Final Report: Automatic Target Recognition and Map Generation

Team Name: Target Acquired

Ryan Ylagan - Technical Co-Lead

- Problem Statement and Objective
- Introduction and Background
- Final Design Selection
- Analysis Plan and Results

Justin Campbell - Project Lead

- Feasibility Studies
- Final Design Selection
- Analysis Plan and Results

Rohan Wariyar - Communications Co-Lead

- Conclusion and Future Work
- Global, Environmental, and Societal Impacts

Preston Hart - Communications Co-Lead

- Introduction and Background
- Feasibility Studies
- Final Design Selection
- Analysis Plan and Results

Nicholas Aufiero – Technical Co-Lead

- Introduction and Background
- Feasibility Studies
- Final Design Selection
- Analysis Plan and Results

Problem Statement

Contributions: Ryan

The Austin metropolitan area has experienced flooding. The rising water levels have become so significant that the Austin Fire Department (AFD) has begun to receive distress calls from individuals stranded on the roofs of their houses trying to escape the rising waters. In order to be more efficient with their search and rescue efforts, AFD needs to be able to know exactly if they need to deploy their rescue teams and where to deploy them. To do so, the AFD wants to install an unmanned aircraft system that is equipped to provide 24/7 search, surveillance, and precision airborne delivery to aid in rescue operations.

In the water, the houses with individuals atop them create targets of interest (TOI). Some individuals may be completely uninjured and in a safe enough position to wait for the water to subside. On the other hand, some individuals may be in need of immediate medical attention. The location of these homes, an assessment of the distress, and any other information gathered from the surveillance should be transmitted to AFD

Our mission will be a joint effort with the Aerospace senior design team (ASE) and the Computational senior design team in charge of the aerial drop payload. The ASE team will be in charge of assembling an aircraft and programming its flight path for the search and surveillance stage of the mission. The aircraft will perform this stage over an approximately 300,000 ft^2 mowed field with the TOIs represented by five 2 x 2 meter targets made with a large blue tarp. Atop two of these TOIs are smaller objects that mark these targets as candidate targets for potential rescue; one with a “smiley face” representing an uninjured person, and another with a “frowny face” signifying an individual in need of immediate assistance.

Problem Objectives

Contributions: Ryan

Our main objective is to create a piece of automatic target recognition software that is capable of categorizing our aforementioned TOIs as a candidate target and furthermore be able to classify the type of assistance needed based on the face that lies upon it. We also need to return accurate GPS coordinates of each TOI that we can return to the aerial drop payload team. This information then needs to be compiled together along with images taken from our mounted camera to generate a map of the targets and their classification. The speed at which our software is able to acquire and identify these targets as well as how accurate our GPS locations are to the true location of the targets will be used to score the ASE team’s mission.

Some of our objectives that are not required but may be good to include as stretch objectives are:

- An interactive map that is paired with the respective GPS coordinates
- Image recognition software that can identify human emotion rather than just our preset TOIs
- A more efficient algorithm that improves accuracy and required search time
- An algorithm that is capable of identifying targets on more complex terrain rather than just an open field

Introduction and Background

Contributions: Rohan, Preston, Ryan, Nicholas

Every year, the aerospace senior design teams are tasked to perform a demonstration mission that is to be conducted at the aforementioned 300,000 ft^2 fields. This mission includes a few important steps that include takeoff and deployment of the aircraft, which will be handled by the aerospace design team. Our involvement concerns the search and surveillance steps of the mission that occur after the aircraft reaches the cruising altitude of between 200 to 300 ft. During this phase, our software should be able to automatically recognize the numerous targets laid out on the field as well as classify these targets based on their display. In previous years, this phase was done manually, but this is now done completely autonomously due to our involvement.

Overall our mission and its objectives are not necessarily a unique problem. There have been many well-documented case studies of other teams creating an unmanned aerial vehicle equipped with a mounted camera software for image recognition and map generation for the purpose of search and rescue. In one particular study, a Japanese team used a small UAV to generate a map of an area recently affected by an earthquake. This map detailed the surrounding area's geological features and marked points of interest that may be considered important for a search and rescue team, such as a collapsed house. Along with the map, the aircraft would return data such as position, altitude, and time that is specific to each individual photo.

Computer vision is an area of technology that a dense amount of research and development efforts are currently being poured into. The ability for equipment to visualize its surroundings, make sense of the environment that it is in and make smart decisions from such perceptions has been a major breakthrough in autonomous capabilities. Specifically, in the case of unmanned aerial vehicles, many military and search and rescue operations have been seeking out computer vision software to automate tasks that are time-consuming and error-prone for the human eye. There is numerous open-source computer vision software available for implementation and most differ in how they algorithmically learn the object(s) of interest and then proceed to evaluate and make decisions on new pieces of data. Both of these key components of image evaluation are integral parts of the runtime complexity that computer vision software are judged on and ultimately a piece of the process that went into evaluating the software discussed in this project proposal. As a note, there is a difference between locating and mapping targets and areas of interest. Today, more modern computer vision algorithms are seeking to combine the process of locating and mapping into one continuous process that uses minimal training data. This combined methodology is referred to as a SLAM process and is the architecture behind self-driving cars as well as autonomous ground-based robots. However other computer vision algorithms simply seek to find large localized differences in color contrasts in order to identify targets of interest. Although this category of computer vision algorithms leans toward being less computationally intensive than other methods, they are not always as practical since there is a large emphasis placed on color differences which could lead to difficulties when color is not the key factor of identification.

Because UAVs have become extremely popular over the past decade, the equipment required to run computer vision on the go has become widely available. Performing computer vision from up to 300 feet in the air requires a camera with enough resolution to accurately capture small ground targets. UAV computer vision also requires a small, light computer that can run independently but in communication with the main flight computer. This computer must be powerful enough to process the images acquired from the camera in real-time. With today's processors, that is possible.

Feasibility Studies

Contribution: Justin, Preston, Nicholas

Concerning the hardware of our project, we had to make decisions about what camera and what coprocessor we want operating within the UAV. The selection process for the camera will be discussed first. The team again began the Preliminary Design Stage by assembling a table with preliminary criteria for the camera that were determined to be the most important for the camera to satisfy the project's end objectives for both the COE and ASE design teams. The ASE team emphasized the importance of weight reduction on their aircraft as weight plays a large role in their final score. As such, the weight of the camera was an important consideration for us. Other important considerations were the operating resolution, the frame capture rate, the output interface terminal type, the camera's physical dimensions, and the available documentation. The three cameras that were considered in the preliminary design stage were the Topotek 10x zoom, the Lumenera Lt-C, and the RunCam Zoom. After a literature review on similar ATR projects, we found a team that had successfully performed ATR at altitudes of around 200 feet using a 5-megapixel camera. We used 5 Mpx as the minimum criteria for the operating resolution. Similarly, we set aside a frame capture rate of 20 fps as the minimum criteria according to documentation found in our literature review. With respect to the total weight and physical length dimensions criteria, values were set in collaboration with the ASE Design Team to operate within their weight and mass distribution constraints. This will ensure that the implementation of our hardware products does not impact the vehicular performance of the UAV. With respect to the other two, the team used a more subjective approach to determine the minimum criteria. For the type of interface output terminals category, the criteria were satisfied for a particular design if it contained at least one in any combination of USB, HDMI, or MIPI CSI output terminals. For the available documentation category, the criteria were satisfied if documentation was easily accessible, readable, and verified/validated.

After the preliminary criteria were assembled, each design was documented according to whether or not it satisfied the criteria. The team then set aside three critical criteria that were determined to be the most important for the camera to achieve its end objectives, namely, the operating resolution, frame capture rate, and type of interface output terminals. The cameras that satisfied a majority of the critical criteria were downselected to the Feasibility Analysis to be considered in the final design cut. Out of the four cameras considered in the preliminary design stage, only the RunCam Zoom did not meet a majority of the criteria. In particular, it neither met the operating resolution criteria nor the frame capture rate and was thus eliminated from consideration. After downselecting our preliminary design ideas, we assembled them into a weighted scoring matrix along with the criteria specified in the preliminary design phase as shown in Figure 1 below. This process marked the beginning of our Feasibility Analysis to obtain final designs from the downselection. Using a linear scoring methodology, the criteria were ranked from most to least important, and assigned numerical weight values in descending order of magnitude.

	Topotek 10x Zoom	Lumenera Lt-C/M2020	Arducam IMX477 HQ Camera Model	
Criteria Description	**Score**	**Score**	**Score**	**Linear Scoring Methodology Legend**
CR - Operating Resolution Range	0	6	6	GREAT >= 85% criteria weight satisfied
CR - Frame Capture Rate	5	5	5	GOOD >= 70% criteria weight satisfied
CR - Type of Interface Output Terminals	4	4	4	NEUTRAL >=50% criteria weight satisfied
CR - Available Documentation	0	3	3	POOR < 50% criteria weight satisfied
CR - Total Weight	2	2	2	
CR - Physical Length Dimensions	1	1	1	
Total Score	12	21	21	

Figure 1: Weighted Scoring Matrix for Camera Design in Feasibility Analysis

Total criteria scores for each downselected design were obtained by summing the contribution from each criterion according to whether or not they were satisfied. For example, the Topotek design did not meet the criteria for resolution range and was thus allocated a score of 0. However, it did meet the criteria for frame capture rate and was thus allocated the number of points corresponding to that criteria rank (since the rank is the second-highest on a list of 6 criteria, this meant 5 points). Using the metric shown for evaluating the "goodness" of a design according to the percentage of criteria weight satisfied, it was determined that the Lumenera Lt-C and the Arducam IMX were the most desirable options. The Topotek was safely removed from contention because it did not meet the resolution criteria, and is poorly documented. To weigh the two remaining cameras against one another, the Lumenera Lt-C was removed because it is restricted to taking still images instead of streaming video, and, after deliberation, the video became a hard requirement for the ASE team. As such, we pivoted to the Arducam camera, which meets all of our important requirements, including resolution, weight, and ease of use. As such, we will be moving forward with the Raspberry Pi Camera in our final design.

The ATR code will run on a coprocessor. This coprocessor must fit within the middle section of the UAV, it must meet stringent weight requirements so as to not significantly shift the center of gravity of the UAV, and it must be able to read video data from the camera as well as flight data from the Ardupilot. The coprocessor must also require no more than 5 volts to run, and it must also be able to effectively run the ATR algorithm within a reasonable time. Raspberry Pi's are a user-friendly, effective solution to edge computing. The model 4 is powerful enough to run our reasonable ATR algorithm. Most importantly, from the literature, it has been determined that many teams have been able to solve similar problems using the Raspberry Pi. We also looked at the Jetson TX2 co-processor. This coprocessor is a machine learning workhorse with 6 CPUs and a GPU. The Jetson might be required for highly complex ATR tasks, but it is overkill and too heavy for our project. In following the same procedure as above for the camera, namely, identifying critical criteria for co-processor designs, documenting designs according to the criteria that they satisfied, and down-selecting preliminary designs that met a majority of these criteria, the decision for the co-processor was easier. In particular, both the Raspberry Pi model 4, and the Jetson TX2 met a majority of the critical criteria in the preliminary design phase and were documented for Feasibility Analysis. In the Feasibility Analysis, it was determined that both the Jetson and Raspberry Pi were acceptable designs satisfying a considerable majority of the criteria weight. The differentiating factor between the two lies in the power supply voltage, and weight. In particular, the Jetson requires a much higher voltage supply and is much heavier than the Raspberry Pi. As such, we selected the Raspberry Pi model 4 as our final design.

Amongst many different ideas that were discovered through extensive research into the problem of automatic target recognition and aerial map generation, three different algorithmic software approaches were evaluated. OpenMV is another open-source computer vision software that was researched after coming across a published paper by a group of engineers that developed an unmanned aerial vehicle able to identify and track moving objects. OpenMV differs from OpenCV in the sense that it was developed as a “machine vision” solution to object detection rather than a “computer vision” solution. That is to say, where OpenCV requires a camera connection typically via USB to a co-processor, OpenMV integrates a small camera directly into a customized co-processor. The custom co-processor comes equipped with a prebuilt integrated development environment (IDE) for users to modify and write their own objection detection algorithms. The customized IDE and camera integrated co-processor alleviate some of the issues that could arise while interfacing hardware components. This open-source software is also advantageous in its low latency and minimal resolution loss when transferring images from the camera to storage on the co-processor. However, out of a desire to keep the co-processor as lightweight as possible, the camera only has a resolution of 640x480 Mp and a small field of view (FOV). The stock lens can be swapped out for better lenses, but the better lenses come at a more expensive price. Following the same procedure as above for the camera, and co-processor, OpenMV was not a solution we moved into the Feasibility Studies phase primarily due to the implications this solution would have on the aerospace engineering design team that the automatic target recognition product is being built for. Currently, the ASE team has designed their aircraft so that the co-processor is housed within the body of the aircraft and the camera is mounted underneath and on the outside of the fuselage. With a camera that is integrated directly into the co-processor, major design changes would have to be made to get power to the coprocessor while making sure the camera would still have good visibility underneath the aircraft and would be thoroughly secured to the aircraft.

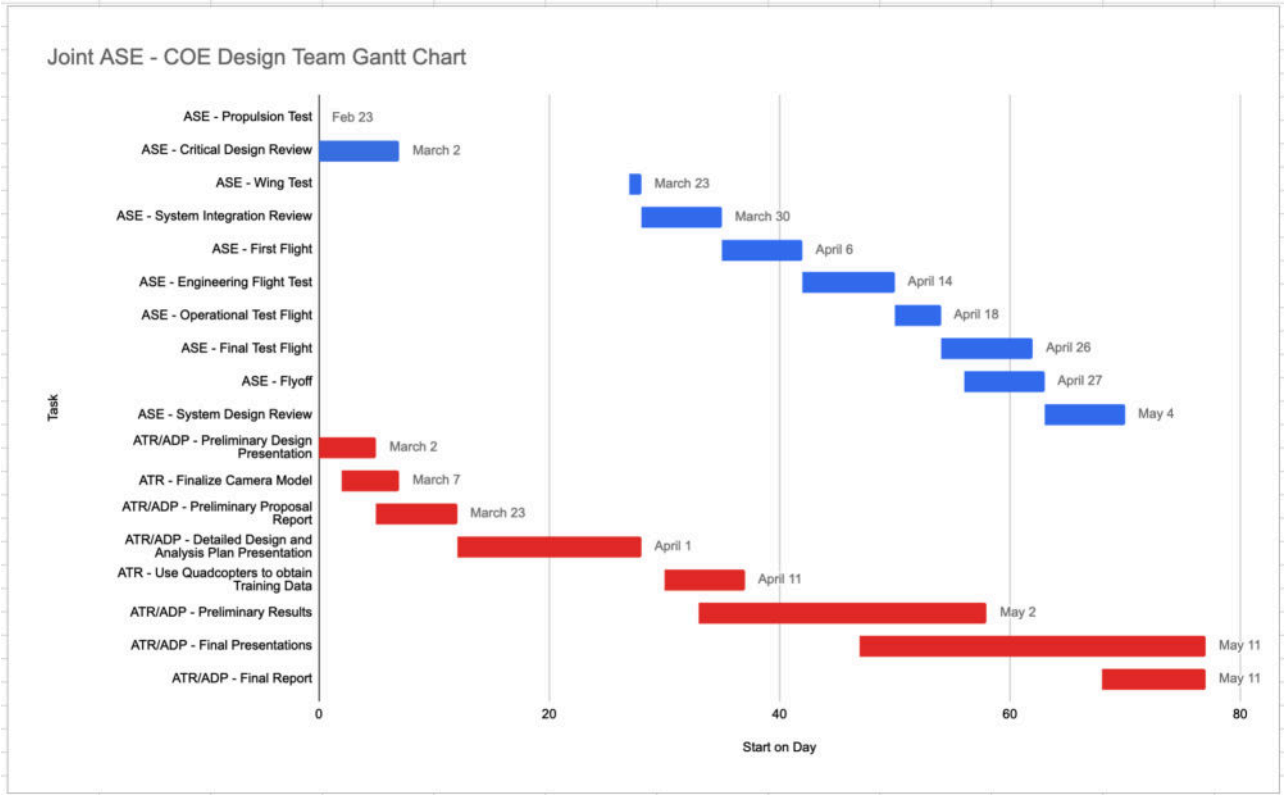

Figure 2: Final Gantt Chart Updated Prior to Mission Flyoff

RGB-based recognition, another valued approach to automatic target recognition, looks to expose color contrasts within an image frame to confidently identify the object of interest. In the case of the posed project objective, this method would work well to identify the high blue contrast of the tarps that the targets of interest will rest on and are in stark contrast in color to the green grass of the ground within the search area. By filtering out each pixel's red and green values and honing in on the numeric blue value, areas of dense blue in an image can easily be identified. The RGB-based recognition algorithm finds the max and means pixel contribution values of blue in the image and if that max - mean is greater than the predetermined threshold then it can be confidently assumed the high area of blue in the image is a tarp. Where the RGB-based method on its own fails to meet the project's objective is differentiating between the smiley and frowny faces of the targets of interest that rest on the blue tarps. The reasoning behind this stems from the fact that the faces of the target do not differ in color, but in orientation; an RGB-based method would struggle to make a distinction purely based on differing orientations.

The final, and most feasible, option when approaching target recognition software is OpenCV, or Open Source Computer Vision Library. After researching the available algorithms and looking at codes from OpenCV, we decided to utilize the Haar Cascades algorithm for target recognition. Haar Cascade classifiers are machine learning object detection programs that can potentially identify objects in photos and videos. The algorithm begins by calculating Haar features in the image or video resolution. Haar features are calculations that are performed at specified locations in a detection (position) window. The calculations involve comparing the differences between the sums of pixel intensities. Examples of Haar features include edge features, line features, and four-rectangle features. Internal images are created with the inclusion of Haar features. However, since the majority of the internal images created will be irrelevant to the user's project, something like Adaboost can be used to determine the "best" features. Once these features are chosen, they are used to train the classifiers. Weak classifiers are often combined together to create stronger classifiers. Finally, the classifiers are implemented and they're able to identify objects. One essential item to note, these algorithms will require training by using positive and negative images.

Unfortunately, Haar Cascades has a high false-positive rate. This is due to the age and nature of the algorithm; more recent ones offer a smaller margin of error. However, with enough tuning, the Haar Cascades code can be optimized against false positives. Furthermore, despite the potential pitfalls of larger margins of error and high false-positive rates, this algorithm continues to remain computationally inexpensive while operating at high speeds. Especially since we will be using a Raspberry Pi, it is important that we do not "overwork" the co-processor. As such, OpenCV is currently one of the only options available that meets all of our software and hardware requirements. Additionally, these algorithms are well documented and developed. Python has an OpenCV library that allows for the potential inputs of images, videos, and live feed to our algorithm.

Consequently, we proceeded to down-select both the OpenCV and RGB Recognition designs into the Feasibility Studies. It was determined that the OpenCV software is the optimal standalone design while the RGB Recognition software satisfied a moderate amount of the criteria. However, given that the RGB Recognition system can be integrated with the OpenCV software for image identification/classification, the team will be proceeding with a combined Haar Cascades and RGB

Recognition algorithm as our final choice on how to address the issue of object recognition in photos and videos.

Final Design Selection Detailed Design

Contributions: Nicholas, Ryan, Justin, Preston

The final design of our system was an adaptation of the conclusions made during the feasibility study phase. We continued to use the Raspberry Pi as our onboard computer and the Arducam as our computer vision camera. However, as we began to write our code for recognizing tarps and smiley faces, and especially as we began testing that code against real images, we realized that OpenCV's Haar Cascades modules were hard to work with. We pivoted towards RGB thresholding and feature matching. The other ATR team was proceeding with this method, and the team leader was knowledgeable about topics in computer vision, so we felt comfortable with this pivot. The modules we used for analyzing colors and features within images were still part of OpenCV, and as such the shift was not too large.

Hardware

Due to a shortage of Raspberry Pi's, we were only able to find a Model 4 with 2 GB of RAM instead of the 8 GB model that we had requested. The 8 GB of RAM was requested due to the expected load of processing many image frames per second. However, the 8 GB RAM was not a hard requirement. Next, we connected the Arducam to the Raspberry Pi via a CSI ribbon cable. These cables are extremely flexible, which was useful for routing the cable in the plane. However, the cables are also fragile, especially the pins at the connections. After breaking a pin on our first cable, we bought a pack of cables so that we would always have extras, especially during flights. For the SD card, which essentially acts as the hard drive for the Raspberry Pi, we purchased one with 64 GB of storage. This SD card was also approved for video streaming, meaning that it was capable of high data throughput, which is a requirement for our problem. Below is a picture of the Raspberry Pi connected with the Arducam via the CSI ribbon cable.

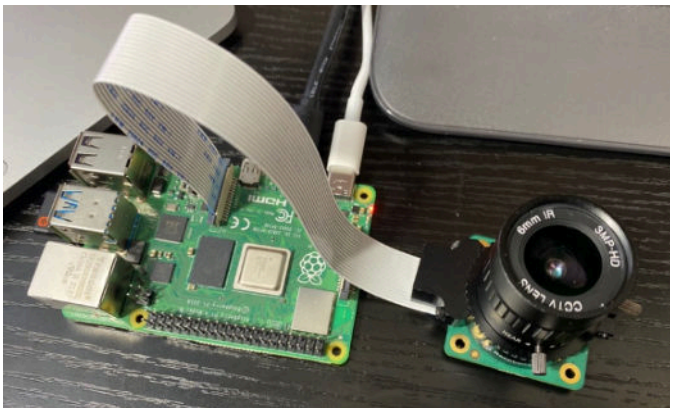

Figure 3: Testing & Development setup for Pi & Arducam Camera

Above is our total hardware system. The camera was placed flushed with the bottom of the plane, and the Raspberry Pi needs to be located near the Pixhawk, which is the onboard flight computer. To power the Raspberry Pi and to set up a communication link between the Pi and the Pixhawk, we connected both computers via the pin schematic below, connecting the Pi's transmit and receive pins to the Pixhawk's telemetry

port and the +5V and ground pins between both. This connection established power transfer from the Pixhawk to the Pi. The connection also established a serial UART connection, which is a configurable communication protocol among the hardware.

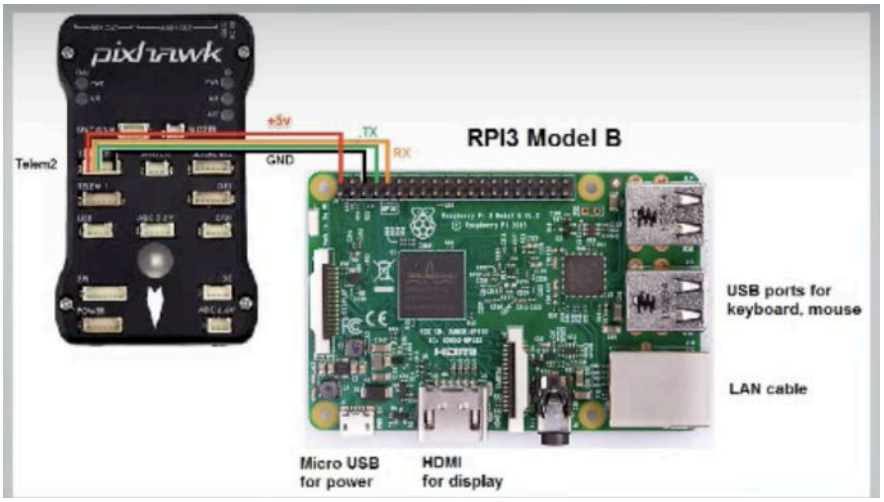

Figure 4: Connection between Pi and Pixhawk

After the hardware was set up, the Pi was configured for serial communication, which was enabled in the Pi's settings. Next, a software port was designated on the Pi which sends and receives messages from the Pixhawk– we decided to disable the Pi's Bluetooth and dedicate that software port for our serial connection. Finally, the frequency of the serial connection, also known as the baud rate, between the Pi and Pixhawk was agreed upon. We settled on 921600, which means the port is capable of transferring up to 921600 bits per second. After this, the software/hardware interface was set up, we downloaded the MavProxy library, which implements the Mavlink communication protocol into Python. At this point, the Pi is set up with power, and we were able to control the camera and communicate with the Pixhawk, all via a Python script.

Software

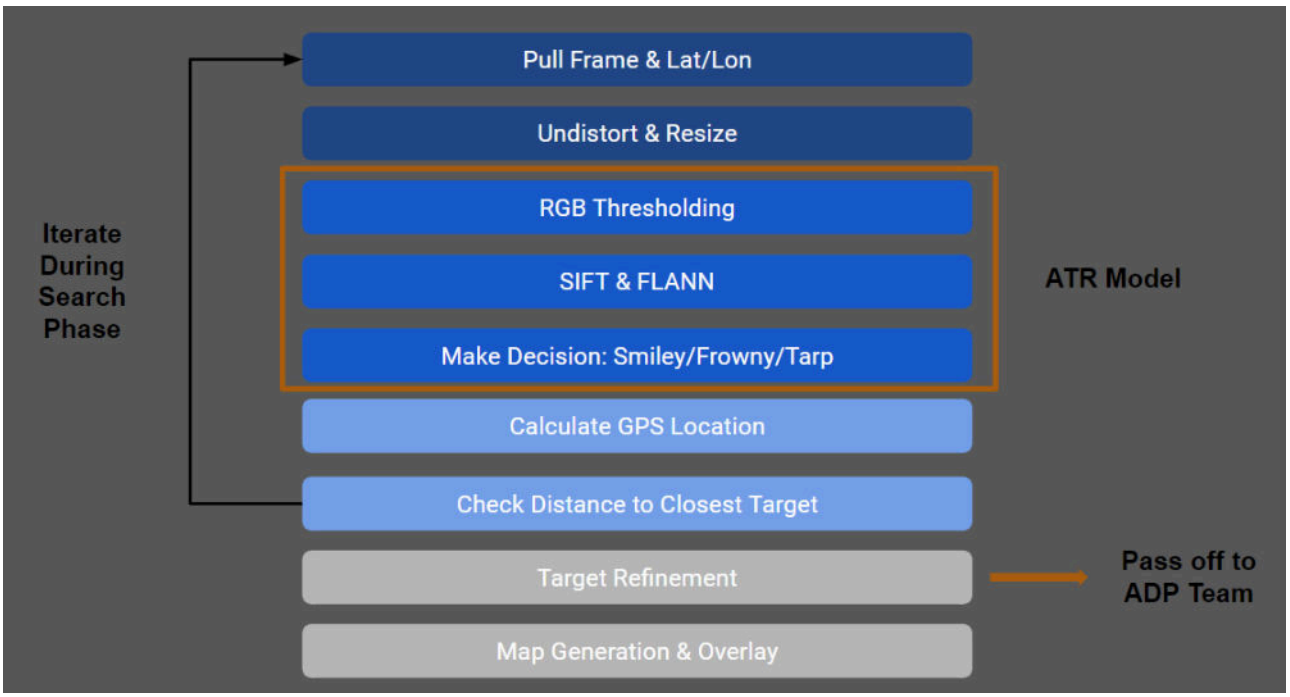

Figure 5: Flowchart for the Mission Software

To make the process to kick off our script as simple as possible, we added our program to a file in the Pi's home directory called ".bashrc". This bash script runs each time the Pi boots up. As such, by adding our program to the end of this script, our program runs each time the Pi gets power.

This flow is designed based on the ASE team's mission plan. As mentioned, our script runs almost immediately once the Pi gets power, which is when the plane turns on. After the plane is turned on, time is spent checking the plane and taxiing it to the runway. Because we don't want the camera to take images of the ground, we wait 5 minutes, by which time the plane is usually in the air. We decided to gather map images first because, per the mission plan, the plane cruises at 320 feet during this first stage. Being at a high altitude lets the camera cover more ground with fewer pictures, which makes map stitching easier. After three minutes, the automatic target recognition script kicks off. During this stage, the plane is flying autonomously and will descend to 200 feet for the search and surveillance phase of the mission. The plane flies according to a preset mission file which is made of waypoints that the plane should hit. The final waypoint is waypoint 30. Leveraging the serial connection with the Pixhawk, the ATR script can check which waypoint the plane is about to hit. So, while the next waypoint is less than 30, ATR continues running. Then, when the condition is met, ATR stops and we hand off control to the ADP team's script. After the flight is finished, we will check the map images to see which are best suited for stitching. Though this stitching could also have run during the flight, we wanted to reduce the complexity of the program as much as possible to avoid in-flight errors.

Next, we will discuss, in a high-level overview, the flow of our software specifically developed for the ConOps mission that the Aerospace Engineering Design teams are tasked with completing. As mentioned above, the Raspberry Pi camera is connected directly to the Raspberry Pi and this is how

live video feed from underneath the aircraft was transmitted into our developed scripts. We controlled the camera using the OpenCV library. During our drone flight to test out capturing images, we found that recording during the entire flight created movies of up to 15 GB in size. This file size is too large and unwieldy for our SD card, so we decided not to record our entire flight. Instead, OpenCV has a function that enabled us to stream input from the video camera, run our target recognition algorithm on that input, and only save those inputs which recognize targets of interest. In this way, we drastically reduce the amount of storage we were using. From the live video feed, still captures were pulled in along with the latitude and longitude coordinates that came from the Pixhawk. Next, the frame from the live video is modified to remove any distortions caused by slight impurities and curviness of the image around the edges of the frame. After being undistorted, the frame is then resized by cropping the width and height to help increase the resolution to complete the preprocessing of the image. After these modifications have been made the frame is then pushed into the Automatic Target Recognition part of the software. First, an RGB Thresholding approach is used to detect the deep contrast between the blue of the tarp and the green of the ground if a Target of Interest is visible in the frame. If there is a target in view, the thresholding method draws contours around the edges of the blue tarp and filters out all other unwanted parts of the image. The feature recognition algorithm is then applied to this cropped image to distinguish if the target of interest is a critical target, a frowny face, or a non-critical target, a smiley face or a blank tarp. Specifically, FLANN, a feature detection method, is used to make these decisions. The contoured-cropped frame is passed into both a feature detection method tailored for a smiley face and a frowny face. Based on the number of key-point connections made for each of the two methods, the designation of the target type is made. If the number of key-point connections made between the training image and the test image, or contoured-cropped frame, is less than 4 then the target is classified as a tarp. If the number of connections is greater than four then the method with the larger number wins the designation.

After this decision is made the Automatic Target recognition algorithm is completed and the software moves on to the calculation of the GPS coordinate of the target. In this phase, the pixel location of the identified target, within the original frame from the live video, is transformed into a latitude and longitude coordinate. This is done using the SWATH width of the Raspberry pi camera to determine how many meters are within one pixel of the camera's view. The distance between the center pixel of the frame and the location of the center of the target is then calculated. This distance in the x and y directions is then converted into meters using the SWATH width and then converted from meters into degrees and tacked onto the latitude and longitude coordinates of the center pixel which are the GPS coordinates pulled from the Pixhawk once the original frame is extracted from the live video. The GPS coordinates are then stored in an array within the code that houses all the GPS coordinates of the identified targets throughout the search phase of the ConOps mission. The identified targets are grouped with other targets that are within close proximity to each other. Once the search phase is complete, this list of identified targets is then refined and saved in a text file that is passed off to the ADP team to be used as input to their code. The target refinement process will be discussed in further detail later in the report. Finally, after the refinement process has been completed a map of stitched images of the area of interest (AOI) is generated and then overlaid with markers and text boxes that display the refined latitude and longitude coordinates of the identified targets. In practice, this map would be used as a resource for the Austin Fire Department as they try to rescue or get supplies to people in critical need.

Shifting gears now to a discussion on the final design for the camera calibration software, an algorithm in place for previous flight tests leading up to the second-to-last flight test (directly preceding the preliminary results presentation) used a binary mask methodology to map the grayscale image equivalent of the colored training image (hardcopy tan-brown chessboard) into a fully black-white image using image thresholding before ultimately identifying the pixel locations of the interior points in the pattern. Unfortunately, this method had severe limitations in the ability of the resulting calibration parameters (camera calibration matrix, and distortion vector) to consistently undistort warped image frames depicting features of the AOI. In particular, this method was heavily dependent on the distribution of light intensity and glare over the image in addition to the "levelness" of the surface. Inconsistencies in these properties made it difficult for the algorithm to identify interior points in associated training images producing resultant calibration parameters that were not robust enough for undistorting test images spanning an expected large range of warped properties during the mission. An example of one of the few training images whose interior points were identified, and that was successfully undistorted is shown in the figure above. Effective in the preparation for the final ConOps mission and final simulation run, changes were implemented to the algorithm producing the flowchart shown in the figure below. In particular, the algorithm involved first mapping the training image from an RGB to a Grayscale colorspace and then feeding the resultant image into a for loop where the algorithm iterated through the pixels of the image and stored pixels coinciding with identified interior points of the image pattern. The pixel locations were highlighted on the grayscale image similar to the feature of the previous algorithm. Once the last training image was read into the workspace, and analyzed, the pixel locations of the interior points of each of the images were used to obtain a resultant camera calibration matrix and distortion vector. These parameters will be discussed in more detail in the "Analysis results and discussion" section of the report.

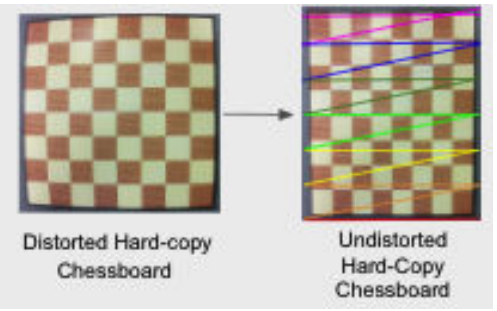

Figure 6: Hard-Copy Chessboard Before and After Interior Points Identified

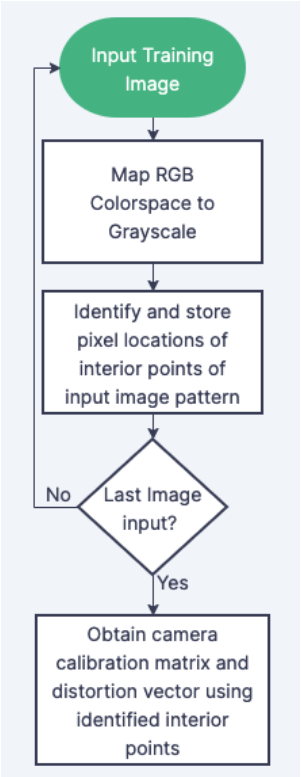

Figure 7: Camera Calibration Flowchart

Undistortion

The next step in the detailed design of the software system involved developing and implementing an algorithm for removing radial and tangential distortion from test images of the field given the calibration parameters (camera matrix, and distortion vector) returned from the camera calibration algorithm. In referencing the flowchart presented in the figure to the left, the first two steps in the algorithm involved reading in the two calibration parameters into the workspace followed by storing the size of the distorted test image. Then, the calibration matrix, distortion vector, and image size were used to obtain a transformation matrix unique to the input test image. Next, an undistortion and rectification transformation map in the form of vectors in x and y was evaluated using this unique transformation matrix. The vectors in this map quantify how the distortion in the 3D world frame is undistorted into the corresponding 2D image frame using rectilinear geometry. Finally, the transformation map was applied to the original test image to remove distortion. Before and after images showing a digital training image of a chessboard being undistorted after applying the transformation map are shown in the figure above and to the right.

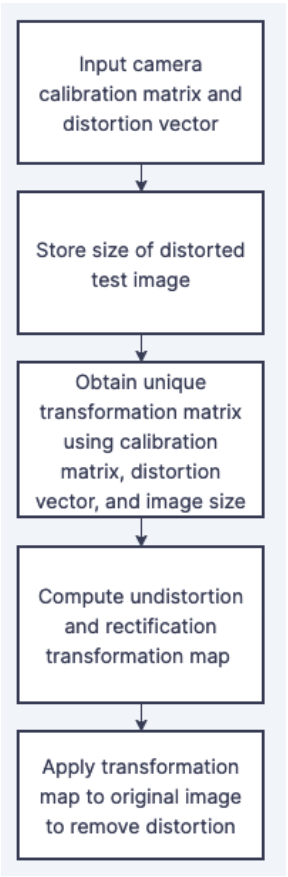

Figure 9: Flowchart for Un-distortion Algorithm

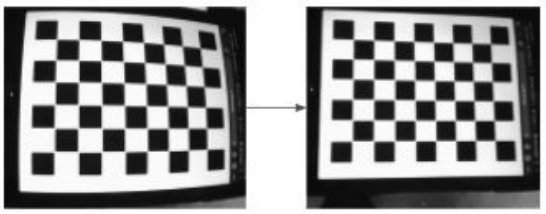

Figure 8: Chessboard Training Image Before and After Application of Transformation Map

Target Proximity and GPS Coordinate Refinement

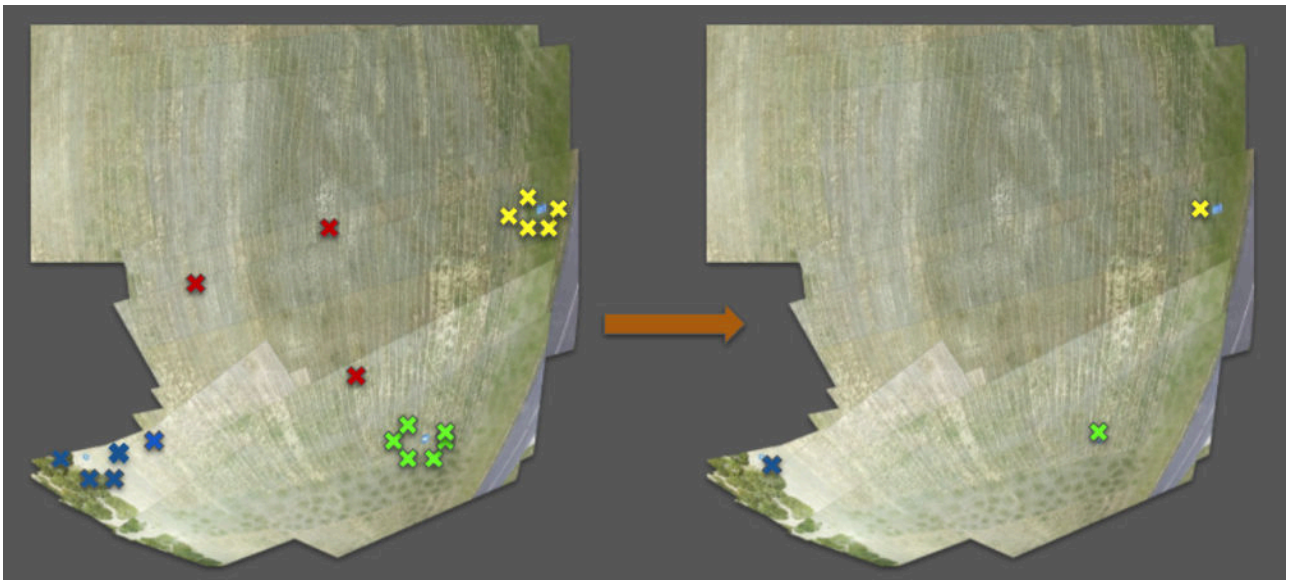

Figure 10: Target Refinement Process

As mentioned above, after a target has been identified and its GPS location has been nailed down it is added into a grouping of previously identifications that are in close proximity to one another relative to their latitude and longitude coordinates. Our algorithm checks the distance between the current target and the average latitude and longitude coordinates of each grouping of targets if the distance is within 11 meters then the current target is grouped with the targets in that group. Essentially, this grouping of targets is to say that the camera and calculation of the GPS coordinates are error-prone and need some cushion when calculating the latitude and longitude coordinates of the same target from different angles. This grouping of targets is an iterative process throughout the duration of the search phase of the ConOps mission. Once the search phase has been completed, our software moves into the target refinement process which takes in the large array of targets grouped by their proximity to one another. If the total number of targets within a group is less than 4, then the group is thrown out as false identifications. The number 4 was selected based on the results we saw during the flight tests that our ATR algorithm was tested on. After removing the false identifications, the average latitude and longitude coordinates for each grouping of targets were found along with the counts of each type of target within the grouping. The target type with the largest number within the group became the final designation for the refined target. For instance, if a grouping of targets contained three frowny faces, ten tarps, and twenty-five smiley faces, then that grouping would be classified as a smiley face. Once this refinement process is completed, the final target classifications along with their GPS coordinates were saved in a text file on the local filesystem for the ADP team's use.

Map Stitching

For the purpose of creating a map of our area of interest, our final design involved the use of an image stitching algorithm. Our initial plan was to have the algorithm run autonomously as the plane is flying; however, due to complications concerning how the algorithm might choose images from those that were taken, we ultimately decided to run it manually in post-processing. Thus, the plan would be to manually choose which images to feed into the algorithm once we receive the images from the flight. The figure below displays the overall workflow for our algorithm.

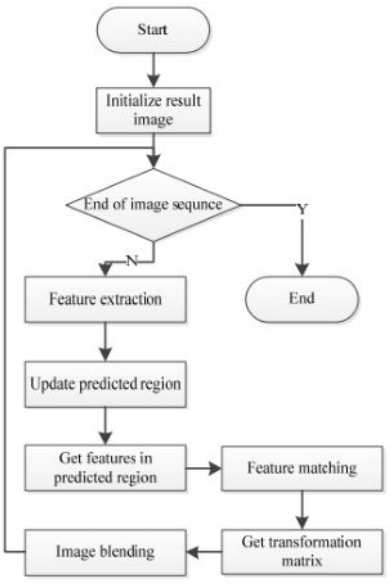

Figure 11: Basic Flowchart for the image stitching algorithm

It utilizes SIFT feature extraction, FLANN feature matching, and homography to receive the transformation matrix. The algorithm then iterates this process for every image in the given directory and returns images each time a new image is stitched onto the final image. The program would be loaded onto the raspberry pi to be run from there.

Identification and GPS Coordinate Overlay Algorithm

The last step in the flow of the software system involves an application of an algorithm that overlays location pins, and text boxes with GPS coordinate identifications given the generated map output from the stitched image algorithm. A flowchart depicting the movement of information in this algorithm is shown in the figure on the left. The first step in the procedure involves reading in the text files with GPS coordinates, identification strings, and the file path to the stitched map. Then, the file path to the text box and location pin overlay images are read into the workspace. From here, the identification strings and GPS coordinates from the refined GPS coordinate text file are stored in lists. Next, the image size of the stitched map is stored. Then, the algorithm iterates through each identification in the list, and for each identification of tarp or. TOI, it first computes the GPS latitude and longitude transformation factors which rescale the GPS location of the identification to the GPS range of the AOI in the image. From here, the x and y pixel locations of the centroid of the TOIs and tarps on the image are approximated by rescaling the GPS coordinates using the image size and the transformation factors. Then, the location pin is overlaid at this corresponding pixel location. From here, the text box is overlaid at a hardcoded pixel location slightly offset from the pixel location of the location pin and identification to prevent overlap. Lastly, the identification string and GPS coordinates are overlaid onto the text box. Then, the algorithm checks if the last GPS coordinate pair/identification has been reached and if so, then it ends the loop, and the resultant overlayed image of the generated map is saved to the local file system effectively marking the final procedure of the software system.

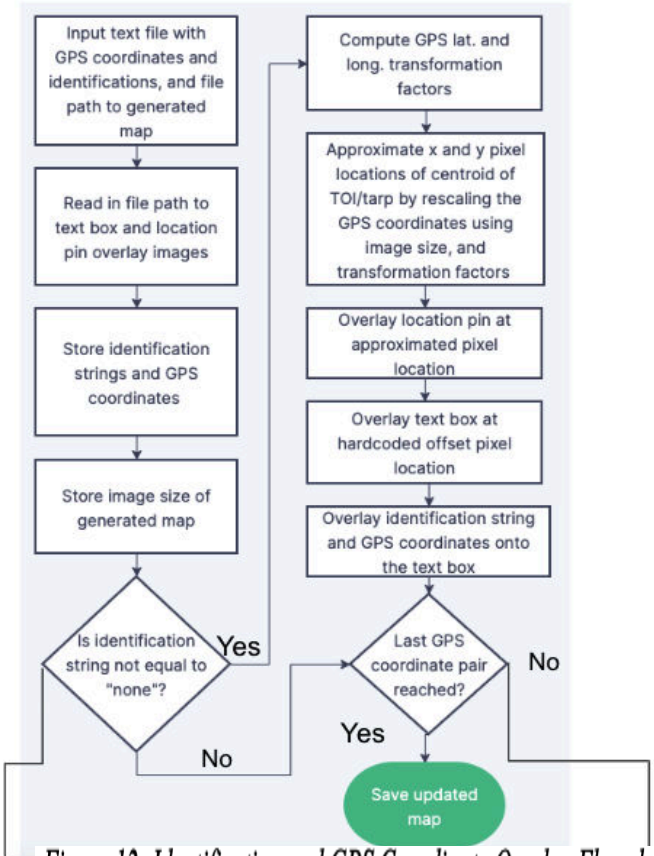

Figure 12: Identification and GPS Coordinate Overlay Flowchart

Script Flow + Pixhawk Communication

Finally, we needed to decide how to control each of these pieces of software as the plane flew. An added complexity of our project is that we have no access to the Pi while it is flying. This means that we can't send commands on the terminal, and we can't see what is happening as the plane is flying. It is hard to connect to the Pi while the plane is on the ground as well because the Pi is located under the wings of the airplane.

Analysis Plan

Contributions: Nicholas, Ryan, Justin, Preston

Our first idea to analyze the code we had was to acquire training data that was as realistic as possible. First, we took pictures of the tarps and faces from a parking garage about 80 feet high. We had some baseline data to begin running the RGB and feature matching code with these pictures. Then, to acquire more realistic data both in height and vibrations, we mounted our hardware system onto a drone and flew that drone out over the flight field. During this test, we recorded video of the drone flying over tarps and faces. Afterward, in the lab, we split the video into frames and ran our algorithms on those frames. The drone video contained significant vibrations, but it was still helpful in getting realistic footage from 200 feet in the air of the flight field, of the tarps, and of the color contrast between the field and those blue tarps.

We also had test flights on the plane itself to test our code. During the first test flight, we simply recorded video from the flight. With this video, we then narrowed down on the color threshold for our RGB algorithm. Then, we tested our ATR script against this video to understand what was being classified correctly and what was not. We learned that when the plane banked and grabbed pictures of trees, those trees sometimes led to incorrect classifications of faces. Pictures early on in the flight when the plane is still on the runway, taking off, or grabbing images of the track and the building, also contributed to false-positive classifications of both tarps and faces. False-negative classifications (i.e. a face is in the video but did not get recognized) usually occurred when the feed was vibrating excessively or the plane itself was banking too extremely. During those sections of the test flight when the plane flew parallel with the ground, the ATR algorithm was successful in classification.

Our software, on the plane, was dependent on a successful connection between the Pixhawk and Raspberry Pi along with a successful transfer of live video from the camera to the Raspberry Pi. Both of these connections were tested thoroughly in the lab to make sure our software was set up correctly to interact with both of these hardware components. A break case for our software would be if any of these two connections failed for any reason. Without a connection to the camera video, and more specifically still frames, would not be pushed into our software for the ATR process to analyze, thus causing the software to break. Another break case, and ultimately the reason that our software failed during the actual ConOps mission, is when there is a failed connection between the Pixhawk and Raspberry Pi. If our software is unable to establish a secure connection with the Pixhawk then the script waits and essentially does nothing. Without this connection, the live video feed is not initialized and a log output file is saved to the local file system denoting a failed connection.

Break cases on the actual feature recognition part of the software are not as detrimental to the functionality of the code. There was a try-except block placed into RGB and thresholding part of the code to take care of any frames that were not suited or liked by OpenCV. This way the software could toss any frames like this and move on to the next frame.It is also worth noting that the video recordings of the three flight tests were saved as training data for use to fine-tune the thresholding values of the RGB's contour-cropping methodology when a trap was recognized within a frame. The cropping was heavily dependent on the altitude the plane was flying at. For instance, at lower altitudes, the tarps and targets of interest appeared bigger in the frames, and thus the cropping threshold needed to be increased to account for this enlargement. Once we knew for certain the altitude at which the search phase of the ConOps mission would take place at, which was 200ft, we were able to tune the thresholding accordingly. This same methodology was applied to the tuning of the threshold used to determine the proximity of targets to previous identifications. This was a difficult task to perform as we were never able to test the grouping and refinement processes in flight. So our educated guess of 11 meters was used to give the GPS calculation the proper cushion it needed to recognize if a target had been identified already.

Analysis Results and Discussion

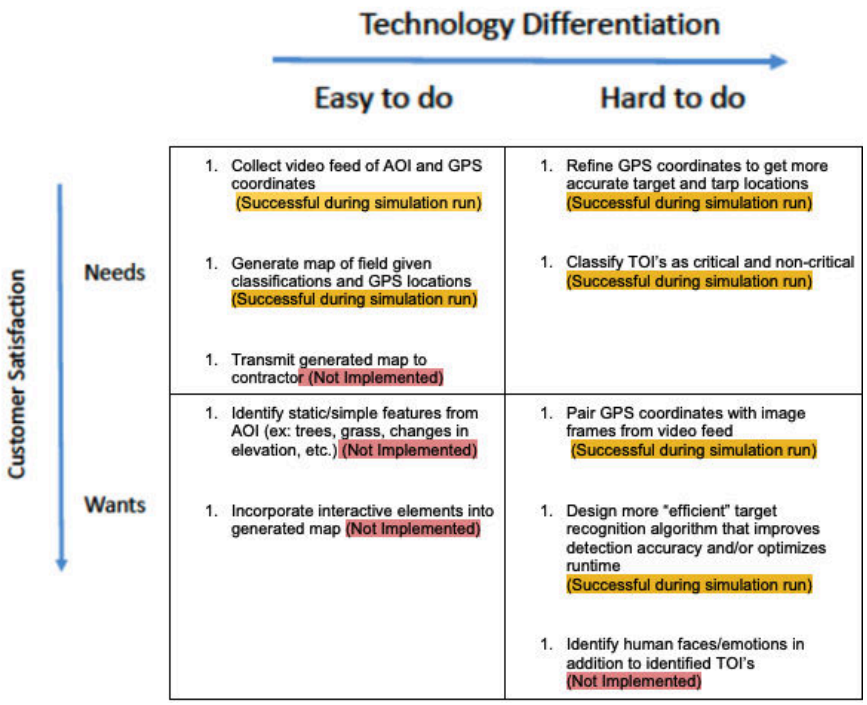

Figure 13: Final Decision Matrix

Contributions: Nicholas, Ryan, Justin, Preston

Decision Matrix

To begin the analysis results section, a detailed discussion of our team's decision matrix shown in the figure below with deliverables and stretch goals delineated according to whether or not they were accomplished will be provided. To first provide context into these accomplishments, it should be recapped that the team was unable to acquire meaningful results from the final fly-off mission as a consequence of a connection issue between the Pixhawk and the Raspberry Pi that prevented our software system from recording video in flight, and thus performing the target recognition and map generation. Thus, all tasks listed as successful were completed in the context of a post-fly-off simulation run using the other Image Processing Team's video feed from their respective fly-off. In referencing the top-left partition of the matrix, we have the easy-to-do guaranteed deliverables. The first involved capturing video of the AOI and collecting GPS coordinates of the field. This deliverable was met through the acquisition of the other image processing team's video feed from the mission, and by using a random number generator to simulate GPS coordinate locations within the bounds of the AOI. With that being said, although this wasn't implemented in the mission, it was confirmed to be functional during pre-flyoff tests The

second guaranteed deliverable involved generating a map of the AOI given identifications from the target recognition algorithm and GPS coordinate readings. This deliverable was also met during the simulation run where the team's software system used image slicing and stitching to produce a generated map of the field given the provided video, and associated GPS coordinate positions in a text file. The third and final easy-to-do guaranteed deliverable involved transmitting the generated map to our contractor (namely, David Meskill of the ASE department). Since the simulation was performed after the mission fly-off date, our team was unable to transmit our results to our contractor for scoring and thus this deliverable was not met. Shifting gears to the hard-to-do guaranteed deliverables, the first item involved refining GPS coordinate positions to get more accurate target and tarp locations. In particular, the goal was to implement an algorithm that refined GPS coordinate locations corresponding to TOI's/tarps so that values within a small positional threshold were grouped into individual lists, and the average value of the coordinates in each list would be evaluated and stored as the approximate GPS location for that identification.

This deliverable was successfully accomplished during the simulation run. The second hard-to-do guaranteed deliverable involved implementing a target recognition system that could accurately classify TOIs as critical and non-critical. The results from both the simulation run and final flight test correctly identified each of the tarps/TOIs with an additional erroneous tarp identification, and thus, were deemed successful. Shifting gears to the bottom-left partition, we have easy-to-do stretch goals. The first of the items involves the goal of identifying miscellaneous features in the AOI such as trees, grass, changes in elevation, etc.). Unfortunately, due to time constraints, our team was not able to implement this feature of the system. The second goal involved incorporating interactive elements into the generated map. In a similar fashion, our team was not able to implement this feature due to time constraints, however, in consideration of future work, the team would consider using Python's "Tkinter" package for implementing an interactive GUI that would enable the user to zoom in and out of the published map and activate collapsible windows that display identifications and GPS coordinates. Lastly, the bottom-right partition depicts hard-to-do stretch goals. The first involved pairing GPS coordinates with image frames from the video feed. Although randomly generated during the simulation, GPS coordinates mimicking the centroid of the AOI in the image were paired with associated image frames. The second involved implementing a more efficient target recognition algorithm that improved accuracy and/or optimized runtime. We were able to accomplish this goal by modularizing our code, and making changes to threshold parameters in the target recognition algorithm that improved classification accuracy in the days leading up to the final fly off and successfully implemented this integration during the simulation run. Lastly, the identification of human faces/emotions was set aside as a hard-to-do stretch goal that the team was unable to complete due to time constraints but would likely be implemented using a form of feature recognition similar to that used for the targets.

Camera Calibration

Shifting focus to the results of the camera calibration, we have an image in the first figure below which depicts the properties of the camera calibration matrix. As a recap, the "fx" and "fy" parameters stand for the x and y components of the focal length, and the other two parameters represent the components of the optical center. As touched upon in the design section, the images

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 14: Camera Calibration Matrix Definition

3910.01	0	2021.48
0	3896.32	1417.29
0	0	1

Figure 15: Camera Calibration Matrix from First Iteration

of the hardcopy chessboard contained inconsistent light intensity and glare properties, in addition to portions of the chessboard being at different relative displacements (i.e. unlevel surface). This generated a matrix (shown in the second figure) with limitations in its ability to undistort test images of the field and ultimately, it was determined that the stitching algorithm did not effectively overlap images throughout the AOI given the undistortion parameter. To account for this, a new batch of training images was obtained that depicted images of a digital chessboard with uniform light intensity and minimal glare whose resultant calibration matrix is shown in the third figure to the right. After applying the resultant camera calibration matrix and distortion vector to the same batch of test images (batch from a previous flight test), it was determined that the stitching accuracy improved significantly while the undistortion of individual images also improved noticeably. The team hardcoded the calibration matrix and distortion vector from this calibration and moved forward to the mission fly off and simulation run with these parameters. Ultimately, it was determined that the undistortion and stitching overlap accuracy of the images in the generated map from the simulation run were qualitatively superior to that of the results obtained from a previous flight test using the former iteration of chessboard training images. Thus, the calibration modifications were fully successful.

587.394	0	329.668
0	590.266	265.307
0	0	1

Figure 16: Camera Calibration Matrix from Final Iteration

Undistortion

As addressed in the previous section, the camera calibration matrix and distortion vector from the calibration algorithm were used to undistort test images of the field, and it was determined that the parameters obtained from the iteration of training images of a digital chessboard were largely successful at removing both radial and tangential distortion from the images.

Figure 17: Field Test Image with Distortion

Figure 18: Field Test Image with Distortion Removed

An application of this undistortion to a field image shown in the figure to the right produced an undistorted test image shown in the figure to the left. It is visible that the warped/curved field lines at the bottom-left portion of the distorted test image were successfully straightened.

Simulated Results

Since we were unsuccessful in gaining meaningful, or really any, results from the actual ConOps mission fly-off, we resorted to using a video captured during the flight by the plane of the other Aerospace Engineering Design team. In an effort to showcase the components of our software that had been thoroughly tested, the video was passed into our software and sliced up into frames in the same manner that a live video would have been. We cropped this video into a ten minute segment of the plane flying directly over the targets of interest. After ten minutes of frames being passed through the ATR algorithm, the following results in the chart below were obtained.

Type of Target	Search Phase Identifications	Refinement
Tarp	265	4
Smiley Face	261	1
Frowny Face	54	2

Figure 19: Table of Simulated ATR & Refinement Results

The same two training images of the smiley and frowny faces used in previous flight tests were used to do the feature matching during the simulation. Observed above are a significant amount of tarp and smiley face identifications which is a by-product of the ten-minute segment of the video that was used having a high concentration of time spent with the blank tarps and smiley face in view. A slight modification had to be made to the GPS coordinate retrieval since our software was not connected to a flying plane that had live latitude and longitude coordinates nor a live altitude reading. So instead of pulling these values from the Pixhawk, random GPS coordinates were generated within the area of interest where the ConOps mission took place for each frame that was sliced from the video. This process of random generation invalidated the correlation between the latitude and longitude coordinates of the targets in the video and the coordinates that our software calculates and then uses to determine proximity to other identifications. However, the target refinement process could still be done with the knowledge that the final latitude and longitude coordinates and target types would not align with the video used for the simulation. Noted in the table above are the results from this refinement procedure. A total of 580 target identifications were reduced down to just 7. Ultimately this shows that our software has the ability to recognize when it has come across a target it has already identified and use that information to obtain a more accurate and refined GPS coordinate for it.

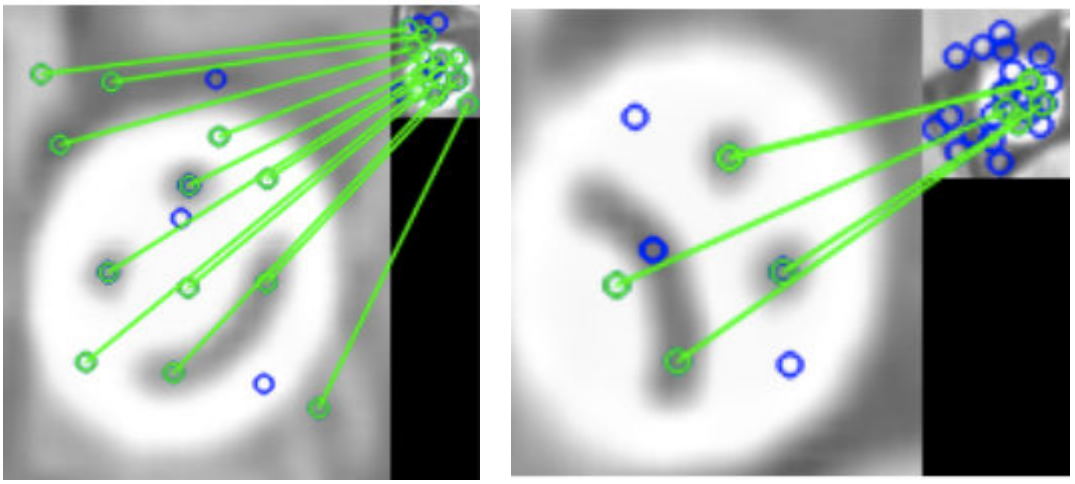

Figure 20: SIFT & FLANN Results

Shown above are the results of the feature mapping part of the ATR algorithm. On the left of each image is the training image used for each classifier method. Green circles and lines mark positive correlations between the training image and the test image in the upper right corner. Blue circles represent places where the feature mapping algorithm was unsuccessful in finding a match between the training and test image. For the smiley face shown above the sheer number of green lines and the minimal crossover of the lines indicates that there is a high correlation between the training image and the test image, and thus this frame from the simulated video was correctly classified as a smiley face. On the right. the same process for a frowny face classification is shown. There are less successful key-point connections, however, the relatively low crossover of the green lines indicates that the feature recognition algorithm was successful in identifying the target as a frowny face. This process was done for each frame that was sliced from the video and frames where fewer than four successful key-point connections were made were classified as a blank tarp.

Map Stitching

Our requirements for our map were to generate a map, by any medium, of the area of interest that displayed all candidate targets and their classification. Because we were unable to ultimately receive images from the actual final fly-off, our script was run on selected frames from a previously recorded flight. As mentioned before, we opted to manually choose these sets of images from the images taken from the video due to complications we encountered when trying to automate the selection of images. The figure below displays what we consider to be the best outputs from our script given the quality of our test video.

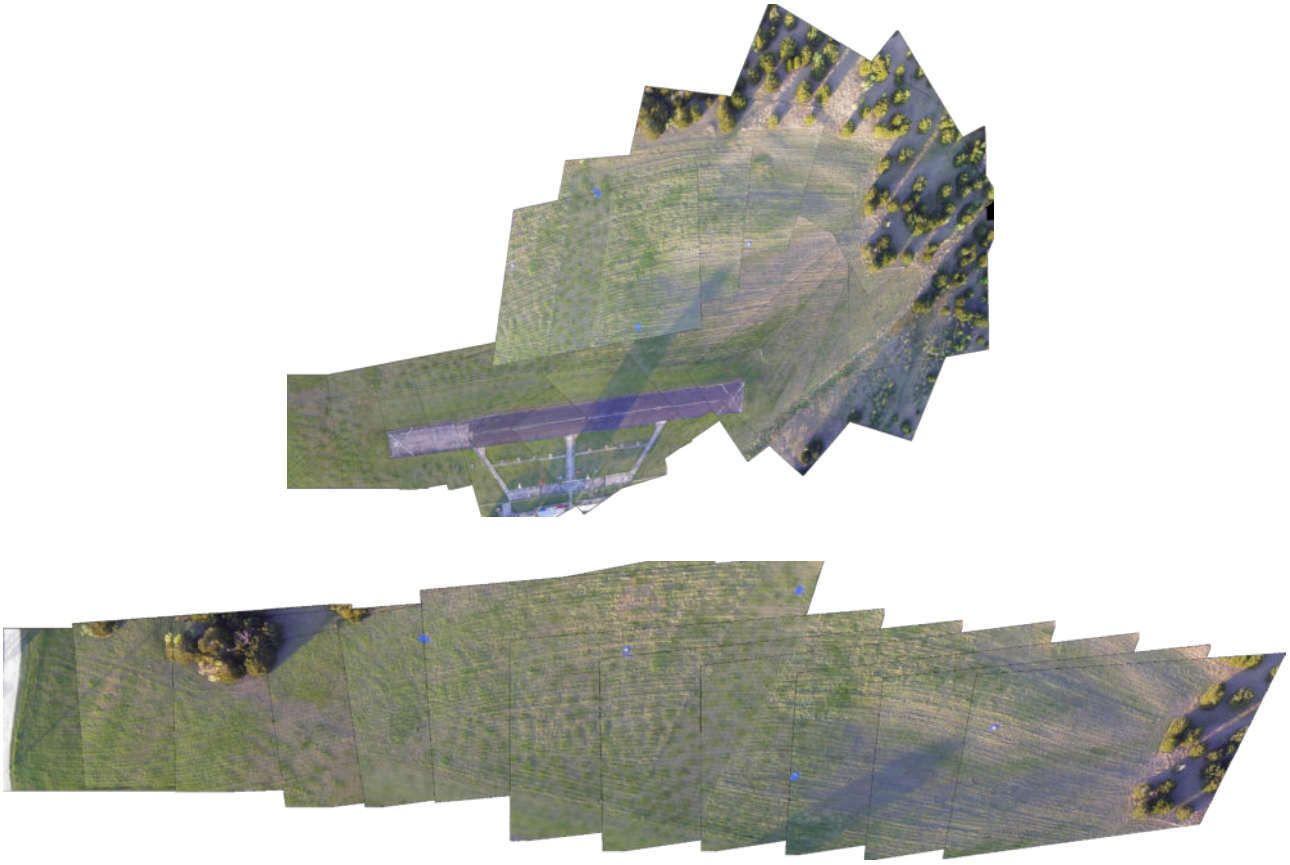

Figure 21: Two Map Generated from our Stitching Script that displays the area of interest along with each of the candidate targets

As we can observe from the images above, these two maps, while they both technically display the area of interest, each display the field differently while still containing all of the candidate targets. The stitching algorithm is limited mostly to the path of the plane, and as such these two maps display two different passes of the field. The top map displays a larger area of the field and includes the runway and a small amount of the periphery. The bottom map instead displays the entirety of the field lengthwise and all of the targets.

Identification and GPS Coordinate Overlay

Wrapping up the discussion on the analysis results from the final simulation run, we have an overview of the mission area map generated from the "identification and GPS coordinate overlay algorithm". Presented in the figure to the right is a digital map produced from text boxes and location pins overlaid onto the map stitched using images provided by Wilson's imaging team. It should be noted that the resultant map was manually cropped to only include the region containing each of the identifications before inserting into the report to conserve space, and that the text is not clearly visible in this cropped image, unlike the base image due to a reduction in focus from the inserted screenshot. As shown in the figure, the red location pins have been overlaid onto the approximate centroid of each respective TOI/tarp, and text boxes with identification and GPS coordinate designations have been overlaid at a fixed offset position from these locations to prevent overlap. A limitation to these results is that for the purpose of the simulation, since the GPS coordinates were randomly generated within the bounds of the GPS coordinates coinciding with the AOI, they do not actually pair up with any of the identifications from the target recognition

algorithm. Thus, a major takeaway from these results is that while the overlay functionality from the simulation was successful, the base functionality of the algorithm where the pixel locations were determined through GPS coordinate rescaling was only confirmed to be successful in the pre-flight checks leading up to the mission fly-off, and could not be validated from the simulation run.

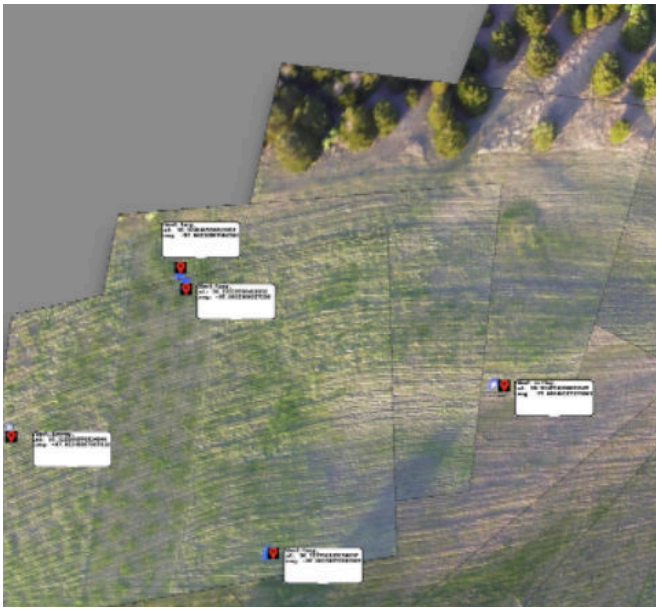

Figure 22: Cropped Mission Area Map Image with Overlays from Simulation Run

Error Analysis

As has already been mentioned, our code failed during the final flyout. We only learned after the flight that the Pi was never able to establish a connection to the Pixhawk. We had tested connecting to the Pixhawk numerous times before, and it was not an error we were expecting to see. Unfortunately, the way our script was written, this error caused the rest of the script to fail.

There are ways to fix this specific error, such as having some way to communicate, in live time, output messages from the Pi back to ground control. This way we could have known immediately that the connection failed and asked the ASE team to land. However, I think there were many uncertainties, driven by time constraints and testing constraints, that led our team to this moment where the final fly-off did not go according to plan. For example, we only pieced together our entire program a week before the final flight, and the program ended up being large. It consisted of connection to the Pixhawk, acquiring images for map stitching, undistortion of those images, running the map stitching itself, RGB, feature recognition, and then the handoff to the ADP team. Each of these pieces in themselves could be 100 lines of code. So, a week before the final fly-off we threw together these codes into one script. We attempted what testing we could, but the flow for this large script was driven by reading in altitude and GPS information from the Pixhawk. In the lab, we had no testing environment where we could simulate the plane flying according to a mission file and read in that data to our script. As such, we could not test our script in full until it was flying on the plane and the plane was flying according to a mission file. Realistically, then, we only had three flights to run our entire code and debug. Ultimately, this was not enough time for us to build the script we had set out to build.

A testing environment to run our script with parameters from a simulated flight would have been extremely useful. We also should have asked the ASE team for a flight day devoted to testing our own code. That way we could get more and shorter flights in with quicker feedback loops. We update the code, the plane flies for 15 minutes, it lands, we see what goes wrong, update the code, and then repeat. Proper logging was another aspect lacking in our code. During the first flights, we

had not incorporated a comprehensive debugging scheme into our script. Since we cannot see what is happening to the Pi as it is flying, it was hard to know what worked and what did not.

Impact From a Global, Societal, and Environmental Perspective

Contributions: Rohan

The completed scripts and programs have the potential to create positive societal and environmental impacts when implemented. The greatest benefit of the AFD having access to this technology is that the average person affected by harsh weather conditions now has a greater chance of getting assistance. With the UAV, the AFD can locate and deliver supplies to victims of flooding much faster than they could've previously. Theoretically, stranded victims would be able to be located and given supplies in a single pass from the UAV. As such, the stranded victims of flooding now face less stress since there is the chance that the UAV will transport the supplies needed to get by until the flooding becomes manageable enough for the AFD to navigate. Another positive impact that comes from depending on automated machines to perform these difficult tasks is that the members of the AFD will no longer be required to risk their lives as often to save the trapped individuals and families. With the UAV, waiting longer times for the weather to dissipate before heading out in-person is now a viable strategy. There are also several environmental impacts to utilizing this machinery. By depending on the UAV to locate targets and deliver supplies the AFD is able to save time and energy. A UAV will not face the same navigational restrictions during a natural disaster that a non-aerial vehicle will. Additionally, UAVs often use batteries and electricity as a source of power instead of gas like other search and rescue vehicles. All in all, there are only positive impacts on society and the environment should this search and rescue strategy be put into effect by the AFD. This project is designed to help people and its completed version will certainly make it easier to do so.

Conclusions and Future Work

Contributions: Rohan

After the completion of the main scripts, the target recognition, and map generation, they were integrated into Raspberry Pi and set up with the ASE team's plane. It is important to note that the scripts were implemented several times over the past few months. Unfortunately, due to unforeseen circumstances, technical issues prevented the Raspberry Pi from starting during the final flyout. As such, the COE team decided to use a video recording of a past flight to simulate results that serve as the final flyout. The conclusion will focus on the results achieved from past flights and the simulation.

The first deliverable was the Automated Target Recognition software. In both the practice flights and the simulation, this script was able to successfully identify positive and negative targets with a small number of false positives. When looking at practice flight test #4, we saw that out of a little bit more than 20,000 frames, there were less than 25 false positives. After touching up this script, we inputted the video recording of the flight and achieved a similar level of success regarding target

identification. However, since our final results relied on a video recording, GPS coordinates had to be randomly generated for each frame that was analyzed.

The next deliverable was the Map Generation script. Even though maps were able to be successfully generated from the practice flight and the simulation, many issues arose that needed to be addressed before the final product could be presented. Due to the hardware limitations of the camera and the plane, there was a severe distortion in the camera that created extremely off-putting maps of the observed area. As such, a camera calibration matrix was calculated and hard-coded into the scripts to ensure that all the images that were to be inputted into the map generation algorithm were undistorted. The preliminary results, before the camera calibration was completed, include a successfully generated map with quite a lot of radial distortion. The tarps with the targets can be observed on this generated map. However, the quality of it leaves much to be desired. Upon the completion of the calibration script, the camera calibration matrix was implemented into the main scripts in order to achieve new results. Fortunately, the generated map from the simulation is made up of images that have significantly less distortion to it. While it isn't necessarily perfect, the map generated in this iteration is able to showcase all five targets in a minimally distorted view. All 3 empty tarps and the 2 with the positive and negative targets on them are displayed on the map.

Due to the fact that technical issues prevented the COE team from getting proper results on the actual flyout, there are many steps that need to be taken for the future application of this project. Since the final outputs are a result of a simulation, the ability to generate accurate GPS coordinates in real-time wasn't even tested. Furthermore, the scripts were implemented separately instead of together in order to get the final results. Therefore, in order to ensure that this issue does not repeat, the COE team should be able to have flights with the plane to only address their own issues. Over the past semester, the COE teams were only able to test their deliverables at the convenience of the ASE teams. This created situations where both teams were working on their own projects on the airfield, preventing the COE team from being able to analyze their issues undisturbed.

The COE team believes that the reason no results were gathered was that the Raspberry Pi didn't properly sync up with the rest of the plane, thus the code was never even executed. This is a result of technical issues on the day of the flight that caused the ASE and COE teams to integrate the hardware in a different way that was not fully tested due to time constraints. Therefore, the first major step in future work will be to properly integrate the Raspberry Pi with the plane and ensure that there is no chance of startup failure.

The next area of future exploration involves specific applications of the Target Recognition and Map Generation scripts. The most obvious is that the COE team will need to make sure that the map is generated and transmitted successfully in live time. Circumstances prevented the testing of this feature, however, it is still an important deliverable that needs to be met. In addition, the COE team was hoping to be able to incorporate interactive elements into the generated map so that the receiving team can utilize the map with greater ease. The next area of development is the target recognition software. The COE team noticed that there were a lot of other features at the airfield that could be identified in a more advanced version of the code. In addition to the targets, it is believed to be important to identify other static features like trees, grass, roads, and buildings. By having this code identify other features, it is able to "disqualify" these areas from its primary purpose. Additionally, changes in elevation from the ground need to be noted as well in order for the Payload Team to have more accurate calculations. Potentially communicating with the Pixhawk

to gather these elevation changes and outputting them is an important next step to improving the overall effectiveness of the UAV. Finally, the algorithm should continue to be trained so that it is able to theoretically differentiate between human emotions. Implementing this aspect of the code can help future team members discern between levels of danger in the victims being saved.

All in all, despite the successful results gathered from the practice flights and the simulation, it is quite obvious that the biggest limitation for the engineering teams was time. Should more time be provided in the future for the team members, it is almost certain that the scripts will be more refined and have more complex qualities to them. Additionally, it will make certain that the overall integration of all hardware has minimal issues and will also allow for proper backup plans to be created.

References

Hoed T., Brandt D., Soerge U., and Wiggenhagen M. (2012). “REAL-TIME ORIENTATION OF A PTZ-CAMERA BASED ON PEDESTRIAN DETECTION IN VIDEO DATA OF WIDE AND COMPLEX SCENES”. Intercommission Working Group III/V.

Lt-C/M2420 (n.d.). https://www.lumenera.com/products/industrial - scientific/ltx2420.html. Accessed: 2022-03-01.

Machine Vision with Python (n.d.). https://openmv.io/. Accessed: 2022-03-01.Sun J., Li B., Jiang Y., and Wen C. (2016). “A Camera-Based Target Detection and Positioning UAV System for Search and Rescue (SAR) Purposes”. Sensors (Basil) 16.11,p. 1778.