Spring 5-1-2022

Autonomous Search and Rescue Helicopter System Design

Terry Zhao
terry.zhao@uconn.edu

Autonomous Search and Rescue Helicopter System Design

Team 2206 Members:

Kevin Joshy, Duy Le, Nathanael Metke, Terry Zhao

Faculty and Thesis Advisor:

Dr. Ashwin Dani

Honors Advisor:

Dr. Krishna Pattipati

Sponsor:

Sikorsky Aircraft

Jason Thibodeau, Siddarth Suresh, Kerry Jones

Abstract

The objective of this project is to design and implement an autonomous search-and-rescue drone system capable of operating on targets positioned within a 30 ft. radius. Targets are located through heat signature, after which the drone performs a controlled approach to retrieve the rescuee and return to the launch point. The Pixhawk 4 autopilot is utilized to run autonomous missions, an IR camera to identify the target through OpenCV, and a winch system to pull the victim to safety. The flight director is designed in MATLAB/Simulink to help guide the drone to the precise location of the target. Simulations of the drone are run using jMAVSim to test software-in-the-loop implementations. At the project's conclusion, several requirements have been fulfilled, while others remain incomplete due to unforeseen obstacles. Although the drone hardware has been assembled and image detection and UAV communication capabilities on the offboard computer are fully functional, the Simulink flight director component of the rescue loop was not completed and test flights of the drone were unsuccessful. The team faced some challenges, such as global chip shortage, limited budget, time constraints, and weight capacity.

Table of Contents

Abstract **Error! Bookmark not defined.**

Table of Contents 2

Glossary 3

Introduction 3

Problem Statement 4

I. Statement of Need 4

II. Specifications 4

Functional Specifications 4

Performance Specifications 5

III. Basic Limitations 5

Approach and Design 5

I. Theories 5

Flight theory 5

Flight Director and Autopilot 6

Rate Controller 6

Computer Vision 7

Thermal Camera 7

Search Algorithm and Motion/Path Planning Algorithms 8

3
uORB Messaging 8
MAVLink Messaging 8
Flight Mission State Machine 9
Rescue Plan 9
Servo Motor 10
II. Models 111
III. Constraints 211
IV. Alternate Designs/Solutions 21
V. Standards 22
VI. Design Results 24
Project Management 26
I. RACI Chart 27
II. Part List and Budget 27
III. Gantt Chart 28
Summary 29
References 29

Glossary

AHRS	Attitude and heading reference system
API	Application Programming Interface
CNN	Convolutional neural networks
COTS	Commercial-off-the-shelf
CV	Computer vision
FLIR	Forward-looking infrared
GPS	Global Positioning System
IR	Infrared
jMAVSim	Simple multicopter/quadcopter simulator
LiPo	Lithium polymer
OpenCV	Real-time optimized computer vision library
PX4	Open-source autopilot system
QGroundControl	Graphical user interface for drone configuration
RTL	Return to launch
UAV	Unmanned aerial vehicle

Introduction

In this project, we utilize the open-source flight control software PX4 to run an autonomous search-and-rescue mission. During the search loop of the operation, the quadcopter engages in a

predetermined survey path to search an area of 30 ft. radius for the target. The onboard infrared camera captures images of the vicinity that are processed using the OpenCV image processing library to identify and track target waypoints that are sent as GPS coordinates to the flight director. The rescue loop then utilizes control system models built within Simulink to initiate the drone's travel to the waypoint and its controlled approach to stably hover above the target. From there, the servo motor is activated and the retractable rescue magnet is dropped to pull the victim to safety.

Problem Statement

I. Statement of Need

The primary objective of this project is to design and implement a drone system design for search-and-rescue applications. The overarching task is to create an algorithm that, once programmed into a UAV, can autonomously guide the system to locate a target and haul it to safety. Although our system design is intended to run on a small quadcopter, it ideally can be adapted for helicopter applications, making it transferrable for realistic search-and-rescue missions. This design would have an integral role in rescue missions set in various hostile environments, whether in war-torn areas or locations of adverse climate and geography, such as deserts or mountains.

In addition, the algorithm designed in this project has the potential to be used in various applications. In the military, drones could be programmed to transport equipment and supplies to and from locations that are inaccessible by other means of travel. In areas of danger, drones are a convenient alternative to other modes of manned transportation that could put lives at unnecessary risk. An algorithm with such capabilities could have uses in the commercial delivery industry. Amazon has been prototyping delivery drones for some time, and our algorithm would have similar functionality. These drones could deliver everyday products to homes and workplaces, reducing foot and motorized traffic in already-congested metropolitan areas. Several humanitarian organizations have adopted the use of drones to transport essentials like medical supplies, food, and water to affected areas. Drones reduce the required manpower and increase the safety of such applications, among other benefits, and if current trends are of any indication, will have a growing presence in our everyday lives.

II. Specifications

Functional Specifications:

Search

- The target must be located using the thermal camera equipped on the drone.
- Potential targets should match human size, shape, and temperature.
- Should use blob detection or edge detection to determine target location.
- Should use a path planning algorithm to traverse graphs from CV algorithms.
- Should be optimized for target approach speed.
- Flight director parameters should be sent from the Raspberry Pi, where CV and path planning calculations are done.

Rescue

- Must maintain a stable hover over the target.
- Must use PX4 for flight director and control system.

- Must use Simulink to design flight director and UAV toolbox for control design.
- Should use jMAVSim for drone simulation.

Performance Specifications:

- The drone is able to autonomously use a camera to identify and track the target.
- The weight of the target must not exceed the carrying capacity
- The drone will have at least one hour of battery life.

III. Basic Limitations

Sikorsky Aircraft requested the team to use a COTS drone, the Holybro S500 with PX4. This drone model can only carry payloads with a max weight of 1 kg. Therefore, with the LiPo battery and other peripherals equipped, the payload must not exceed ~200 g. The sponsor also recommended designing the flight director using Simulink in MATLAB. To simulate our Simulink design, the PX4 toolchain must be installed for jMAVSim integration, and the UAV PX4 supporting package is needed. The drone should have safety shutdown functionality to ensure our safety during flight tests.

Approach and Design

I. Theories

Flight theory

Drones are unmanned aerial vehicles that do not require a human pilot. The commercial drones on the market today are fairly small with four rotors that can be controlled remotely. Drones use rotors for propulsion and control. The motors are similar to small fans that push air to create upward force. The faster the blade spins, the greater the force lifting the drone. These fans can be turned to create different flight motions, such as Throttle, Yaw, Pitch, and Roll.

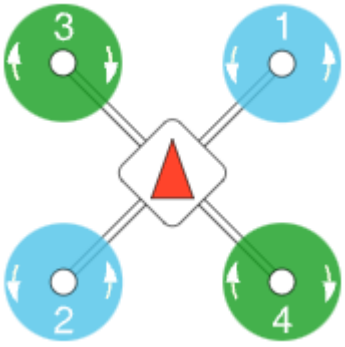


Figure 1: The orientation of a Quad X drone

Throttle - Up and down vertical motion
Yaw - Clockwise and counterclockwise rotational movement
Pitch - Forward and reverse lateral movement
Roll - Right and left lateral movement

For vertical motion, the drone can go upward, hover, or downward. In hovering, the net thrust of the four rotors pushing the drone up must be equal to the gravitational force pulling it down. Then, the force needs to be greater to go upward, and lesser to go downward. For rotating the done, two motors diagonal from each other can rotate clockwise, and the pair of motors can rotate counterclockwise. This causes the total angular momentum to zero. Depending on the direction of rotation, the motors can decrease their angular speed to choose which direction it wants to rotate. For forward motion, the drone increases the rotation rate of the back motors with respect to the forward direction, which creates a larger thrust in the back. Therefore, there is an imbalance in the thrust forces that helps the drone flights in the direction of the lesser thrust. However, we also have to take the weight of the drone and the drag of air into consideration. In this search and rescue project, we are planning to use PX4 autopilot, which is a microcontroller-based flight controller that is connected to the Raspberry Pi. The PX4 will receive data from the search team. The drone will hover over the target and activate the target rescue system.

Flight Director and Autopilot

The role of a flight director in a drone is to utilize the sensors built into a drone in order to compute the proper pitch and bank angles for the drone to stay on the path it intends to travel. In our drone, we intend to have a camera sensor, gyroscope sensor, and depth sensor which will be relaying information to the flight director which will, in turn, be able to use its closed-loop controls to accomplish its mission. The flight director will calculate the desired altitude based on where the drone's programming decides its next location. Given the next location to search, the flight director will calculate the desired altitude and direction to travel in. The flight director will not directly control the drone's flight path, that is the job of the autopilot. The flight director is the brains of the autopilot making the calculations needed to figure out which altitude and direction are needed to reach its destination. The flight director is the calculation tool for the autopilot.

The autopilot system is used in aircraft to provide little to no input from the pilot. When the autopilot is engaged, it is responsible for keeping the aircraft stable on the three-axis. It is also responsible for guiding the aircraft over a particular waypoint. The autopilot reduces the pilot workload, which is extremely important in long-distance flight, and assists in flying the aircraft safely, especially in rough weather conditions. If there is any disturbance in roll, pitch, or yaw, the autopilot is responsible for beginning the aircraft back to its original attitude. The autopilot can also maneuver an aircraft over a particular flight plan with the assistance of control surfaces. In smaller aircraft, such as a drone, the autopilot is coupled with the flight controls directly. Here, the autopilot's function is mainly to keep the aircraft stable. To have the drone fly over a particular route, we can implement the code into the computer, which is a Raspberry Pi. The autopilot uses the information to determine the position of the drone and to adjust the direction or attitude accordingly.

Rate Controller

When using a drone waypoint system, we will be utilizing a rate controller that is used to calculate the needed velocity to start heading to specific coordinates. The further away the drone is from the specified point, the more acceleration will be needed. The closer, the drone will need to start

slowing down to not miss the target. This is classically used with a PID controller. The PID controller will take the X, Y, and Z coordinates as a setpoint, while the controller will correct, as the location will not match. This will allow the drone to fly to different waypoints, without the need for user input. The rate controller is a large part of the autopilot process and creates the error feedback needed for flight correction.

Computer Vision

The various areas of computer vision utilize machine-learning-based algorithms. Image classification aims to automatically classify images into predefined classes. Current models are based on deep CNN. This consists of several convolution layers followed by activation functions and pooling layers, and several fully connected layers before prediction. It comes into a deep structure to facilitate filtering mechanisms by performing convolutions in multi-scale feature maps, leading to highly abstract and discriminative features.

Object detection determines and locates the object instances from either a large number of predefined categories in natural images or a given particular object. Deep learning has substantially advanced the object detection field. Image segmentation focuses on the problem of how the objects are exactly presented in the visual scene.

OpenCV is an open-source library that includes several hundred computer vision algorithms. OpenCV has a modular structure meaning it includes several shared or static libraries. Main library modules include Core Functionality, Image Processing, Video, Video I/O, Calib3d, Features2d, Objdetect, and Highgui. After performing the steps to install the necessary libraries, code can be written to leverage the features provided.

OpenCV offers blob and edge detection algorithms that can be used to extract accurate images of a target through the live camera feed. Blob detection relies on parameters like thresholding, grouping, merging, and center/radius calculation to detect groups of similar, connected pixels. Edge detection extracts the boundaries of objects or regions by detecting sudden changes in pixel intensity. There are two main edge detection algorithms used in OpenCV: Sobel edge detection and canny edge detection. In Sobel edge detection, a Sobel operator detects edges that are marked by sudden changes in pixel intensity. Canny edge detection follows a four-stage process to extract edges from an image: noise reduction, intensity gradient calculation, false edge suppression, and hysteresis thresholding. The culmination of these steps helps to create robust and flexible output images that make this a popular method for image extraction. By using a combination of these two detection algorithms and adjusting certain parameters, accurate images of the target can be developed from various angles.

Thermal Camera

The camera module we selected to do our search is the FLIR Lepton thermal camera. We selected this camera as it had the advantage of having a high resolution for its camera type while still being very lightweight. These factors make it ideal for integration onto our drone. To allow for easy interface with the offboard computer and computer vision algorithms, the camera module is placed in a breakout board, which exposes the I2C, SPI, and power I/O. This I/O can be directly connected

to the offboard computer. The module automatically calibrates to the readings, to allow for easier capturing of thermal images of interest.

Search Algorithm and Motion/Path Planning Algorithms

Search problems consist of a state space (set of all possible states of where one can be), a start state (the state from where the search begins), and a goal test (a function that looks at the current state and returns whether it is the goal state). Search algorithms employ a sequence of actions called a plan that transforms the start state to the goal state (solution). These algorithms can either be classified as uninformed search (used when there is no information about the cost of navigating between states) or informed search (used when we know the cost or have a solid estimate of the cost between states). Path planning algorithms are used to find the fastest or shortest path through a graph of nodes. There are many different approaches to solving this problem, although some are more applicable to the specific situation. For this mission, we will want to path-plan between the starting location and the perceived location of the target. In select instances, there may be impending obstacles, while in others, there may be many potential targets with varying confidence values. Our goal is to avoid these obstacles and maneuver to the correct target location. We will discuss the advantages and disadvantages of the different algorithms, and discuss our potential to use each.

Dijkstra's algorithm is used to find the shortest path between nodes in a graph. A single node is designated the "source" node and the shortest paths from the source to all other nodes are found according to aggregate edge costs to produce a shortest-path tree. These optimized paths are calculated by determining those with the minimum total edge costs between the source and node in question.

A* is an algorithm that is similar to Dijkstra's Algorithm and is even considered an extension of it. It has better performance than Dijkstra's with a more complete optimization process. It is a heuristic algorithm, with a focus to minimize the cost needed to take a certain path summed with the cost of the next node from the starting node. Although faster than Dijkstra's, a downside is that performance is based on the accuracy of the heuristic algorithm used to compute the function.

uORB Messaging

uORB is an asynchronous publish-subscribe messaging API used for inter-thread and inter-process communication. The API is used as an internal communication mechanism for PX4, allowing users to read and write specific messages to the component drivers onboard the drone. These messages are divided into topics that encapsulate the various aspects of the drone. For this mission, the Simulink flight director communicates through the uORB topics actuator_armed, vehicle_gps_position. and vehicle_status. The purpose of reading and writing messages defined under these specific categories is described in the Models section.

MAVLink Messaging

MAVLink is a lightweight messaging protocol for communication with drones and drone components. Part of the PX4 middleware, the protocol is to communicate with QGroundControl, and as an integration mechanism for connecting to offboard systems like the Raspberry Pi

companion computer. MAVLink follows a hybrid publish-subscribe and point-to-point design pattern, where data streams are sent and published as topics while configurations are point-to-point with retransmission. Messages are defined within XML files where the message set supported by a particular MAVLink system is defined. The C reference implementation used is a header-only library that is highly optimized for resource-constrained systems with limited RAM and flash memory. Throughout the mission, MAVLink is used specifically to communicate between the Raspberry Pi and PX4 controller. In addition to the standard messages and microservices that are defined, MAVLink allows users to create custom messages that can be built into programming-specific libraries. Communication from the Raspberry Pi is implemented through MAVSDK, a collection of libraries that provide an API for managing vehicles, providing programmatic access to vehicle information and telemetry, and control over missions, movement, and other operations. Communication from the Simulink flight director is implemented using a combination of Simulink blocks and MATLAB functions from the UAV toolbox. The specific MAVLink interactions between the offboard computer and Pixhawk 4 are described in the Flight Mission State Machine section.

Flight Mission State Machine

The search-and-rescue operation is organized into a sequence of states that coincide with the various drone flight mode transitions taken throughout the flight's duration. The state machine contains the following states: initialization, takeoff, survey, mission, controlled approach, and return-to-launch. The controlled approach state represents a multi-step process and is further divided into the states descend, hover, deploy, and verify.

Because the Raspberry Pi and Simulink flight director share control of the drone's flight throughout the various states, it is necessary to mirror and synchronize the state machine on both platforms. The implementation is done through C++ enumerations and a series of case statements on the Raspberry Pi and in Simulink Stateflow on the flight director. As described in earlier sections, the Raspberry Pi utilizes MAVSDK and the Simulink flight director uses uORB messaging to guide the drone. State machine synchronization is done using the custom MAVLink messaging scheme mentioned previously to ensure both entities are aware of the drone's current flight mode. During initialization, the Raspberry Pi arms the drone actuators and uploads the predetermined survey mission waypoints to the PX4. The initialization block of the Simulink Stateflow repeatedly checks whether the actuators have been armed and only afterward transitions to the takeoff state and sets the vehicle's vertical velocity to initiate the ascent. The flight director simultaneously sends custom MAVLink messages to the Raspberry Pi signaling its transition to the takeoff stage. The Simulink takeoff block transitions to the survey state only after a uORB read has determined the drone has reached the designated altitude. The survey block repeatedly sends custom messages to the Raspberry Pi signaling its transition to the next state. Having entered the survey stage, the Raspberry Pi initiates the survey mission using the waypoints uploaded to the PX4 during initialization. The drone proceeds through the search loop, processing infrared images captured during the survey and extracting the GPS coordinates of the detected target waypoint. From there, the Raspberry Pi transitions to the mission state and initiates the drone's horizontal travel to the target. Custom messages are sent to the flight director signifying the completion of the survey.

Once the mission is complete and the drone is hovering above the target, the Raspberry Pi transitions to the controlled approach and sends custom messages to the flight director signaling this transition. The drone's descending velocity is set through a uORB write within the descend block. The Raspberry Pi continuously reads from the onboard ultrasonic sensor until the drone descends to a reasonable distance above the target. At the same time, the descend block repeatedly reads from the vehicle_gps_position uORB topic and writes the hover navigation state to the corresponding uORB topic once the designated hover height has been reached. Transitioning to the hover block, the vertical velocity is set to zero through a uORB write and the flight director transitions to the drop state while notifying the Raspberry Pi of this change. In this stage, the PWM block from the PX4 Autopilots package is used to send signals to the servo motor, unwinding and winding the rope to drop the magnet and pick up the target. As the flight director moves onto the verify state, custom messages are sent to the Raspberry Pi, initiating the process on the mirroring state machine. Here, the Raspberry Pi processes images captured through the live infrared camera feed, scanning the area underneath the drone to determine whether the target has been retrieved. If not, both state machines revert back to the drop stage and the servo motor implementation is rerun. Once the retrieval is successful, the drone transitions to return-to-launch, where the flight director uses the initial takeoff coordinates saved in a previous state to guide the drone through uORB writes.

Rescue Plan

Our implementation for rescuing the target begins by first setting a waypoint above the target and instructing the drone to move towards the waypoint at a set point. Once the drone is hovering over the target it descends until the distance sensor is triggered. Once triggered, the winch is deployed. A string with neodymium magnets is lowered to the target until it is secured to the target which has its own magnet. Once attached to the string, the target is retracted until the target is acquired. After verifying that the target is picked up with OpenCV our drone will return to launch.

Servo Motor

Figure 2 shows our Simulink model for the servo motor. This model is set to activate once the distance sensor is triggered. By sending a PWM on duration of 0 to 2000 milliseconds we are able to turn the servo clockwise with 2000 being the fastest rotation. Likewise for counterclockwise rotation, we use a duty cycle of 2000 to 4000 depending on the speed needed. Our servo motor does not support feedback therefore we need to use speed and duration in order to calculate the distance traveled. Using this we are able to determine how long to run the servo motor to attach the magnet to the target.

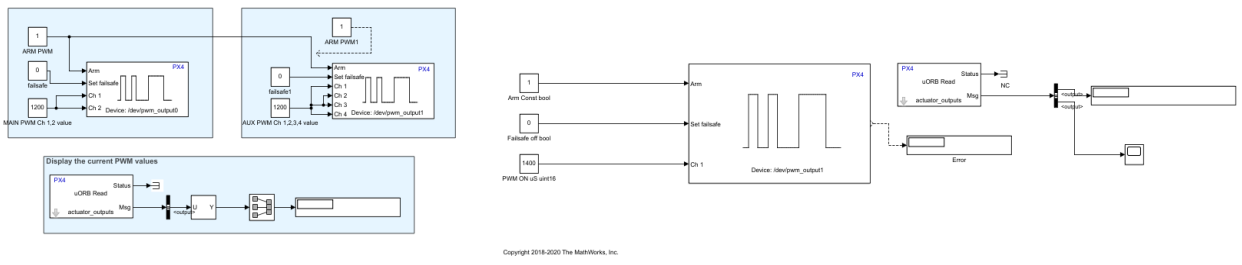


Figure 2: PWM Simulink model

II. Models

To design our drone model, we first created a high-level drone block diagram. In figure 3, the diagram separates into multiple blocks, but the Raspberry Pi and PX4 Controller are our most important blocks. We also added other blocks to indicate sensors that attach to the drone, including an IR camera, GPS module, and distance sensor. It is important to note that the GPS module, gyroscope, and accelerometer are included in the Pixhawk 4 flight controller.

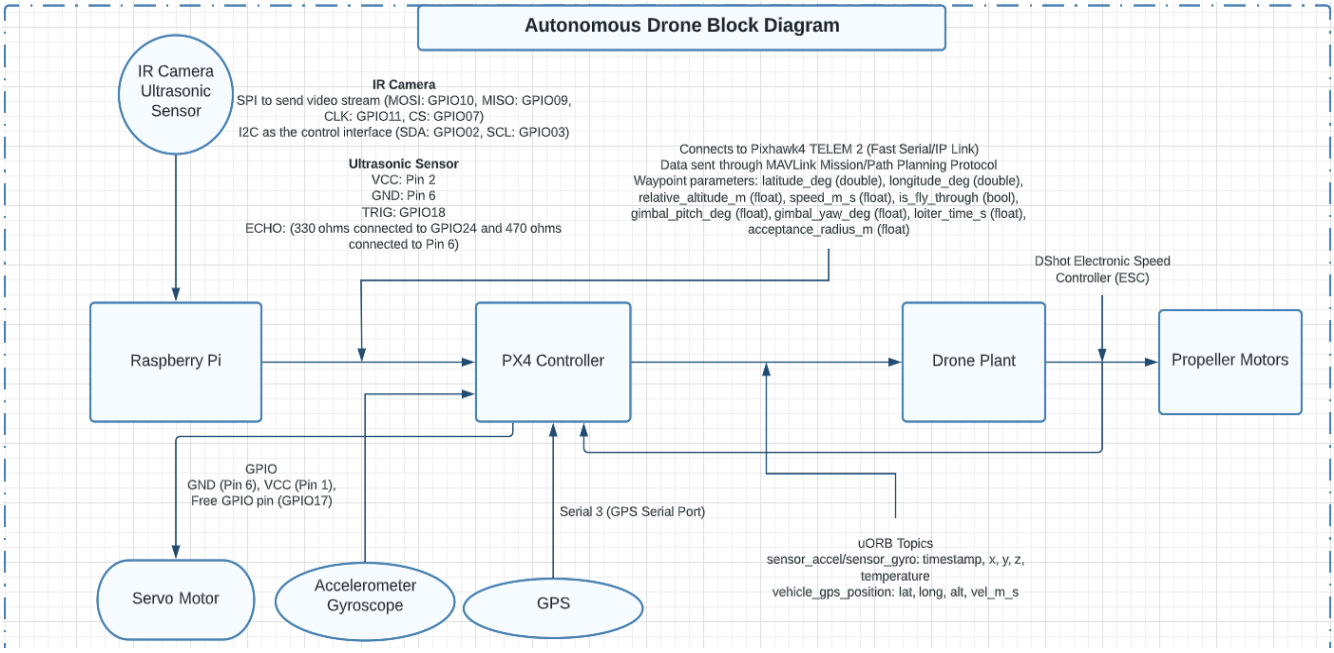


Figure 3 : Block diagram of the overall design of the search and rescue algorithm

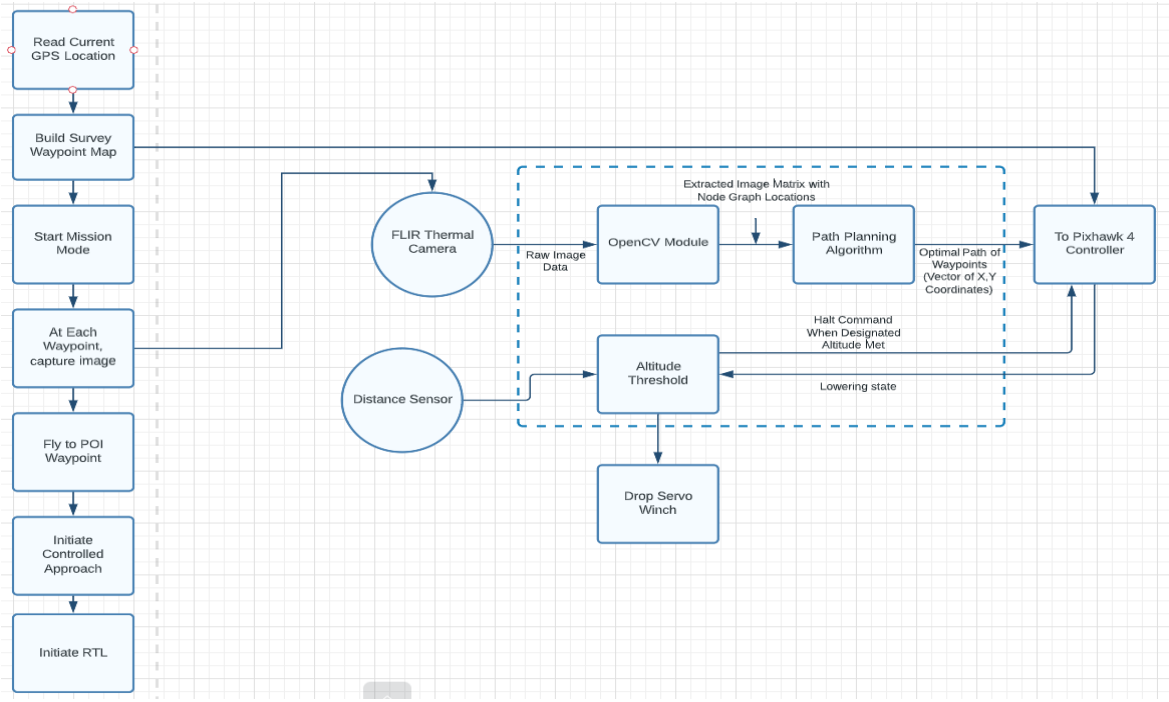


Figure 4 : The subsystem of the search mission in Raspberry Pi block

The block diagram in Figure 4 illustrates the state machine implementation inside the Raspberry Pi block. The state machine completes the flight sequence from takeoff, mission, controlled approach, and return-to-launch. The Raspberry Pi sends waypoints to the PX4 that take the form of a spiral survey path. The camera then identifies the target using IR and converts the image with the target signature into GPS coordinates. The Raspberry Pi also uses the distance sensor to monitor the distance from the drone to our target, making sure that the drone is at the ideal distance during the controlled approach state.

In our Simulink portion, we also formulated a block diagram (Figure 5) for the flight process that matches the state machine in Figure 4. The flight sequence is similar except the Simulink model mainly handles takeoff, mission, controlled approach, and return-to-launch. The controlled approach process has multiple smaller steps to safely rescue our target. During controlled approach, the drone needs to be directly above our target. The drone will then slowly descend to the suitable height and begin to release the rescue system. After pulling up the target, we will then verify if the target has been successfully picked up. The drone will then continue to return-to-launch and end the flight mission.

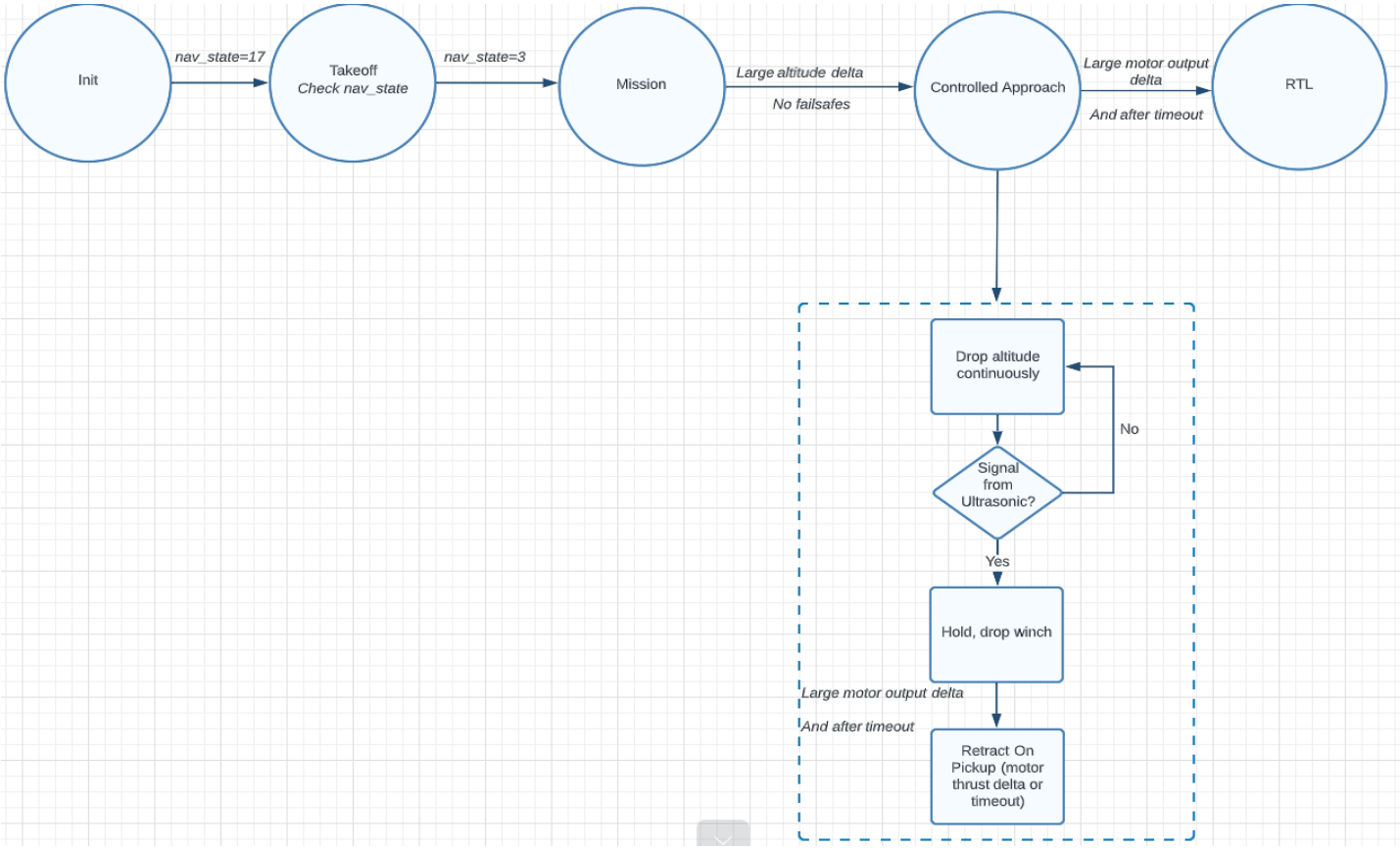


Figure 5: The block diagram of the flight director in Simulink

For the main Simulink model, our goal was to design a flight director that can autonomously guide our drone in the right trajectory and adjust the roll, pitch, and yaw of the drone. We separated the design into two main parts: Stateflow and Drone Simulation (Figure 6). After getting suggestions from our sponsor, we decided to use the Stateflow to monitor the states in controlling the drone. The Stateflow would then send commands to the Drone simulation block to control its movement.

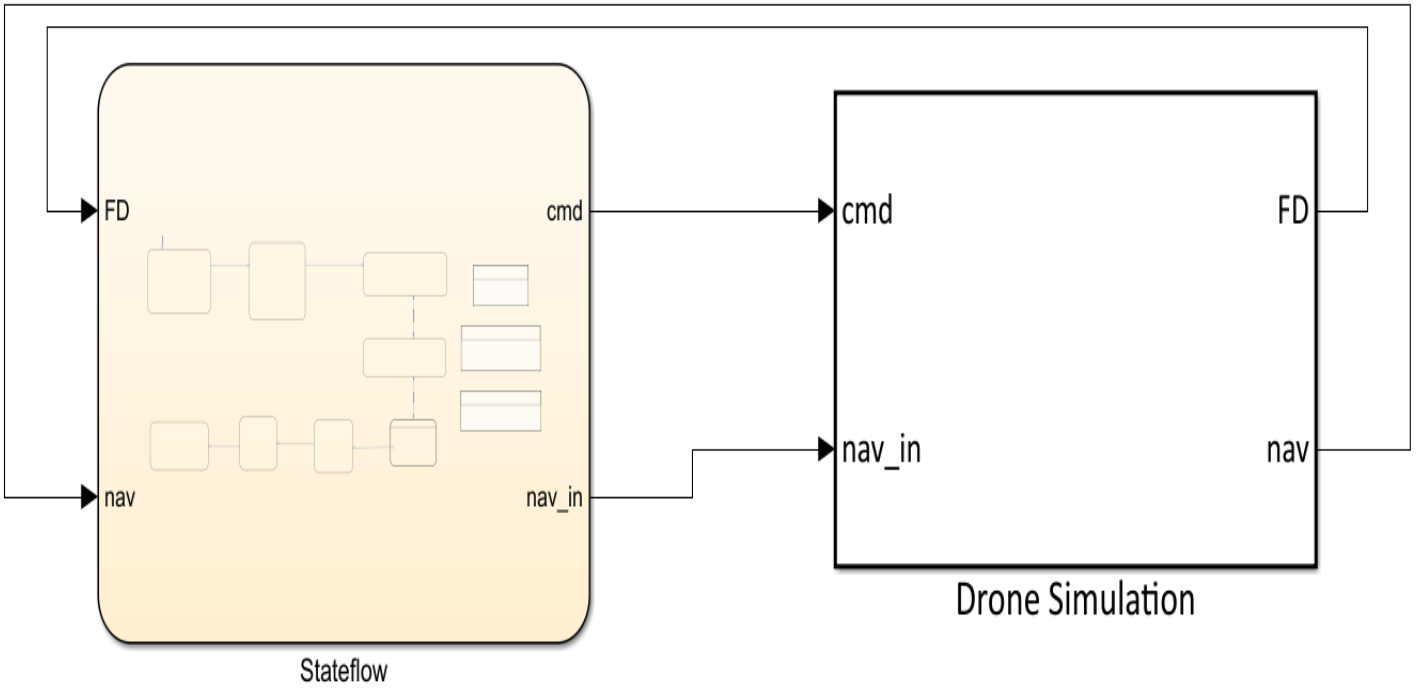


Figure 6: Simulink model of the flight director

In Figure 7, all states of the Stateflow block are connected to create the flight procedure. The flight procedure is very similar to the steps in Figure 5, but we added a few additional states in our Stateflow. After the takeoff state, we had the drone set the initial position to later return to. The drone would stop hovering above the ground to set its base. It is important to note that the longitude, latitude, and velocities in all directions should be zero, except for the altitude or the height. We ideally would want our drone to be 10 meters above the ground before continuing with its mission. To transition to the next state, the current state needed to satisfy certain conditions that were set out for them using the parameters in uORB topics. In this model, we mainly used the uORB parameters to set conditions, such as altitude from vehicle_gps_position and navigation state from vehicle_status.

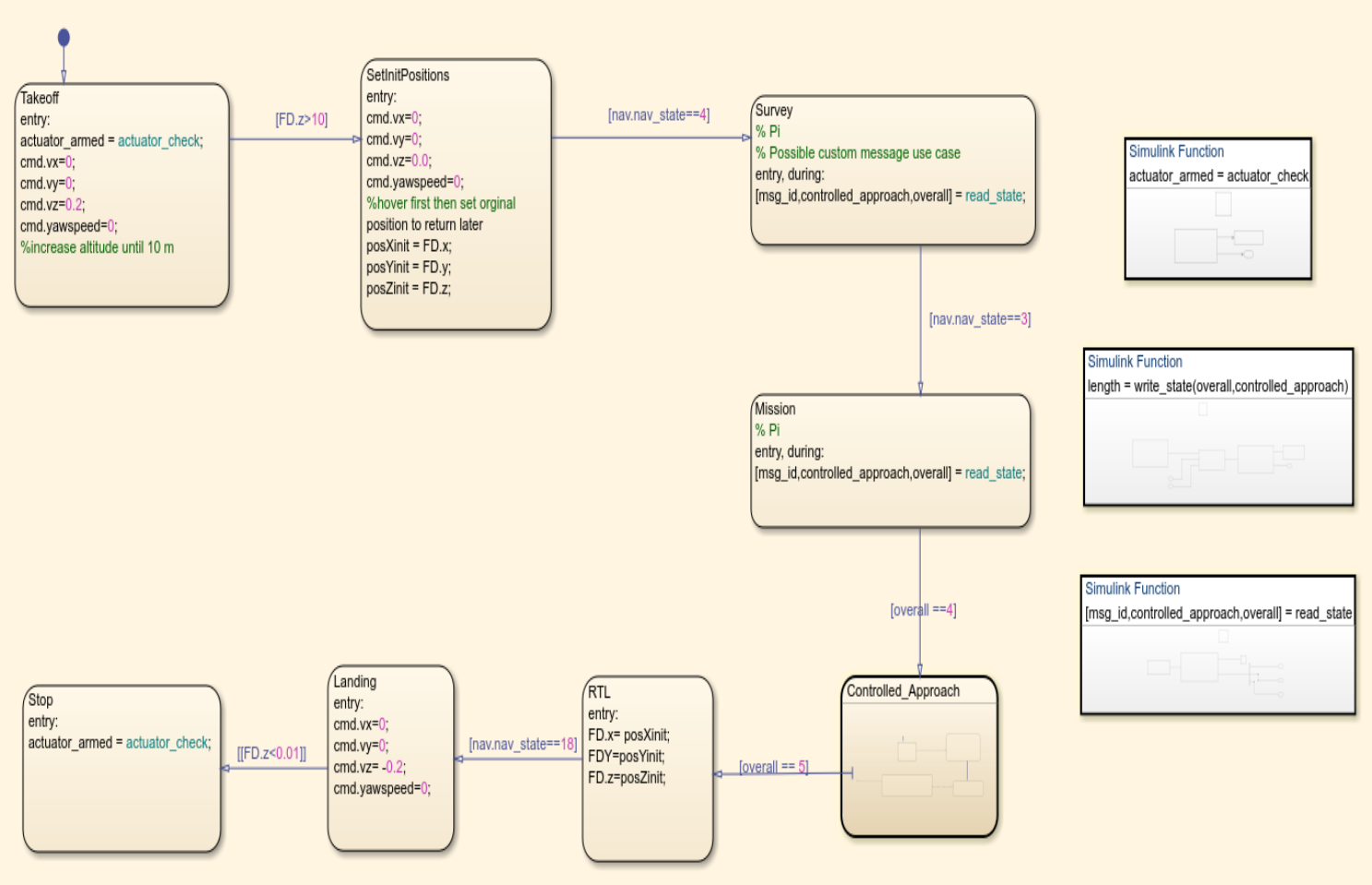


Figure 7: All flight states and functions inside out Stateflow block

In order to communicate with the state machine in MAVSDK, we also created a Simulink function to transition between the states survey, mission, and controlled_appoach since these states were handled by the image processing in the Raspberry Pi. Figure 8 shows the custom MAVLink messages to create commands to arm/unarm the drone, the length of the custom message, and the overall state of the vehicle. Each function could be called to evaluate the current status of the state machine or to create a condition to transition to the next state.

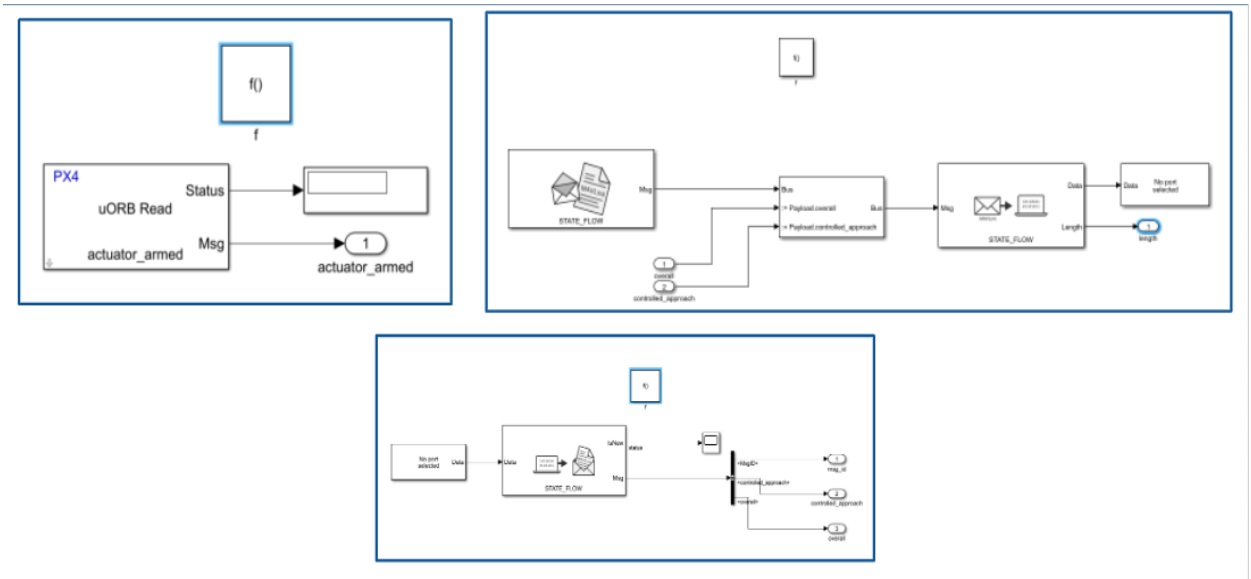


Figure 8: Functions to communicate with Raspberry Pi using MAVLink custom messaging

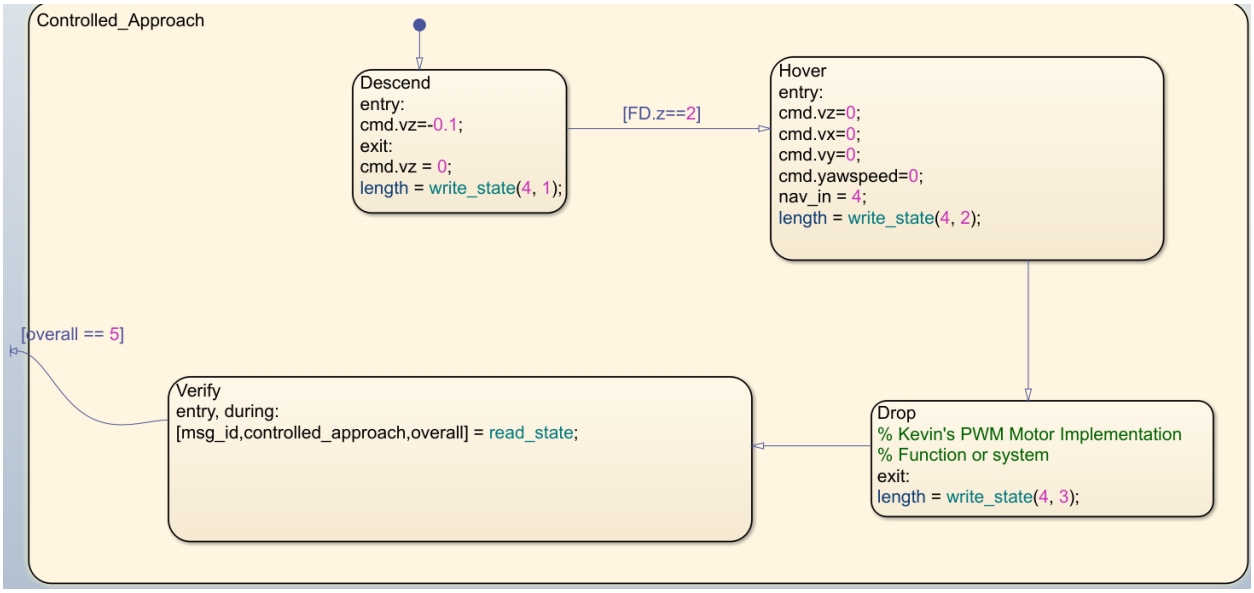


Figure 9: Sub-states inside the Controlled_Approach state

The Controlled_Appoach state has multiple sub-states in order to successfully rescue our target. During the controlled approach, the drone should be directly above our target. The drone starts by descending to an ideal height of ~2 meters. It should then stop and start hovering above our target. The rescue system should engage at this moment and release the magnet that would attach to our target. After successfully pulling up our target, we would then verify the target using our camera, ascend, and continue to the next state. Figure 9 shows each sub-state inside our Controlled_Approach state.

In the Drone Simulation, we used the PX4 UAV Supporting Package to design our model. The PX4 component utilized different uORB topics, which is an asynchronous publish() / subscribe() messaging API used for inter-thread/inter-process communication. We used the vehicle_gps_position and vehicle_status for our uORB read and write. The vehicle_gps_position and vehicle_status gave different parameters below:

vehicle_gps_postion:

lat	# Latitude in 1E-7 degrees
lon	# Longitude in 1E-7 degrees
alt	# Altitude in 1E-3 meters above MSL, (millimeters)
vel_m_s	# GPS ground speed, (meters/sec)
vel_n_m_s	# GPS North velocity, (meters/sec)
vel_e_m_s	# GPS East velocity, (meters/sec)
vel_d_m_s	# GPS Down velocity, (meters/sec)

vehicle_status:

nav_state	# set navigation state machine to specified value

These parameters enabled us to set the coordinates and velocity of the drone. The outputs were looped back to our Stateflow to continuously update the flight path. Figures 10-14 show the design of our Drone Simulation using uORB topics.

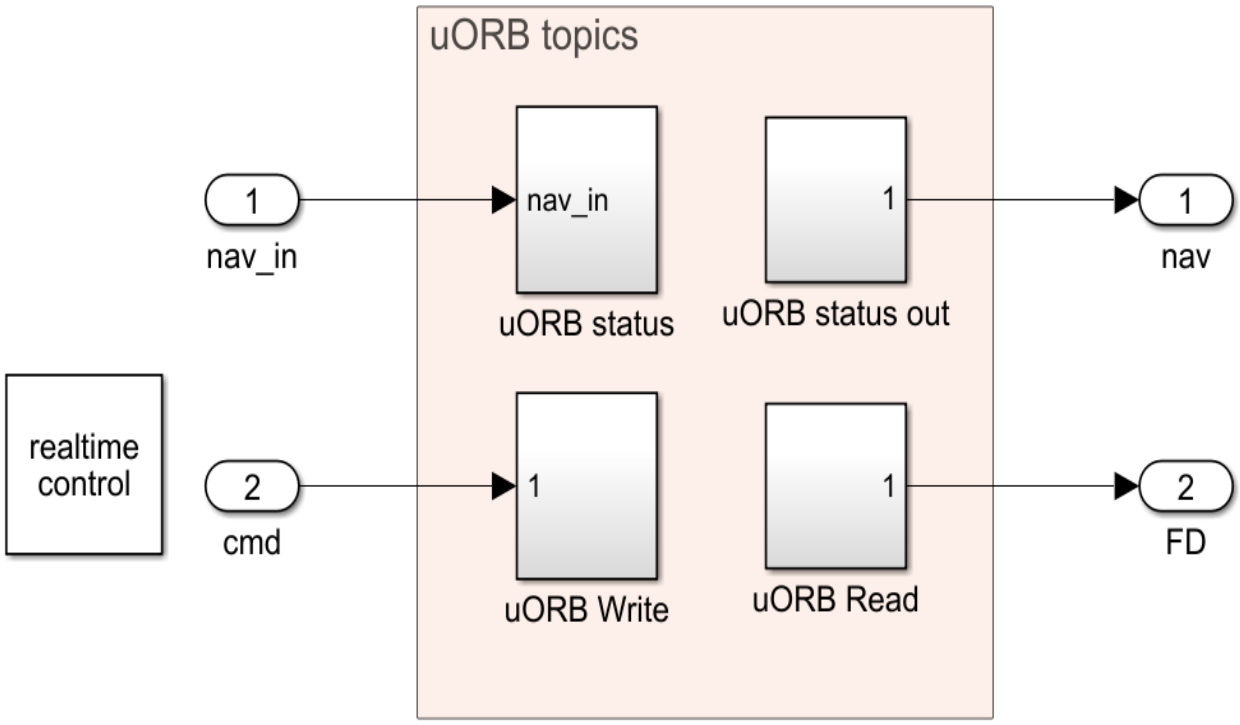


Figure 10: uORB blocks inside Drone Simulation

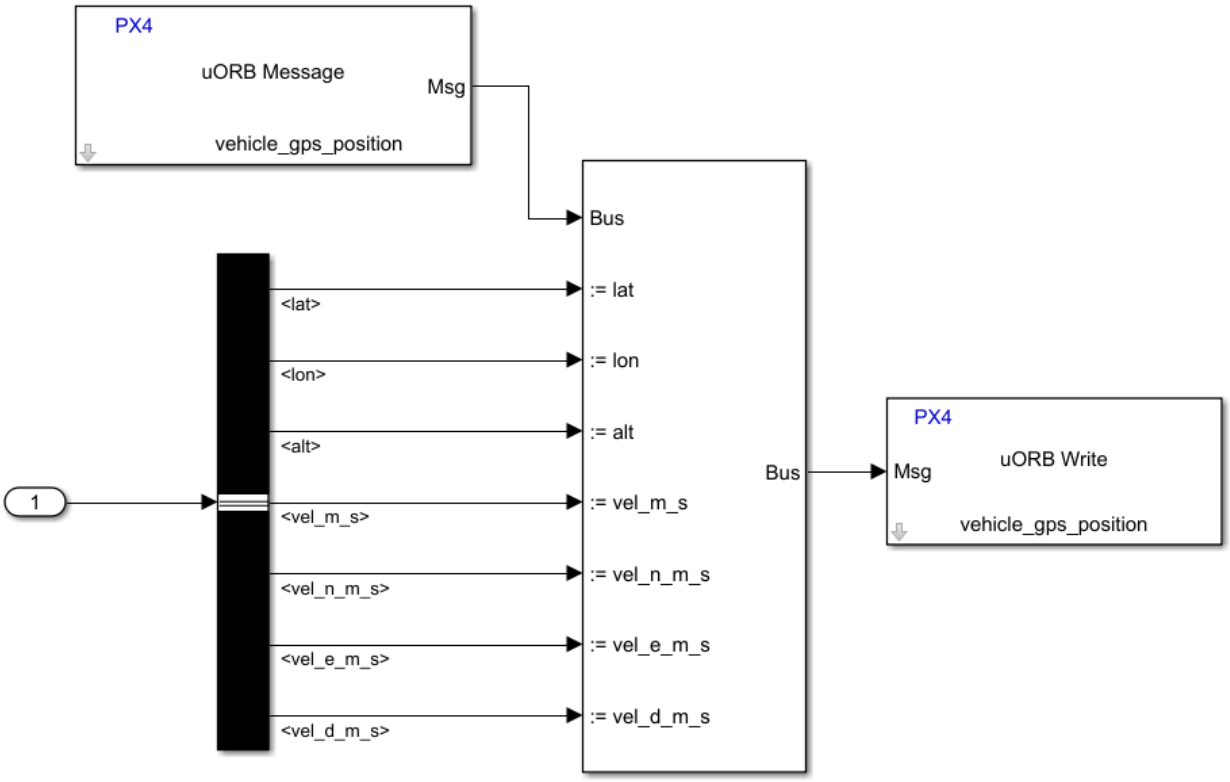


Figure 11: uORB write using vehicle_gps_position

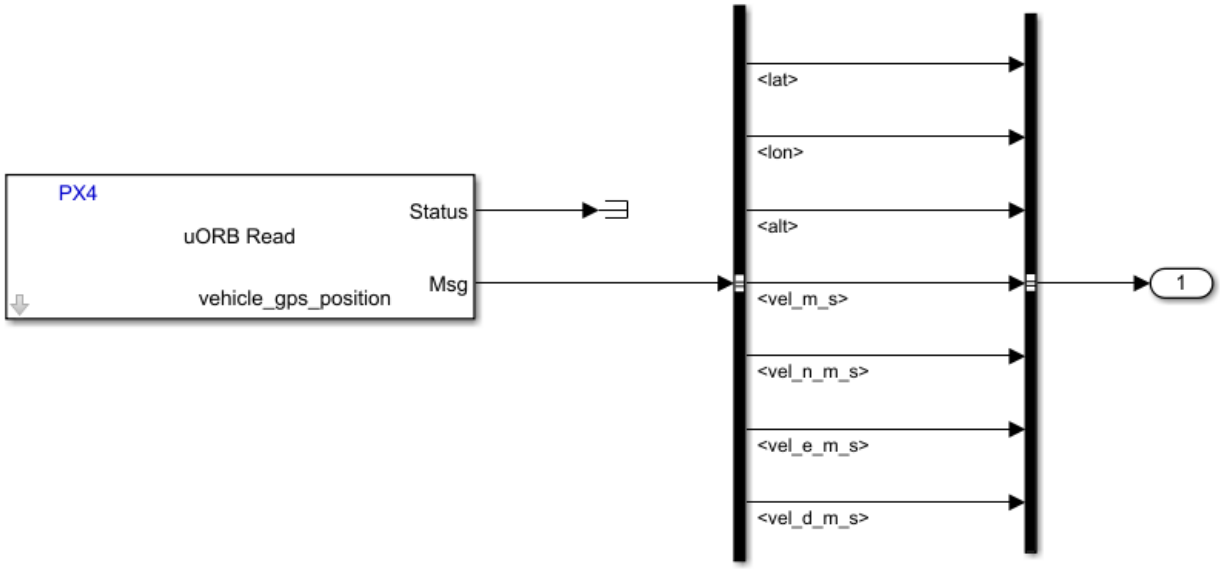


Figure 12: uORB Read using vehicle_gps_position

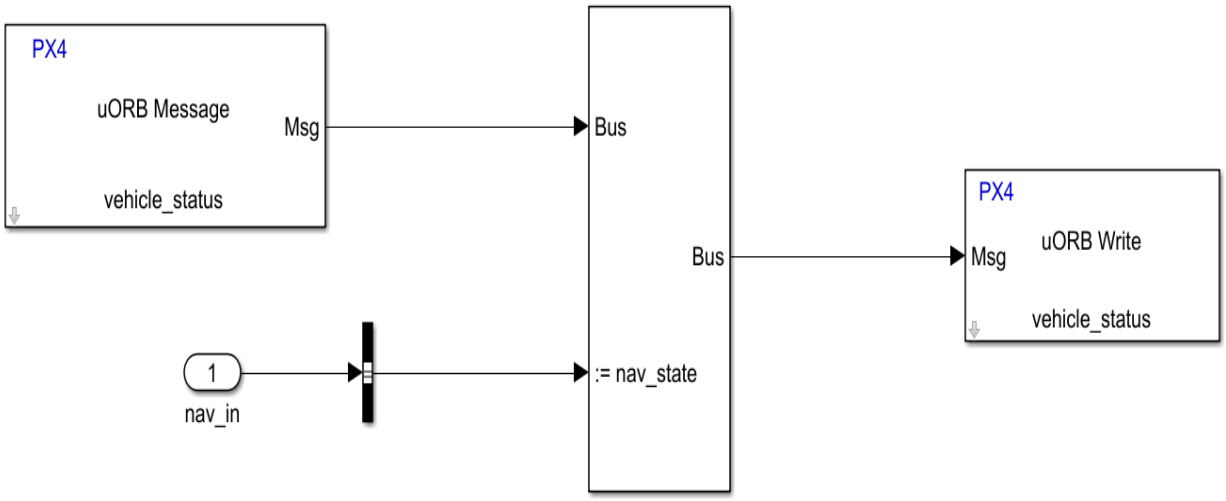


Figure 13: uORB Write using vehicle_status

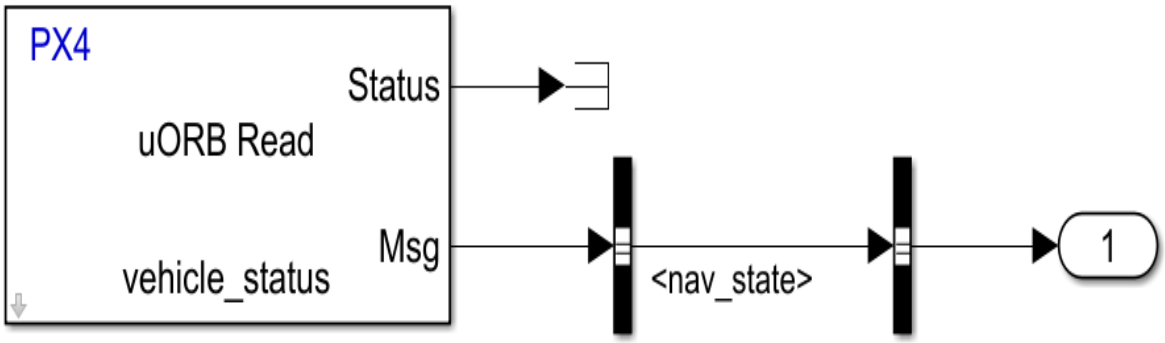


Figure 14: uORB Read using vehicle_status

To run drone simulations, we first needed to install PX4 Toolchain to run jMAVSim, which is a simple multirotor/Quad simulator. The toolchain needed to be version 0.8 to be compatible with the Simulink model. We needed to install QGroundControl to provide full flight control and to easier monitor the drone during our flight test. The PID in QGroundControl also needed some tuning to successfully fly without crashing. Figure 15 shows our simulated drone performing a simple take-off in the jMAVSim window. We then proceeded to send the spiral survey path to the drone. In Figure 16, the QGroundControl windows on the left show the simulated drone following waypoints in our spiral pattern. The jMAVSim window on the right shows our drone flying in the air and following its desired flight path.

Figure 15: Simulated drone takeoff using jMAVSim

Figure 16: Drone simulation running in jMAVSim with spiral survey area (left) and real-time simulated drone flight (right)

III. Constraints

One of our biggest constraints is time. Because the project contains many components of varying complexity, time management is essential to ensure that everything is completed. A complete and synchronized team effort is needed to ensure contributions are communicated to all participants so that component integration is seamless and rework is minimized.

Our budget is another constraint. With a budget of $1000, we had to carefully choose our parts. In addition, we were constrained by the ongoing supply chain issues with computer chips. As a result, we had to pay more and wait longer for specific parts like the drone, offboard computer, and IR camera to ensure that we could use them for this project.

Our sponsor provided us with additional constraints. We had to use a COTS drone that integrates with the PX4 flight control system. In the scope of the project, this limited our choice of a solution to the task. This dictated our decision to pick the Holybro x500 drone kit.

IV. Alternate Designs/Solutions

Alternatively, the drone simulation can be done using Gazebo instead of jMAVSim. However, Gazebo SITL is not supported by Windows. jMAVSim is a better option since most of our team members and the senior design desktop are windows.

The flight director can be designed using C++, but this way is considerably more difficult than using Simulink. Our sponsors also recommended that the design should be on Simulink. It is also important to note that only two of our team members have a strong knowledge of C++.

In our first design, we were going to use the Holybro X500 model instead of our current drone model S500. The X500 was out of stock, and the global chip shortage affected the process of back ordering and delivery time. Therefore, we decided to switch to the S500 model, which is an older model but does not affect our design. However, we managed to find a way to use a different x500 kit, with additional parts from other locations to move back to our original design.

V. Standards

Aircraft Requirements
Your UAS must weigh less than 55 pounds (including all items on board or attached to the aircraft)
Your UAS must be registered with the FAA at https://registermyuas.faa.gov/ if it weighs more than 0.55 pounds. You must be 13 years of age or older and must be a U.S. citizen or legal permanent resident to register your UAS with the FAA.
You must label your UAS with the FAA-provided registration number
Your UAS must be capable of sustained flight and in a condition for safe operation
Pilot Requirements
Pilots must be at least 16 years of age and hold a remote pilot airman certificate with a UAS rating or be under the direct supervision of a person who does. The remote pilot in command must be able to immediately direct control of the UAS if he or she is supervising the operation of the UAS. See https://www.faa.gov/uas/getting_started/fly_for_work_business/becoming_a_pilot/.
The remote pilot must ensure that he or she, the person manipulating the controls, and visual observer are able to safely carry out their responsibilities. The remote pilot must ensure the UAS poses no undue hazard to people, aircrafts, or property if there is a loss of control of the UAS.
No one may serve as a remote pilot, person manipulating the controls, visual observer, or other crewmember if he or she: • Has consumed any alcoholic beverage within the preceding 8 hours; • Is under the influence of alcohol; • Has a blood alcohol concentration of .04 percent or greater; • Is using a drug that affects the person's mental or physical capabilities (which may include over-the-counter medications); OR • Has any other medical condition that creates a risk to operations (e.g., epilepsy).
Operational / Safety Requirements
Do not fly faster than a groundspeed of 87 knots (100 miles per hour)
Fly at a maximum altitude of 400 feet above ground level (AGL) or, if higher than 400 feet AGL, remain within 400 feet of a structure
Do not fly if minimum weather visibility is less than 3 miles from control station
Stay 500 feet below clouds and no less than 2000 feet horizontally from the clouds
Maintain visual line-of-sight (VLOS) over the UAS that is unaided by other devices. The UAS must remain within VLOS of the remote pilot in command and the person manipulating the flight controls of the UAS. Alternatively, the UAS must remain within VLOS of the visual observer.
Do not operate UAS over any persons unless they: • Are directly participating in the operation; OR • Are under a safe cover, such as a protective structure or a stationary vehicle.
No operation at night. UAS operations during civil twilight is permitted with appropriate anti-collision lighting. Civil twilight is 30 minutes before official sunrise and 30 minutes after official sunset.

Perform safety risk assessment and pre-flight check to (at a minimum): • Confirm that the UAS is in a condition for safe operation; • Conduct an assessment of the operating environment (local weather conditions, local airspace and any flight restrictions, location of persons and property on the surface, and other ground hazards); • Ensure that all persons directly participating in the UAS operation are informed about operating conditions, emergency procedures, contingency procedures, roles and responsibilities of each person involved in the operation, and potential hazards; • Ensure that all control links between the control station and the UAS are working properly; • Ensure there is sufficient power to continue controlled flight operations to a normal landing; • Ensure that any object attached or carried by the UAS is secure and does not adversely affect the flight characteristics or controllability of the aircraft; and • Ensure that all necessary documentation is available for inspection, including the remote pilot certificate and aircraft registration.
Autonomous operations are permitted if the remote pilot in command retains the ability to direct the UAS to ensure compliance with the requirements of Part 107.
Always remain clear of and yield right of way to manned aircraft.
Additional prohibitions: • No person may act as a remote pilot in command or visual observer for more than one unmanned aircraft operation at one time • No operations from a moving aircraft • No operations from a moving vehicle unless the operation is over a sparsely populated area • No careless or reckless operations • No dropping of objects that creates a hazard • No carriage of hazardous materials
Location Requirements
Operations in Class B, C, D and E airspace require air traffic control permission. For more information see https://www.faasafety.gov/gslac/ALC/course_content.aspx?cID=42&sID=505&preview=true; and https://www.faa.gov/uas/where_to_fly/airspace_restrictions/.
Operations in Class G airspace do not require air traffic control permission. For more information see https://www.faasafety.gov/gslac/ALC/course_content.aspx?cID=42&sID=505&preview=true; and https://www.faa.gov/uas/where_to_fly/airspace_restrictions/.
Download B4UFLY app at https://www.faa.gov/uas/where_to_fly/b4ufly/ to determine whether there are any restrictions or requirements in effect at the location where you want to fly.
Check for NOTAM temporary flight restrictions before flying: https://pilotweb.nas.faa.gov/PilotWeb/
Reporting Requirements
You must report to the FAA within 10 days of any flight that results in: • Serious injury or loss of consciousness; OR • Property damage greater than $500. See Part 107 for reporting details. Certain accidents must also be reported to the NTSB.

Table 1: UConn standard for the use of Drones/Unmanned Aircraft Systems

VI. Design Results

As discussed in the following section, our project approach is divided into several phases:

- Project conception and initiation
- Research and experimentation
- Project design, analysis, and validation
- Project building and prototyping

To gain a better understanding of the project requirements, we devoted time to learn concepts involving flight direction and control, path-planning, and computer vision algorithms, in addition to getting familiarized with the OpenCV library, Simulink, PX4 jMAVSim toolchain, and QGroundControl.

Designing a flight director in Simulink provided some unique challenges, both with viewing simulator data, and ensuring that signals we sent were properly formatted. At this point, we are able to send commands to a simulator, however, there needs to be a lot of tweaking of PX4 signal values, as the drone is unable to make a safe takeoff.

To detect and formulate a target image from the camera, we decided to use a combination of the blob and edge detection algorithms within OpenCV. Before receiving our FLIR Lepton camera, we ran some tests on test images to ensure that we knew what parameters to change and adjust. The important parameters were shape, gathered from the canny edge detection, as well as color saturation, blob area, and pixel value threshold. We got inconsistent results with the raw image but were able to clear up the analysis only by looking at the red channel of the returned image.

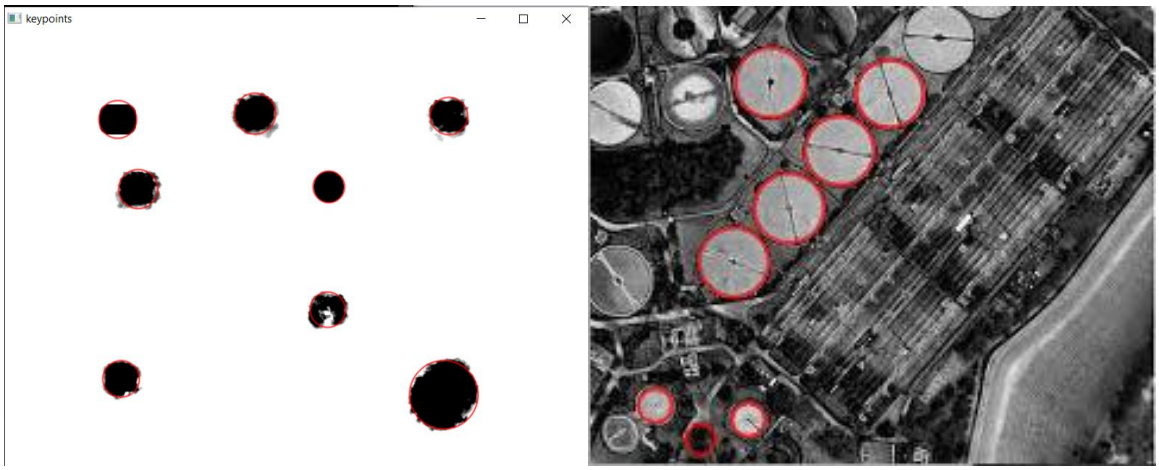


Figure 17: Early blob detection, outputs relative location data

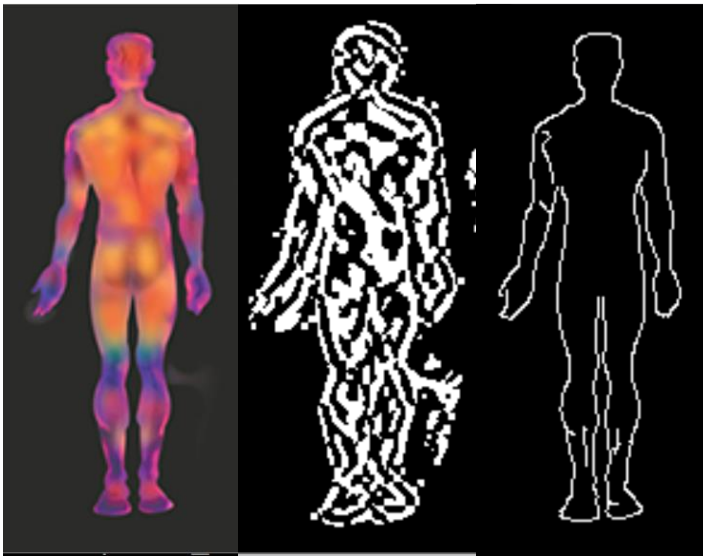


Figure 18: Early Edge Detection from Original Image to Sobel XY Edge Detection to Canny Edge Detection

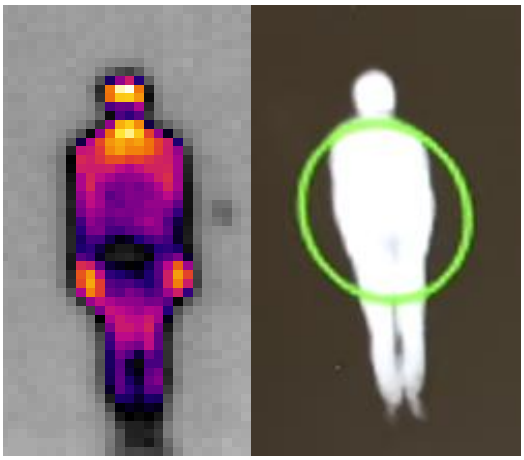


Figure 19: Final Object Detection using FLIR Lepton Camera Module

After researching path-planning and computer vision algorithms, and the OpenCV library, we decided to use either Djikstra's or A* search algorithm. These are simple algorithms, allowing us to find the optimal path between nodes. As seen in the images above, our blob detection algorithm can find points of heat and similar pixel values. This can be easily tweaked and refined to match our target.

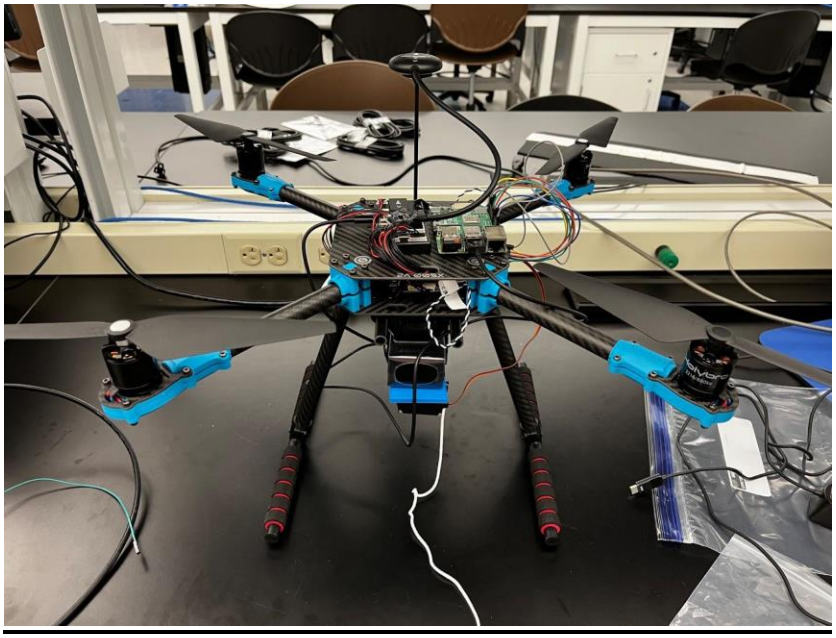


Figure 20: The completed hardware drone

Our team worked to put together the X500 drone. We soldered the motors to the power management board, which is connected to the PX4. The camera was mounted to the bottom of the drone to directly capture images during flight. We 3D-printed the servo motor rod and brackets for the rescue implementation.

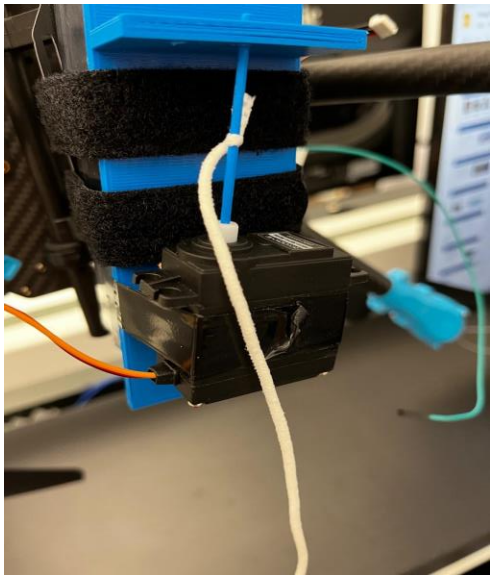


Figure 21: The servo motor and mount of the rescue system

Project Management

I. RACI Chart

Project Deliverable	Duy Le	Kevin Joshy	Nathanael Metke	Terry Zhao	Dr. Ashwin Dani
Team Management	C	C	R	C	I
Path Planning	C	C	R	R	I
Computer Vision	C	C	R	R	I
Flight Director	R	R	R	R	I
Control System	R	R	C	C	I
PX4 Implementation	R	R	R	C	I
Hardware	R	R	R	R	I
Testing	R	R	R	R	I

Table 2: RACI indicates each member's responsibilities

Our project is split into two main sections. The search section and rescue section. The search section is composed mainly of the computer vision to gather data from the infrared sensor and utilize OpenCV in order to gather targets that will be used to create a path for the drone to travel in. This path will be created using the path planning algorithm. The rescue portion of the project is to design a flight director capable of performing a controlled approach on a target.

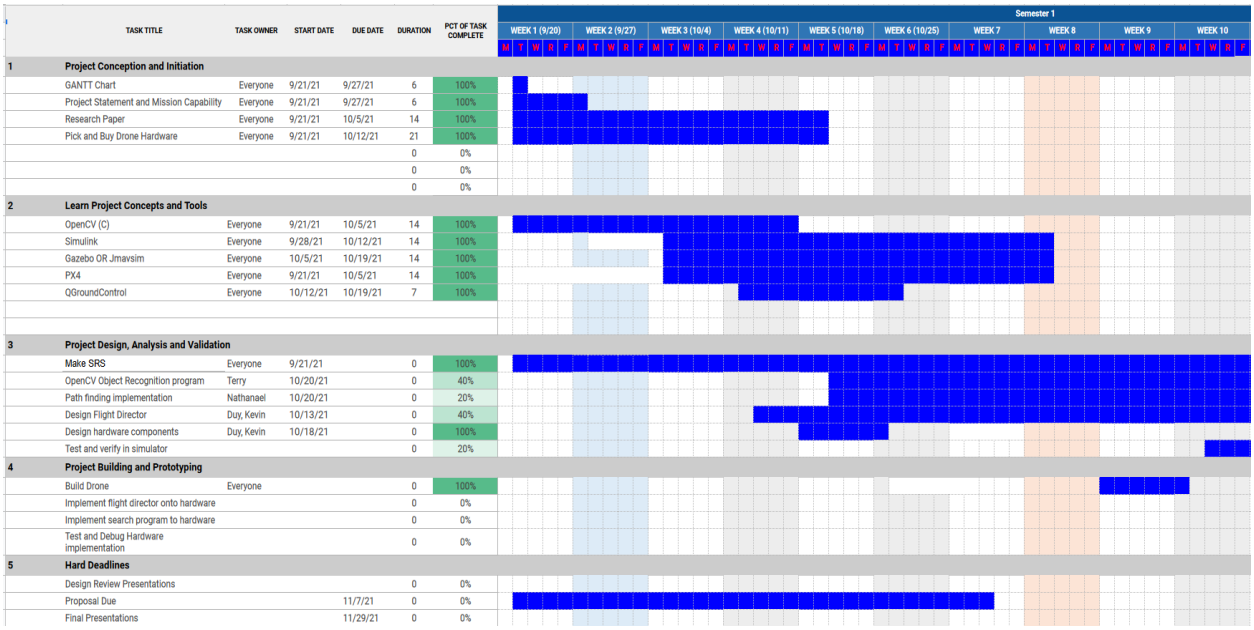
II. Part List and Budget

Name	Code	Price
Holybro x500	30125	$236.00
Flight Controller	20034	$236.00

LiPo Battery	8940	$89.99
Doll (Payload)	N/A	$10.00
Gyroscope/Accelerometer	2738	$39.95
Camera	500-0771-01-ND	$199.00
Magnets	B08L7KFL6Z	$9.99
Distance Sensors	3317	$14.99
Raspberry Pi	4295	$55.00
Servo motor	B01HSX1IDE	$14.97
Total		$905.89

Table 3: list of components and costs

III. Gantt Chart



	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE
1	**Project Conception and Initiation**					
	GANTT Chart	Everyone	9/21/21	9/27/21	6	100%
	Project Statement and Mission Capability	Everyone	9/21/21	9/27/21	6	100%
	Research Paper	Everyone	9/21/21	10/5/21	14	100%
	Pick and Buy Drone Hardware	Everyone	9/21/21	10/12/21	21	100%
					0	0%
					0	0%
					0	0%
2	**Learn Project Concepts and Tools**					
	OpenCV (C)	Everyone	9/21/21	10/5/21	14	100%
	Simulink	Everyone	9/28/21	10/12/21	14	100%
	Gazebo OR Jmavsim	Everyone	10/5/21	10/19/21	14	100%
	PX4	Everyone	9/21/21	10/5/21	14	100%
	QGroundControl	Everyone	10/12/21	10/19/21	7	100%
3	**Project Design, Analysis and Validation**					
	Make SRS	Everyone	9/21/21		0	100%
	OpenCV Object Recognition program	Terry	10/20/21		0	40%
	Path finding implementation	Nathanael	10/20/21		0	20%
	Design Flight Director	Duy, Kevin	10/13/21		0	40%
	Design hardware components	Duy, Kevin	10/18/21		0	100%
	Test and verify in simulator				0	20%
4	**Project Building and Prototyping**					
	Build Drone	Everyone			0	100%
	Implement flight director onto hardware				0	0%
	Implement search program to hardware				0	0%
	Test and Debug Hardware implementation				0	0%
5	**Hard Deadlines**					
	Design Review Presentations				0	0%
	Proposal Due			11/7/21	0	0%
	Final Presentations			11/29/21	0	0%

Figure 22: Gantt chart of Fall 2021

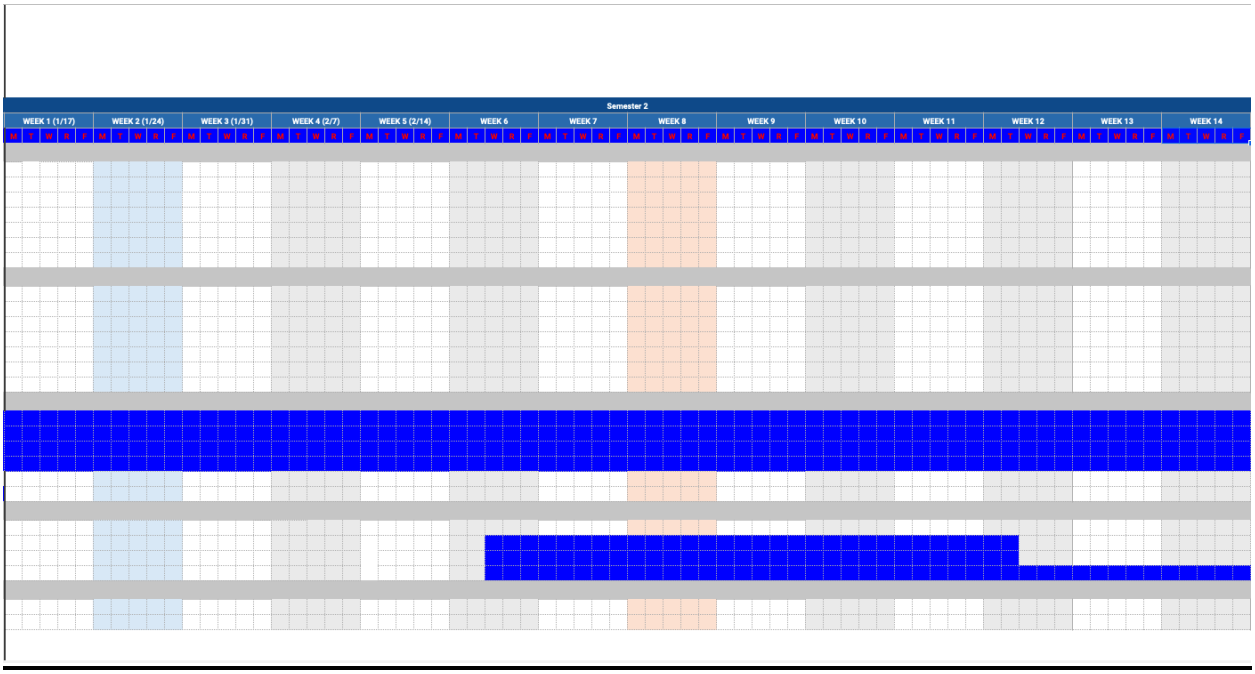


Figure 23: Gantt chart of Spring 2022

Summary

In this project, we utilized the flight control software PX4 to run a search-and-rescue mission autonomously. These systems were built using a combination of a flight directory in Simulink that was uploaded right to the PX4 flight controller, as well as an offboard computer to computer waypoints using a thermal camera, to be sent to the flight controller using MAVLink. The waypoints were coupled by using OpenCV to locate our target in the image with the instantaneous GPS location. We designed a flight mission that transitions from state to state synchronously between the offboard computer and flight director. One of the major stages was the controlled approach, which uses OpenCV for object validation as well as a servo with a retractable magnet to retrieve the target.

References

1. UConn, pp. 1–11, *Use of Drones/Unmanned Aircraft Systems*.
2. “PX4 Autopilot User Guide.” *PX4 User Guide*, PIx Hawk, https://docs.px4.io/master/en/.