi-Role Swarming Behaviors for Search and Rescue. In *2020 IEEE Conference on Cognitive and Computational Aspects of Situation Management (CogSIMA)*, pages 122–128, August 2020. doi: 10.1109/CogSIMA49017.2020.9215994. ISSN: 2379-1675.

[5] Giuseppe Bevacqua, Jonathan Cacace, Alberto Finzi, and Vincenzo Lippiello. Mixed-Initiative Planning and Execution for Multiple Drones in Search and Rescue Missions. *Proceedings of the International Conference on Automated Planning and Scheduling*, 25:315–323, April 2015. ISSN 2334-0843. URL https://ojs.aaai.org/index.php/ICAPS/article/view/13700.

[6] Shivam Chourey. *Cooperative human-robot search in a partially-known environ-*

ment using multiple UAVs. Thesis, Virginia Tech, August 2020. URL https://vtechworks.lib.vt.edu/handle/10919/99870. Accepted: 2020-08-29T08:00:27Z Artwork Medium: ETD Interview Medium: ETD.

[7] Jane Cleland-Huang and Ankit Agrawal. Human-Drone Interactions with Semi-Autonomous Cohorts of Collaborating Drones. Technical Report arXiv:2010.04101, arXiv, October 2020. URL http://arxiv.org/abs/2010.04101. arXiv:2010.04101 [cs] type: article.

[8] Vladimir Ermakov. mavros - ROS Wiki, March 2018. URL http://wiki.ros.org/mavros.

[9] ArcGIS Esri. Coordinate systems, projections, and transformations—ArcGIS Pro | Documentation. URL https://pro.arcgis.com/en/pro-app/2.8/help/mapping/properties/coordinate-systems-and-projections.htm.

[10] Richard C. Gerum, Sebastian Richter, Alexander Winterl, Christoph Mark, Ben Fabry, Céline Le Bohec, and Daniel P. Zitterbart. CameraTransform: A Python package for perspective corrections and image mapping. *SoftwareX*, 10:100333, July 2019. ISSN 2352-7110. doi: 10.1016/j.softx.2019.100333. URL https://www.sciencedirect.com/science/article/pii/S2352711019302018.

[11] Sanchit Goyal. A Survey on Travelling Salesman Problem. page 10, January 2010.

[12] Karin Anna Hummel, Manuela Pollak, and Johannes Krahofer. A Distributed Architecture for Human-Drone Teaming: Timing Challenges and Interaction Opportunities. *Sensors*, 19(6):1379, January 2019. ISSN 1424-8220. doi: 10.3390/s19061379. URL https://www.mdpi.com/1424-8220/19/6/1379. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

[13] Ali Maher, Ce Li, Hanwen Hu, and Baochang Zhang. Realtime Human-UAV Interaction Using Deep Learning. In Jie Zhou, Yunhong Wang, Zhenan Sun, Yong Xu, Linlin Shen, Jianjiang Feng, Shiguang Shan, Yu Qiao, Zhenhua Guo, and Shiqi Yu, editors, *Biometric Recognition*, Lecture Notes in Computer Science, pages 511–519, Cham, 2017. Springer International Publishing. ISBN 978-3-319-69923-3. doi: 10.1007/978-3-319-69923-3_55.

[14] MapBox. Rasterio: access to geospatial raster data — rasterio documentation, 2018. URL https://rasterio.readthedocs.io/en/latest/index.html.

[15] Sujit Rajappa, Heinrich Bülthoff, and Paolo Stegagno. Design and implementation of a novel architecture for physical human-UAV interaction. *The International Journal of Robotics Research*, 36(5-7):800–819, June 2017. ISSN 0278-3649. doi: 10.1177/0278364917708038. URL https://doi.org/10.1177/0278364917708038. Publisher: SAGE Publications Ltd STM.

[16] Hunter M. Ray, Nicholas Conlon, Zachary Sunberg, and Nisar R. Ahmed. User Preference Elicitation for Unmanned Aircraft System Collaborative Search. In *AIAA SCITECH 2022 Forum*, AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, December 2021. doi: 10.2514/6.2022-2343. URL https://arc-aiaa-org.ezproxy.lib.vt.edu/doi/10.2514/6.2022-2343.

[17] Open Robotics. ROS: The ROS Ecosystem, 2021. URL https://www.ros.org/blog/ecosystem/.

[18] Kazumitsu Shinohara, Hiroshi Naito, Yuko Matsui, and Masaru Hikono. The effects of "finger pointing and calling" on cognitive control processes in the task-switching paradigm. *International Journal of Industrial Ergonomics*, 43(2):129–136, March 2013. ISSN 0169-8141. doi: 10.1016/j.ergon.2012.08.004. URL https://www.sciencedirect.com/science/article/pii/S0169814112000728.

[19] Dante Tezza and Marvin Andujar. The State-of-the-Art of Human–Drone Interaction: A Survey. *IEEE Access*, 7:167438–167454, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2953900. Conference Name: IEEE Access.

[20] Michael Vierhauser, Md Nafee Al Islam, Ankit Agrawal, Jane Cleland-Huang, and James Mason. Hazard analysis for human-on-the-loop interactions in sUAS systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 8–19, Athens Greece, August 2021. ACM. ISBN 978-1-4503-8562-6. doi: 10.1145/3468264.3468534. URL https://dl.acm.org/doi/10.1145/3468264.3468534.

[21] Jeffrey Whitaker. pyproj Documentation — pyproj 3.3.1 documentation, 2022. URL https://pyproj4.github.io/pyproj/stable/.

[22] Issa Zidane and Khalil Ibrahim. Wavefront and A-Star Algorithms for Mobile Robot Path Planning. pages 69–80, September 2018. ISBN 978-3-319-64860-6. doi: 10.1007/978-3-319-64861-3_7.