WPI

BranchBot: Autonomous Quadcopter for Branch Attachment

A Major Qualifying Project Report submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the Degree of Bachelor of Science

Submitted By:
Zane Altheimer (RBE)
Keelan Boyle (RBE/ESS)
Cooper Dean (CS/MA)
Andrew Kerekon (CS)

Project Advisors:
Professor Andre Rosendo (RBE)
Professor Dmitry Korkin (CS)
Professor Oren Mangoubi (MA)
Professor Robert Krueger (ESS)

I. Abstract

According to the United States Environmental Protection Agency, pollinators are responsible for assisting 90% of all flowering plant reproduction. However, the effects of climate change and human development have caused their habitats to decline in recent decades. Technology such as robotic bees could be used to supplement pollination efforts, but they require a mobile pollination base station. To address this, we designed a quadcopter with an underactuated grasping mechanism, custom flight software, and a depth-based vision model to identify and perch on tree branches. Our approach allows for autonomous mobile pollination in hard-to-reach locations, ensuring that natural pollinator populations have the ability to rebound.

II. Acknowledgements

We would like to thank our advisors Andre Rosendo, Dmitry Korkin, Oren Mangoubi, and Robert Krueger for their continuous support and guidance as we progressed through our project. We would also like to thank Nitin Sanket and the PeAR Lab for allowing us to use their drone flight cage for testing as well as Yijia Wu for their assistance in printing dual-material test parts. Cameron Best, Spencer Granlund, John Lemieux, and Brendan Leu also provided invaluable assistance in the completion of this project through providing advice on soldering, helping us achieve stable flight, and 3D printing components of our drone. Finally, we would like to acknowledge the Underwater Filmography Using Robots MQP team for sharing their LiPo battery charger to recharge our drone.

III. Authorship

Section	Author
Abstract	All
Acknowledgements	All
Authorship	Andrew
1 Introduction	All
2.1 Background – Unmanned Aerial Vehicles (UAV) Selection	Keelan
2.2 Background – Grasping Mechanism	Zane
2.3 Background – Object Detection	Cooper
2.4 Background – In-Flight Communications	Andrew
3.1 Design – Drone	Keelan
3.2 Design – Grasping Mechanism	Zane
3.3 Design – Vision Model	Cooper
3.4 Design – In-Flight Communications	Andrew
4.1 Results – Drone Flight	Keelan
4.2 Results – Grasping Mechanism	Zane
4.3 Results – Vision Model	Cooper
4.4 Results – Communication and Autonomous Simulation	Andrew
5 Conclusions and Future Work	All
References	All
Appendix A: A Comparison Between Premade and Custom Drones for Environmental Robotics	Keelan

Table of Contents

1. Introduction

According to the USDA, one out of every four bites of food we eat is the responsibility of honeybees. However, honeybee populations are vulnerable to diseases such as Colony Collapse Disorder (CCD), which causes worker bees to abandon the hive. CCD is responsible for 30% of all honeybee colonies lost between January and April of 2023 (USDA, NASS, ASB, 2023). We currently do not know the cause of this disease.

The ecosystems that have lost their most productive pollinators now face an uncertain future. Pollinators are responsible for 80% of all flowering plant reproduction, and without them these environments will collapse (Randall, 2022). One possible way to address this situation is to introduce artificial pollination methods to these ecosystems. This allows for the plant life to continue reproducing while we can investigate more potential causes and treatments for CCD.

To successfully implement these artificial pollination methods, a base station is required to recharge or reorient from. In the outside world there are very few naturally flat surfaces for base stations to be attached or built. Perching or being able to sit on other natural surfaces is beneficial as they require no modification of the environment to complete their objective. Ideally, these systems require as little human interaction as possible. To this end, an autonomous system capable of performing these tasks is preferred.

The goal of this Major Qualifying Project (MQP) was to develop an autonomous quadcopter that could identify and attach itself to a tree branch. In order to complete this goal, the project was split into four objectives. First, we assembled a quadcopter capable of flight. Second, we created a grasping mechanism that was capable of attaching the drone to a tree branch. Third, we created a computer vision model that could identify a branch and calculate its position relative to the drone. Fourth, we proposed a pipeline to support autonomous missions. By completing these objectives and combining the results into a drone, we created a proof-of-concept for a mobile base station that can be further modified to suit different artificial pollination needs.

2. Background

2.1 Unmanned Aerial Vehicles (UAV) Selection

There are many types of Unmanned Aerial Vehicles (UAV) such as fixed wings, helicopters, and gliders each with unique benefits and drawbacks. For this project a quadcopter was chosen as they are capable of vertical take off, hovering mid air, and have a relatively low cost to manufacture. Some drawbacks however include that they have a relatively low payload and battery life (Mohsan et al., 2023).

2.1.1 Frame Configurations

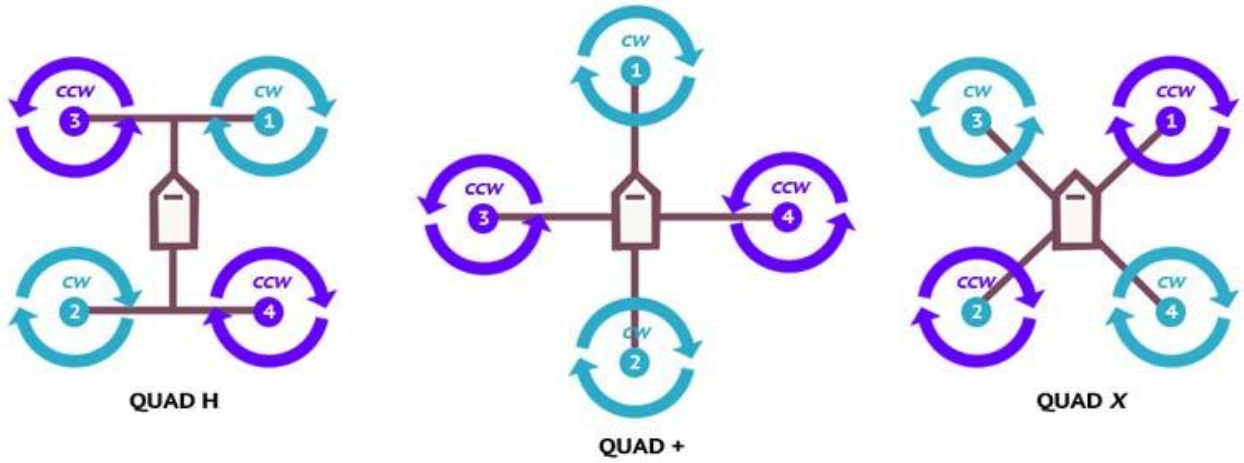


Figure 1: The base layout of the main quadcopter configurations.

Quadcopters have three main frame configurations: H-frame, X-frame, and plus-frame (see Figure 1). Each of these frames have four motors however they are positioned in slightly different ways. For the H-frame each of the arms are connected to the frame perpendicularly as seen in Figure 2. Mounting to the end of the frame allows for an increased payload and mounting capacity at the cost of increased weight and an asymmetric roll and pitch axis (Oscar Liang, 2023a; Tattu Gens, 2017). For the plus-frame which has the arms placed so that one of the motors is directly in the forward position. This is so that the side motor is spinning in turbulence free air. However, a forward facing camera gets blocked by the front motor. The X-frame is one of the most popular quadcopter frames as the roll and pitch axis are symmetric and is highly weight efficient (OscarLiang, 2023a; Tattu Gens, 2017). The X-frame motor layout is similar to the plus frame but each arm is rotated to a 45° angle which allows for a forward facing camera. The downside however is that there is limited space for the battery, flight controller, and other electronics (Tattu Gens, 2017; Unmanned Tech, 2018). These three frame types have been expanded upon to create quadcopters with all types of attributes. For example, a HX-frame combines the long chassis of the H frame with the motor layout of the X-frame as seen in Figure 2. This type of quadcopter is still a H-frame in terms of payload capacity however there is a more

symmetrical control scheme in the roll and pitch axis.

Figure 2: A comparison of H-frames vs HX-frames (Unmanned Tech, 2018)

2.1.2 Quadcopter Performance Metrics

One of the major performance metrics when creating a quadcopter is the thrust to weight ratio (Drone Edger, n.d; Walter, 2024). This ratio is calculated by taking the expected output of each motor/propeller combination and adding them together to get the total thrust. Then this thrust is divided by the total mass of the quadcopter. This ratio is then used to characterize the sensitivity of the quadcopter. Generally the lower the thrust to weight ratio the less control the operator will have with the benefit of smooth flight. In general quadcopters that are looking for a smooth stable flight are aiming for around 2-4:1 thrust to weight ratio where stunt and racing drones are aiming for closer to 10:1 (Drone Edger, n.d.; Walter, 2024).

2.1.3 Motor and Propeller Configurations

Quadcopter motors have two main attributes: the motor velocity constant (KV) and shaft diameter. These two attributes allow for thrust calculations and integration with the propeller. The KV is calculated by measuring the velocity of the motor when there is no load when one volt is applied. This rating then correlated with the pitch and diameter of a propeller to calculate the thrust (Vector Technics, 2023). The shaft size for both the propeller and the motor is important as they need to be able to fit together.

Propellers have three important metrics: diameter, pitch, and blade count. These metrics determine the thrust that the propellers are able to provide per revolution (Oscar Liang, 2023b). The diameter of the propeller is generally bounded by the distance between motors and the type of propellers that need to be used. Overall the larger the propeller the more thrust that is output at the expense of a higher torque needed from the motor. The pitch or the angle of attack is the ratio

of vertical distance to horizontal distance. This generally means as the pitch increases more thrust per rotation occurs giving the motor more control at the expense of requiring more energy to rotate. Similarly, the number of blades on a propeller increases the amount of thrust per rotation at the cost of an increased torque requirement for the motor. Overall, it is important to maximize the thrust that is achieved per rotation but it needs to be balanced with the required thrust-to-weight ratio, and energy requirements (OscarLiang, 2023b).

2.2 Grasping Mechanism

For the drone to successfully perch on branches, a lightweight, yet easy to actuate mechanism must be implemented. On a drone there are essentially two possibilities for the location of a manipulator: on top, or hanging underneath the drone. Grasping mechanisms that are located on the top of the robot allow for the robot to hang rather than perch atop a branch, similar to the skew gripper proposed by Molina and Hirai (2017b). Having the grasping mechanism on top of the robot introduces concerns about hitting the target branch with the propellers, a disastrous, and potentially fatal event for the drone. Depending on how large the mechanism is, it also introduces concerns of the center of gravity of the entire drone, and potential disturbances to the flight characteristics of the drone. If the mechanism is a manipulator with multiple joints, care must also be taken to prevent the manipulator from interfering with the propellers. The benefit presented by a hanging drone is that it has little chance of becoming an inverted pendulum problem.

A secondary approach is to put the grasping mechanism underneath the drone body, leaving it safely out of range of the propellers. This position allows for easier takeoff procedures, as hanging underneath the branch requires the drone to reach flight capability while falling, while takeoff from on top of the branch resembles the typical procedure of taking off from the ground.

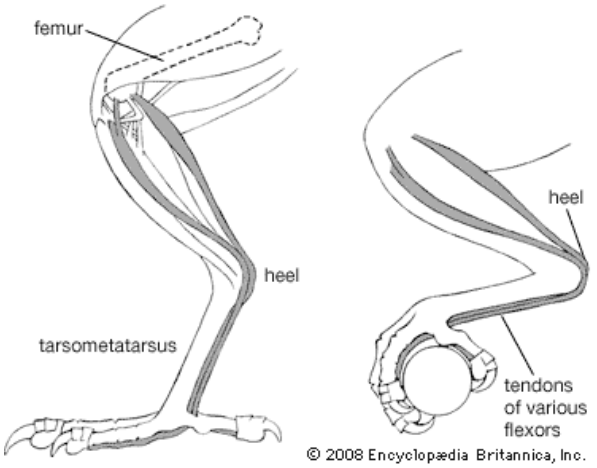


Figure 3: The tendons and other anatomical features of a pigeon that allow it to perch on branches even while asleep (Encyclopædia Britannica, n.d.)

Many of the existing perching mechanisms are inspired by the biomechanics of perching songbirds (Chi et al., 2014; Doyle et al., 2013; Nadan et al., 2019; Roderick et al., 2021). These birds can sleep perched on a branch by entirely passive means through the tendons in their legs, see Figure 3. These designs make use of tendon-actuation and compliant materials, to keep the mechanism lightweight and adaptable to irregular shapes, such as tree branches. These bio-inspired designs rely on the weight of the drone itself to maintain the drone's grasp on the branch, eliminating the need for servos or other heavy actuators. These passive designs are ideal for drones that are intended to spend a long time on location, performing surveillance, or waiting for other tasks to be completed as they spend little to no power to maintain their position, whether on a manmade or natural surface or perch.

2.3 Object Detection

Common computer vision models make use of object detection in some form. We decided that the methods explained in this section would provide the most benefit to us as we explored the detection of tree branches.

2.3.1 YOLO: The Industry Standard

You Only Look Once, or YOLO, is a relatively new approach to object detection (Redmon et al., 2016). By using regression in a single neural network, the model is able to be optimized for performance. This allows YOLO to process images at an incredibly fast rate. The model itself does have a few limitations that affect our project. Since YOLO draws a bounding box around objects, it is very inefficient when detecting elongated objects that do not appear exactly perpendicular to an image, such as tree branches. The loss function of the model would treat this inefficiency as a large error, causing problems during training.

2.3.2 Line-Based Deep Learning Method for Tree Branch Detection from Digital Images

Silva et al. (2022) introduces a unique method of identifying tree branches in an image. The proposed model attempts to estimate the orientation and grasping position of a tree branch through the use of a Convolutional Neural Network (CNN). First, the model predicts the straight line that best represents the location branch itself. Second, the model estimates the direction and position of the line using a Hough transform. Finally, the model finds a specific "grip point" defined as the point on the line with the highest probability of being on the branch. With application to our project, this model also contains a few limitations. First, the model can only be applied with the assumption that a tree branch is already in an image. Second, the model has significant computational cost associated with its use.

This paper also introduces a complete dataset of 1868 branch images. These images were collected by cropping images of trees found through a search engine. The resulting images were rotated by three angles to create the final dataset. Each image also corresponds to a label in a csv file that was also given with the dataset.

2.3.3 Contour Plots

An alternative approach to object detection makes use of generating the contour plot of an image. With this approach, it is very simple to obtain the size and shape of objects in an image allowing the use of additional computation to retrieve image data without excessive processing time. To create a contour plot, the image must pass through some edge detection algorithm.

A Computational Approach to Edge Detection by John Canny (1986) describes an algorithm commonly known as “Canny Edge Detection”. This algorithm is commonly used in Object Detection models to draw an outline around all objects in an image. Canny explains that the optimal use of the detector is on Gaussian-smoothed images. This means that the ideal pipeline for generating a contour plot of an image will follow three steps:

1. Gaussian Blur
2. Canny Edge Detection
3. Contour Plot Generation

After generating a contour plot, the properties of any object can be calculated using image moments. In mathematics, the moments of a function provide measurements related to the shape of the function's graph. By setting the function’s shape to a contour the equation to find a specific moment in a 2D image follows:

$$M_{ij} = \iint_D x^i y^j dA$$

We can compute the 0-th order moment M_{00}, or the area, as well as both first order moments M_{10} and M_{01} discretely over a finite set of pixels I using the equations below.

$$M_{00} = \iint_D dA = \frac{1}{2} \sum_{i \in I} \begin{vmatrix} x_{i-1} & y_{i-1} \\ x_i & y_i \end{vmatrix}$$

$$M_{10} = \iint_D x dA = \frac{1}{3} \sum_{i \in I} \frac{1}{2}(x_{i-1} + x_i) \begin{vmatrix} x_{i-1} & y_{i-1} \\ x_i & y_i \end{vmatrix}$$

$$M_{01} = \iint_D y dA = \frac{1}{3} \sum_{i \in I} \frac{1}{2}(y_{i-1} + y_i) \begin{vmatrix} x_{i-1} & y_{i-1} \\ x_i & y_i \end{vmatrix}$$

To compute the centroid of a contour $\{\bar{x}, \bar{y}\}$, we take the corresponding first-order moment, then divide by the area so that $\bar{x} = M_{10}/M_{00}$ and $\bar{y} = M_{01}/M_{00}$. To convert this centroid to an exact pixel, we just need to round the result to the nearest integer.

2.4 In-Flight Communications

2.4.1 System Overview

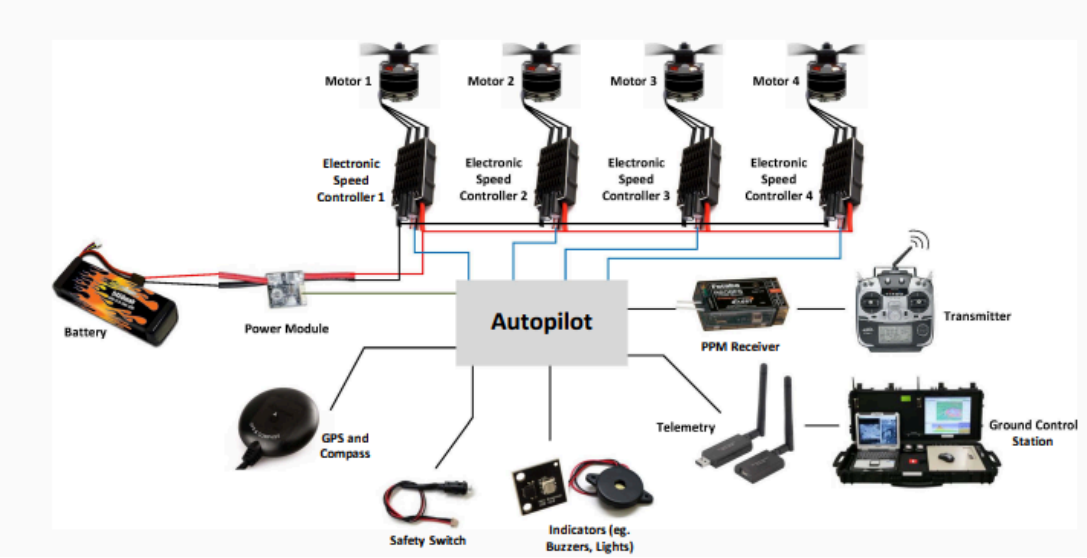


Figure 4: A standard autopilot configuration (Yang et al., 2016)

To support safe flight, it is imperative that all components onboard a drone are able to communicate without error or latency. At minimum, a drone must utilize a real-time operating system known as an autopilot that takes in sensor inputs such as gyroscope and compass, passes them through a filter to estimate proper signals for each motor, and outputs these signals continuously (Chao et al., 2010). According to Yang et al. (2016), autopilots are often installed on a dedicated board known as a flight controller that is fully integrated with the sensors necessary for flight. The combination of a dedicated board with an operating system that responds in real time allows for optimal control over a drone, since other computational processes do not interrupt continuous motor signals.

2.4.2 Common Autopilot Software

To allow for continuous communication to a drone's motors, autopilot software platforms – many of which are open-source – are installed on a drone's flight controller. Ebeid et al. (2018) compares nine such Open Source Software autopilot platforms that support autonomous flight and telemetry data as of February 2018. Yang et al. (2016) additionally references PX4 as a potential platform. Of these ten, four remain in common use as of 2024 – Ardupilot, Betaflight, INAV, and PX4:

- Ardupilot has the capacity to run under both 32-bit ARM machines and Linux, enabling testing on a variety of machines. Ardupilot also has ground control software written for Windows, Mac OS X, and Linux to enable mission planning, calibration, and vehicle setup.
- Betaflight focuses on high performance and cutting-edge features, with optimized code that officially supports 17 flight controller boards.
- INAV focuses on user comfort, with navigation features such as "follow me" commands, ground control software on Windows, Linux, iOS, and Android, and support for 25 flight controller boards.
- PX4, the main software supporting flight for Pixhawk brand flight controllers, is designed for "high-end research, amateur and industry needs" and can be customized to the needs of each user due to its open-source design.

As each autopilot platform contains similar features, of principal concern is that the chosen platform is compatible with the chosen flight controller board. Open source software is also preferable as it enables greater portability between flight controllers and enables researchers to validate and build upon existing work (Ebeid et al., 2018).

2.4.3 Sensor Requirements

Autopilots depend on input from sensors to guide the signals they send to a drone's motors. Common sensors – many of which are built into the flight controller itself – include IMUs to approximate linear acceleration and angular velocity, GPS to approximate position, and pressure sensors to approximate altitude (Ahmad et al., 2013; Yang et al., 2016). Additionally, Hell et al. (2017) cites radio controllers as necessary to ensure "safe and trouble-free radio communication between the drone and its pilot" that allow "the operator to see and control all of the drone's telemetry data and auxiliary sensor data in real time." Thus, ensuring an autopilot is provided with proper sensor data and the ability to communicate that data back to its pilot is paramount to permitting safe flight.

2.4.4 Permitting Autonomous Flight

In full autonomous flight, however, radio communication is not always possible – the drone must be able to process telemetry data and make flight decisions in the absence of user input. Decisions about flight directions require a drone to understand location – either relative to the objects around it or through knowledge of its global position. In this fashion, a combination of GPS navigation, devices known as rangefinders, and onboard cameras are used to provide sufficient input to either a flight controller's predefined mission or another onboard computer's directions.

2.4.4.1 Autonomous Flight Through GPS

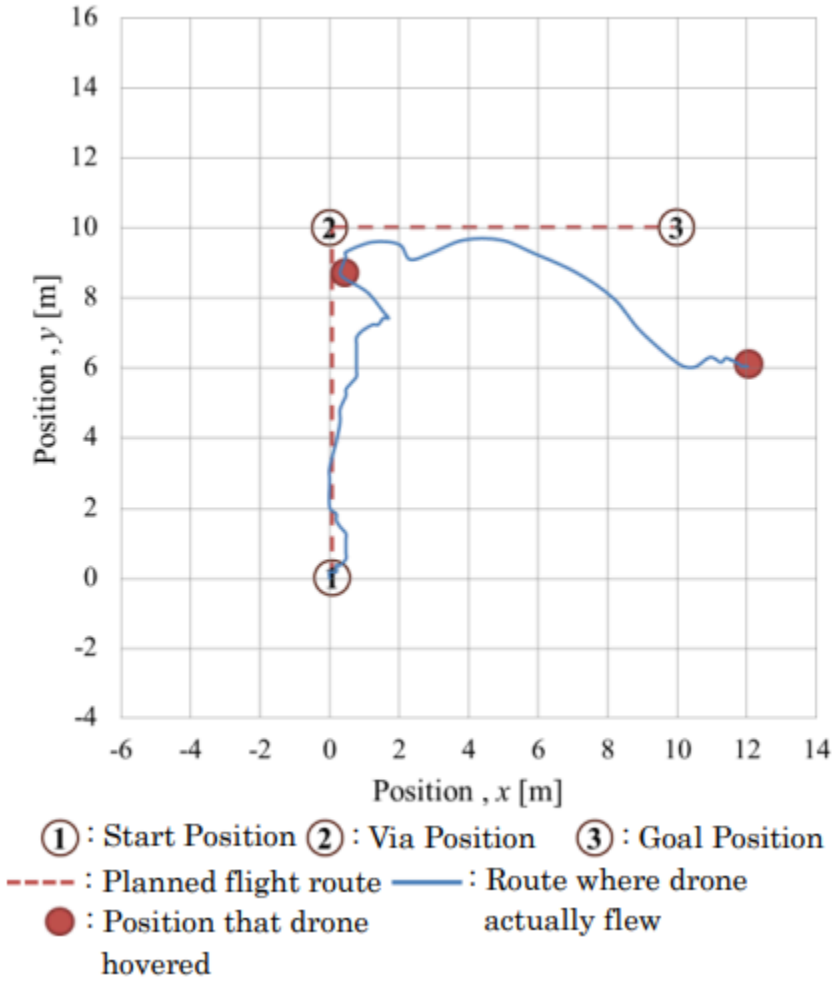


Figure 5: GPS-enabled autonomous flight is imprecise in reaching specific locations (Kan et al., 2018)

Kan et al. (2018) explored autonomous flight through the use of a GPS connected to a Raspberry Pi serving as the onboard computer of a drone. They observed that GPS guided flight followed directions as indicated, but did not display precision in reaching its goal positions.

Additionally, GPS autonomous flight is only feasible in outdoor conditions, as precise location can vary as much as 60 meters indoors due to “signal attenuation” and “multipath phenomena” where not many satellites are in view and those that are visible suffer from many echos before reaching a GPS receiver (Kjærgaard et al., 2010).

2.4.4.2 Autonomous Flight Through Rangefinder and LIDAR Measurement Unit

Sanjukumar et al. (2022) combined a HC-SR04 rangefinder with a LIDAR-Lite v3 optical measurement unit to determine objects surrounding a drone without an explicit need for global position. They found that under indoor conditions, this combination permitted successful

object avoidance at a range up to 5 meters, allowing for autonomous missions where a drone followed a preprogrammed autonomous path while avoiding any objects in the way. As these sensors have the ability to detect objects, they have the potential to be used alongside other sensors to identify types of objects at a given distance as well.

2.4.4.3 Autonomous Flight Through Depth Camera

Kawabata et al. (2018) studies the use of an Intel RealSense ZR300 depth camera onboard an autonomous drone to identify damaged infrastructure in Japan in the absence of GPS data. This camera, which can provide both RGB and depth output, generates a local map of its surroundings using the SLAM algorithm and sends this data to a Stick PC companion computer. From there, the companion computer sends position updates to the drone's flight controller via the MAVLink protocol. In this fashion, the drone can navigate autonomously by identifying objects in its way and commanding its motors to navigate around them.

2.4.5 Incorporating Vision

For a vision drone to detect objects, it must follow flight instructions from an attached flight controller, which in turn receives input from either manually operated radio controls or an onboard computer based on images from an attached camera.

Figure 6: Flight controller, Raspberry Pi, and camera for object detection in simulated flight (Szolga, 2021).

Szolga (2021) proposed such a drone using a Mamba F405 MK2 flight controller, a Raspberry Pi 4 onboard computer, and a Pi Camera Module V2. By mounting this drone to a shaking platform on a motorcycle, Szolga was able to faithfully replicate flight conditions and take photos with the Pi Camera at approximately 5.4 frames per second. Each photo was processed through a Python-based Single Shot Detection algorithm on the Raspberry Pi that overlaid objects in each image with one of 1000 predefined classes (Szolga, 2021). This project shows that a drone with an attached Raspberry Pi and camera is able to detect and classify objects under flight conditions.

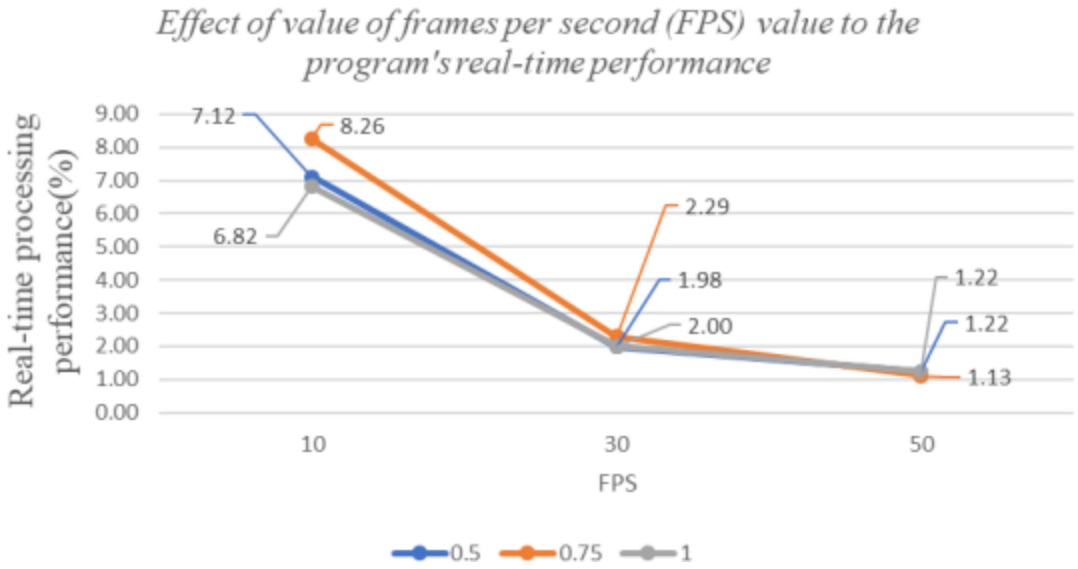


Figure 7: As camera frames per second increases, real-time processing performance decreases significantly (Khoi et al., 2021)

Object detection on Raspberry Pi drones comes with one principal challenge: ensuring sufficient processing capacity to classify images in real-time. Under changing conditions, real-time classification is crucial to accurate decision making. Khoi et al. (2021) defines real-time as "the ability to process all images that the camera obtains in a shorter or equal amount of time that the camera needs to capture the images." Therefore, each image from a camera capturing at 10 frames per second must be processed in 0.1 seconds or less in order to be considered real-time. Using a video feed from a drone equipped with a Raspberry Pi 3B+, Khoi et al. observed that approximately 14 seconds were required to process 1 second of video, and concluded that 0.71 frames per second would be the maximum permissible input to ensure real-time classification on a Raspberry Pi 3B+. To alleviate this bottleneck, this paper suggests the use of a graphics processor such as a NVIDIA Jetson Nano as well as improved image detection algorithms (Khoi et al., 2021). Although real-time image classification is possible on a Raspberry Pi, great care must be taken to ensure framerates are not higher than what can be processed by onboard computing equipment.

2.4.6 Flight Communication Protocols

To ensure all components can communicate, valid protocols must be chosen at every interaction within the drone. These protocols can be broken down into three main interactions – flight computer to flight controller, intra-flight-controller, and flight controller to motors.

2.4.6.1 Flight Computer to Flight Controller

A flight computer, such as a Raspberry Pi, must be able to not only take in signals from peripheral components such as GPS units, but also communicate with the flight controller onboard the drone (Kan et al., 2018). On a physical level, communication between these components often exists via wifi connectivity, a USB connection, or a serial port (Brand et al., 2018; Kan et al., 2018; Alkadhim, 2019). On a software level, the MAVLink protocol is the industry-standard communication method to send information back and forth between a ground control station such as a flight computer and the drone's flight controller (Atoev et al., 2017).

Since MAVLink messages are low-level and often difficult to interpret, many libraries exist to allow for more user-friendly interaction with MAVLink's drone movement capabilities. One such library is mavutil, which is built on top of the Pymavlink Python library and is optimized specifically to communicate with the Ardupilot autopilot (Gustafsson et al., 2023).

2.4.6.2 Intra-Flight-Controller

Within a flight controller, UART serial pads may be used to solder any peripheral components that are not already built-in. Additional components may be connected via IBUS, SBUS, and I2C connections. Depending on the choice of autopilot software, such connections may need to be configured before they are recognized by a flight controller (Pütsep et al., 2021).

2.4.6.3 Flight Controller to Motors

Once a flight controller has evaluated proper headings at a given moment based on input from a flight computer or other onboard sensors, it sends a PWM signal via an ESC (electronic speed controller) that converts this signal into a pulse each motor can understand (Hadi et al., 2014). In essence, this changes how long each motor should stay on or off to adjust the speed at which it spins. Changing the PWM value sent to each motor will therefore allow for a drone to move in varying directions.

2.4.7 Simulation and Evaluation Options

(a) AirSim (b) Gazebo

Figure 8: The AirSim and Gazebo flight simulation software (Ebeid et al., 2018).

Figure 9: The Mission Planner flight simulation software (Kumar et al., 2019).

Although modeling exact flight physics is not always possible, proper simulation before flight is vital to ensuring a quadcopter behaves in a relatively expected manner when given a set of commands. Ebeid et al. (2018) studies six possible simulation platforms, of which three are in common use as of 2024 – AirSim, Gazebo, and jMAVSim. Additionally, the Ardupilot autopilot team maintains their own SITL (software-in-the-loop) simulation tool to observe flight behavior within their Mission Planner tool:

- AirSim is a platform released by Microsoft in 2017. Based on Unreal Engine 4, AirSim contains support for MAVLink as well as integration with Ardupilot.
- Gazebo is an open-source robotics simulator that additionally has support for simulating quadcopter flight via MAVLink and Ardupilot.
- jMAVSim is a Java-based physics simulator developed by the PX4 autopilot team with support for MAVLink but no support for Ardupilot.
- Mission Planner SITL is built into Mission Planner, the industry-standard base station for drones equipped with the Ardupilot autopilot. Through this tool, users can observe a drone's altitude and headings both in actual flight and when sent simulated commands via the MAVLink protocol (Kumar et al., 2019).

3. Design

3.1 Drone Hardware

In order for the drone to successfully fly and complete its mission multiple goals need to be met. The first parameter is that the drone is able to fly with precise inputs for 5 minutes. For this to happen the drone must be constructed so that there is a 2:1 thrust-to-weight ratio so that our Raspberry Pi can easily control the drone. The next goal is that the center of mass of the drone is directly underneath the flight computer. The last goal is that there is a minimum flight time of 5 minutes in order for a significant amount of testing to happen in one session.

3.1.1 COTS vs Custom

One of the first design choices that we needed to make was whether we were going to buy a premade drone or make a custom drone. We ended up choosing a custom drone due to budgetary restraints and the necessity to have custom mounting points for our grasping mechanism. See Appendix A for more details on why we chose a custom drone.

3.1.2 Coordinate System

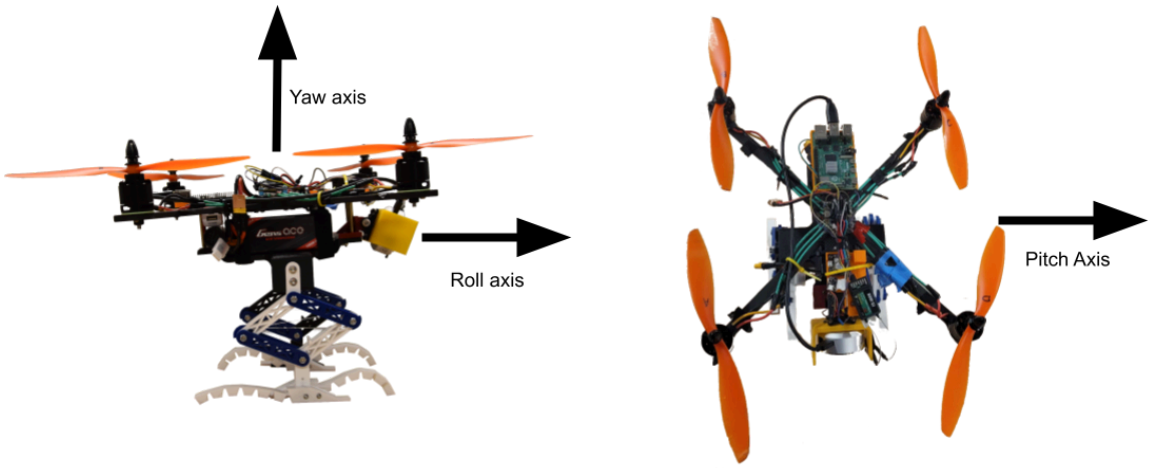


Figure 10: The roll, pitch, and yaw axis of the quadcopter.

In order to standardize the coordinate frame of the quadcopter it was decided the camera is the front of the drone and from its orientation positive is moving in the forward direction. This can be seen in Figure 10.

3.1.2 Camera Mount

We selected the Intel RealSense L515 Depth Camera as it is more capable for the mission (see section 3.3.1). However, as this change happened after the drone frame was bought the frame and motor selection were not able to be changed.

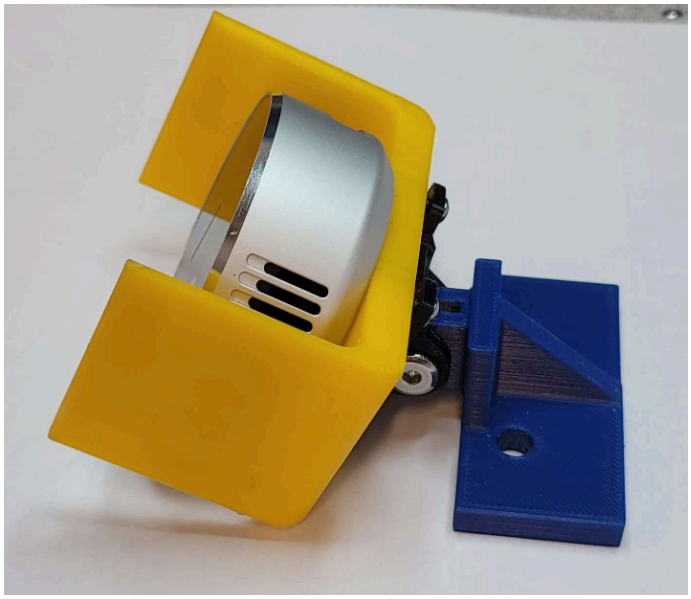


Figure 11: The adjustable camera mount.

In order to mount the Intel RealSense camera two main considerations needed to be addressed: the orientation and the safety of the camera. The orientation of the camera is important as the field of view (FOV) of the camera is limited to 70° horizontally and 55° vertically (Intel, n.d). This led to us rotating the camera 90° as we decided having a larger FOV in the vertical direction was beneficial, especially since the drone needed to land on a tree branch. Then to protect the camera from damage the mount extended past the sides of the camera (see Figure 11). Then to test the best possible angle the camera was mounted to an adjustable mounting point that allowed the camera to swing anywhere from +20º to -20º off the horizontal. This allows us to test the optimal angle for our quadcopter without having to remake the mount every time.

3.1.3 Quadcopter Frame

Figure 12: A top down view of the Nazgul XL10 V6 10-inch Frame Kit found on IFlight (IFight, n.d).

For the frame, there were a few major constraints, the size of the camera and the center of mass for the quadcopter, the material, and availability. In order for the quadcopter to be able to detect tree branches the camera needed to be in front of the quadcopter and have an unobstructed view of the horizon. This design constraint along with the more compact design led us to choose the X-frame over the H-frame and square-frame designs. The next step was to determine the size of both the propellers and the frame itself. The quadcopter needed to support a Raspberry Pi 4B, flight controller, camera, battery, radio receiver, and grasping mechanism. Due to the camera needing at least 160 mm on the edge of the quadcopter and 10 in propellers a 30 cm frame was chosen as the camera was able to comfortably fit inside the motors. The frame we bought to accommodate these is a Nazgul XL10 V6 10-inch Frame Kit. This frame is made out of carbon fiber as it is incredibly strong and light.

3.1.4 Motors and Propeller Selection

Table 1: Mass Breakdown of the Quadcopter

Name	Number of Object	Mass (g)	Total Mass (g)
Frame	1	219.9	219.9
Motor	4	141.77	567.08
Raspberry Pi 4B	1	49.6	49.6
MAMBA Flight Controller	1	29.4	29.4
Intel RealSense L515 Camera	1	94.6	94.6
Camera Mount	1	31.55	31.55
RC Controller	1	20	20
Barometer	1	5	5
Raspberry Pi Mount	1	10.78	10.78
Sensor Mount	1	12.31	12.31
Battery	1	585.2	585.2
Bottom Plate	1	219.51	219.51
Grasping Mechanism	2	112	224
Payload	1	500	500
Total Mass			2568.93

Once the frame and electronics were chosen an initial mass could be calculated. This involved massing every item as we received or created them. The final mass estimate ended up being roughly 2.5kg and the breakdown can be seen in Table 1. Notably, this mass estimate does include a 0.5kg payload that can be increased or decreased based on the needs of the mission. In other words, this estimate gives the minimum thrust requirements for the quadcopter. The total thrust can be calculated by using a thrust-to-weight ratio of 2:1, giving a total thrust required of 4.6kg. This is then divided among the four motors giving a thrust requirement of 1.15kg per motor propeller combination. Due to the large volume of combinations we decided to use a 10in propeller primarily as it is what our frame was rated for. We then looked for motors when combined with a 10in propeller would produce the necessary thrust at ~25% of their max so there would be no risk of burning out the motor. This led us to choose the SunnySky x2814 1000kv motor that produces a max thrust of 2230g with a 10in, 5” pitch propeller (SunnySkyUSA, SunnySky X2814 Brushless Motors). This leads to a max thrust to weight ratio of 5:1 with no payload and max throttle. This ratio can be decreased to 4:1 with the 0.5kg of

payload specified. This payload can be increased to 1.5kg to achieve a thrust to weight ratio of ~2:1. This increase of payload would come at the cost of the flight time of the drone (calcauted in section 3.1.7).

3.1.5 Bottom Plate

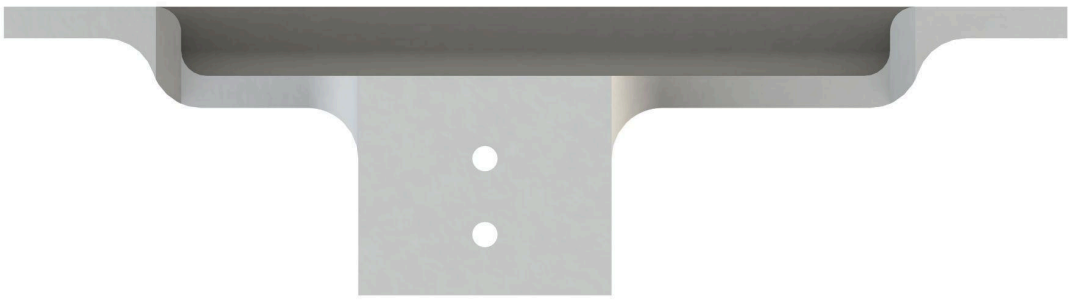


Figure 13: A side view of the final baseplate.

Throughout the duration of the project the baseplate went through three major revisions. The baseplate is used to hold the battery and mount the gripping mechanism to the quadcopter. To achieve this 51x3mm standoffs were screwed into heat set inserts embedded in the bottom plate. The gripping mechanism was then attached to the vertical holes below as seen in Figure 13. It was quickly realized that the two grasping mechanism modules were mounted too close to each other and the drone would easily tip over onto its side. This was removed by increasing the distance from 50 to 150mm as seen in the transition between Figure 14 (a) to (b). Another design change made during the redesign was to move the mounting standoffs closer to the front of the drone. This was due to the center of mass being slightly towards the front of the drone. These two changes allowed for the quadcopter to both stand on flat ground and a tree branch. The next challenge the baseplate faced was that when the quadcopter quickly impacted the ground the baseplate split right up the middle leaving the quadcopter unflyable. To address this the rib pattern seen in Figure 13(b) was changed to a square box as seen in Figure 14(c). This adds a significant amount of reinforcement while not impacting where the gripping mechanism is attached.

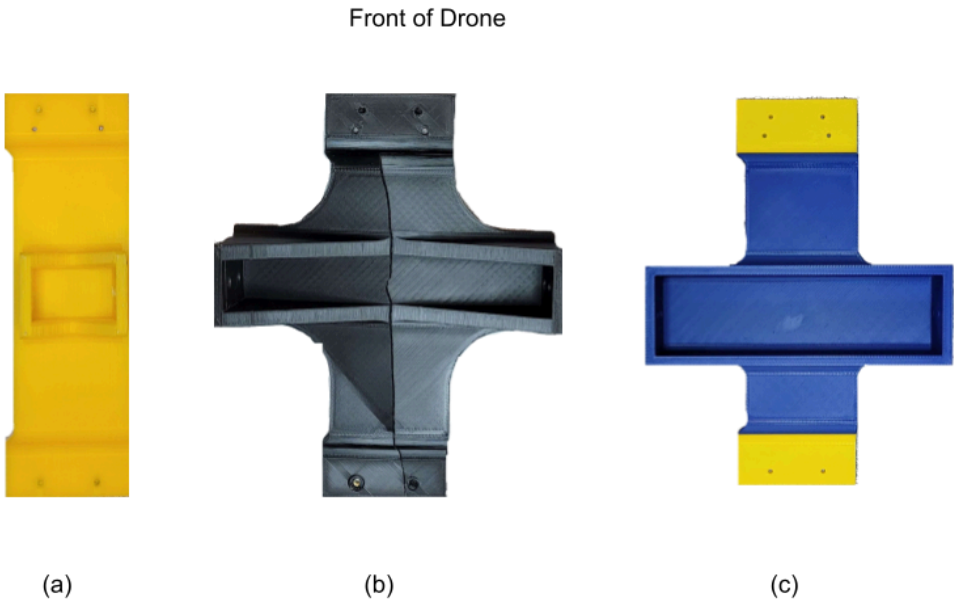


Figure 14: The versions of the base plate over the course of the project. (a) is the first iteration, (b) is the second iteration that has a crack down the middle, (c) is the final iteration.

3.1.6 Sensors and Communication

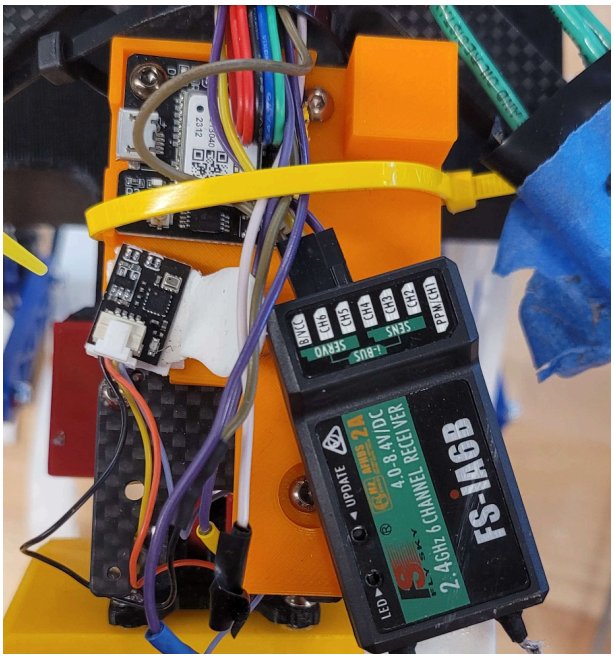


Figure 15: The non-integrated sensors on the quadcopter. In the top left there is a GT-U7 GPS module, middle left is a Flywoo BQNANO V1 barometer/compass, and on the bottom right is a FS-iA6B RC receiver.

While preparing for the quadcopters first flight we noticed the flight controller required more sensors than initially planned for. Primarily the flight controller was expecting to have a

barometer and a remote control receiver, both of which were not included on the board. To accommodate these new electronics we had to remove the secondary Raspberry Pi battery and add a sensor mount (see Figure 15). This allowed for the new barometer to have access to the open air and the RC receiver to be mounted in optimal conditions. This board also has space for a GT-U7 GPS module that was not used due to testing limitations.

3.1.7 Power Requirements

Table 2: Power Requirement Breakdown

Name	Voltage Required (V)	Amperage (A)
PS4 Camera	3.5	0.5
Raspberry Pi 4B	5	3
Motor	14.8	80
Flight Controller	5	3
Total A		86.5
Total mA		86500

According to the last section, the motors require ~14V input voltage. The standard way to produce this is to use a 4 cell Lipo battery which has a voltage of 14.8V. Then to achieve a flight time of 5 minutes we created a table with all the expected current draw in table 2. Using the expected current draw (83500 mA) we then use the equation capacity = milliamps x hours. Using this we found the ideal capacity of 7000mAh. The best battery in terms of weight and capacity ended up being a four-cell 7200mAh LiPo battery due to its high energy concentration.

3.1.8 Power Distribution

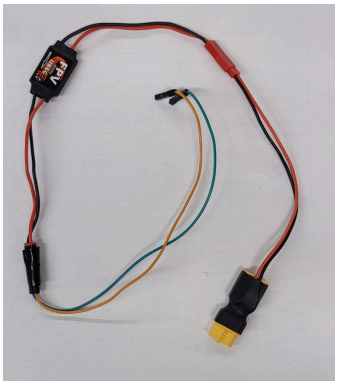


Figure 16: The Raspberry Pi 4B power cord. This contains a 30V to 5V converter connected to a xt60 to JST connector.

Our flight computer, a Raspberry Pi 4B, requires 5V and a continuous 3A of power in order to run at full capacity. The main battery voltage is 14.8V as per the motor and flight controller specifications. Initially, we were planning on adding a separate power supply as it would extend the flight time and not require any modifications of premade parts. However, we found that we needed space for other sensors such as a barometer and a RC receiver. This led us to replace the separate battery with a 30V step-down converter. This then posed the problem of how it would be connected to the main battery as it has a XT60 connector. To do this adaptation we bought a JST to XT60 adapter that would give power to the Pi in parallel. This connection is shown in Figure 16. This change allowed for a decrease in cost, an increase in space, and a decrease in complexity.

3.2 Grasping Mechanism

To successfully perch on a tree branch, the drone's grasping mechanism needs to be able to successfully interact with irregular surfaces while maintaining its enclosed grasp. To this end, both bio-inspired and inorganic designs were considered from the Walker et al. work focused on cephalopod-inspired arms, to the Roderick et al. work inspired by peregrine falcons, to inorganic designs such as Molina and Hirai's skew-gripper. Avian-inspired designs were of particular interest, as birds can easily achieve the intended behavior of the drone, ie., being able to perch in a branch with little to no actuation.

The major design constraints given by this problem is keeping the grasping mechanism lightweight, as it will be attached to a drone, and the design should require little to no power to maintain its perch to maximize battery life for other functions of the drone.

The initial design for the grasping mechanism is based on the Nadan et al. work, including a four-bar linkage that houses the tendon system, and dual-material talons. The four-bar linkage and the talon segments are made from 3D-printed PLA filament, and the compliant rail attached to the talon segments are printed from 3D-printed 95A TPU filament shown in Figure 17. In this initial prototype the compliant rails and the talon segments were joined together with a flexible cyanoacrylate adhesive. In this iteration the talons had little durability, being able to withstand approximately 10 actuations before the compliant rails split. The construction of the compliant talons was not appropriate as it wasn't durable enough for application, and the compliant rails were too soft to extend the talons when not under tension from the tendons.

Figure 17: Initial prototype grasping mechanism talons made up of separate pieces of PLA and TPU compliant rails glued together with flexible adhesive.

3.2.1 Talon Material Selection

From the initial design it became evident that the talons needed to be stiffer, and the four-bar mechanism needed to be reduced in size to be in proportion to the drone body. The following iterations experimented with materials.

The first combination of materials were dual-material 3D prints made up of PLA for the talon segments and 95A TPU for the compliant ribbon. This combination of materials did not adhere to each other, and the print ultimately failed because of the material incompatibility. The second iteration of these prints was made up of 75D TPU for the talon segments, and 95A for the compliant ribbon. This combination initially seemed promising, however we ran out of the 75D TPU, and it was too expensive to purchase for this project. The final material decision for the talons was to print them out of a single material with 95A TPU. It meant the talons were compliant enough to create a fully enclosed grasp, while being stiff enough to return to the initial position of the talon once the tendons are no longer in tension.

3.2.2 Talon Design

With this material selected, we began iterating on the geometry of the talons. The original design was very avian-inspired, particularly drawing from the Nadan et al. work. This talon featured a clawed tip, and a rear facing talon to provide more stability on perching surfaces as seen in Figure 18. The vertical difference between tendon pass-through points in the segments was intended to provide larger bending moments when the talon was made up of a compliant spine and less compliant segments, however this design required too much force to fully collapse the talons. In the next iterations the vertical difference was removed from the pass-through points for the tendons.

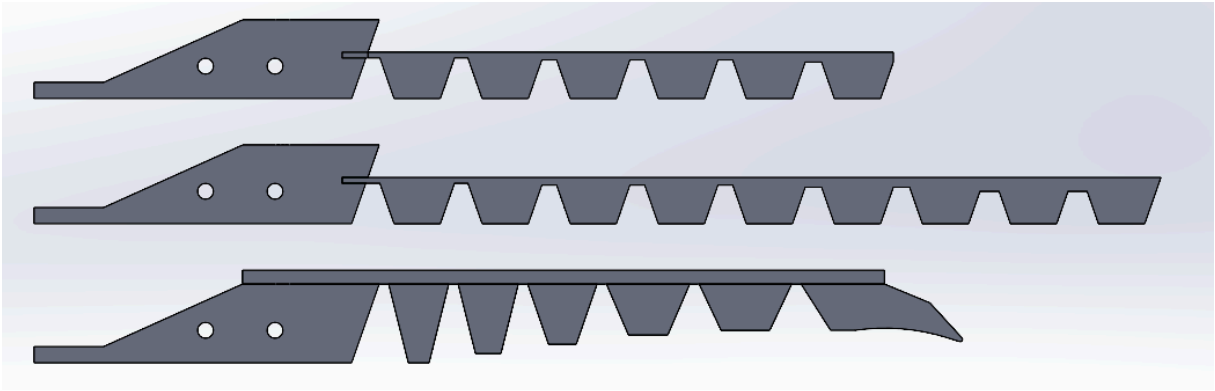


Figure 18: Several iterations of the compliant talon design. Bottom talon is inspired by the Nadan et al. work. Middle and top talons were inspired by the Doyle et al. work.

The next talon iteration was inspired more from Doyle et al.'s work and was less avian-inspired and more inorganic in appearance. The clawed tip has been removed, and the segments of the talon are the same height. The functional difference introduced in this design was the difference in thickness between the segments of the talon. This was introduced so the talon would begin to collapse at the segment closest to the ankle of the gripper, as seen in Figure 19, allowing for better grasp of irregularly shaped objects.

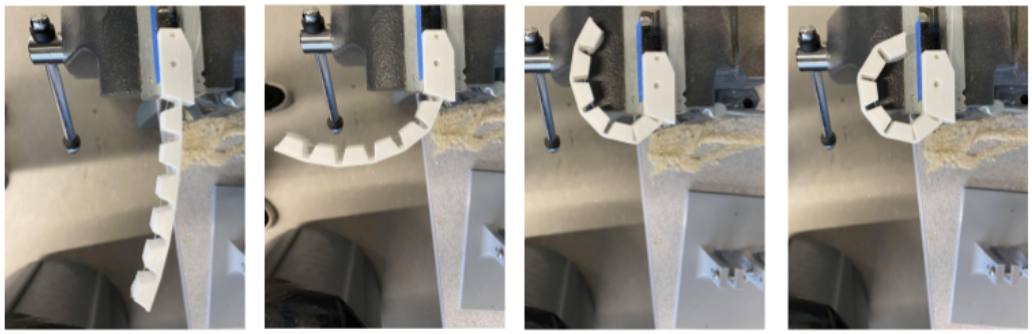


Figure 19: A four-step sequence of a singular talon collapsing.

To improve upon this design, in the next iteration the overall length of the talon was reduced in order to decrease the displacement required between the four-bar joints to fully collapse the talons while grasping. The other geometry of the talon remained the same, shown in the upper talons in Figure 18.

These newer talon iterations collapsed better than the more avian-inspired talons, however they provided less stability on flatter surfaces, leading to the reintroduction of the rear-facing talons from the original talon design inspired by Nadan et al.'s work. This increased the drone's ability to stand on uneven surfaces that did not require an enclosed grasp such as dirt or grassy surfaces.

3.2.3 Four-Bar Linkage Design

The four-bar linkage is responsible for housing the tendon system the grasping mechanism relies on to collapse the talons. It needs to be able to bear the weight of the drone as well as the tension force present from the tendons, while being lightweight. To this end, the four-bar mechanism was 3D-printed with PLA.

In the initial prototype of the mechanism, the overall size of the four-bar mechanism was much too large for the drone frame we purchased, and needed to be downsized in the subsequent iterations, but it proved the tendon system could collapse a compliant foot. This iteration was not intended to be put on the drone as it features no attachment points for the drone body.

The next iteration of the four-bar mechanism saw an overall decrease in size to make it appropriate to fit on the underside of the drone. From the center of the joint to the next on one link the four-bar mechanism is 75mm. In this iteration the tendons proved difficult to keep in place to ensure that they would not catch on the four-bar, the talons, or any other part of the drone body. This was solved by printing keepers (see Figure 20) that replaced the spacers in the four-bar linkage. These new keepers acted as spacers for the four-bar linkage, to allow the links to be spaced apart, and also included an enclosed space on the outer diameter of the spacer that could act as a keeper for the tendons. This prevented misalignment on the four-bar linkage as well as keeping the tendons from getting caught in other mechanisms.

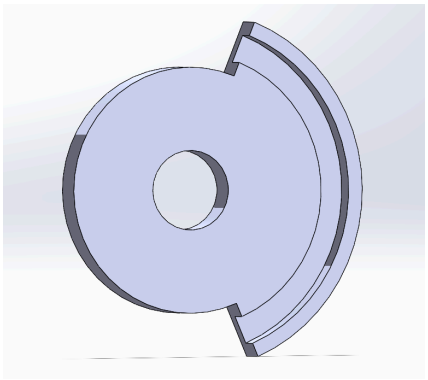


Figure 20: This is a section view of the keeper, shown at the midplane. The channel present on the outer diameter is for the tendon.

The next improvement upon the four-bar linkage was to introduce a hard stop to prevent overextension of the leg while in flight (see Figure 21). If the legs were allowed to extend beyond the vertical the four-bar linkage can get stuck in the extreme opposite of its collapsed position preventing any landing attempts. The hard stop introduced on the linkage prevents this behavior and ensures a safer landing sequence for the drone. It also keeps the legs in a more consistent position while the drone is in flight.

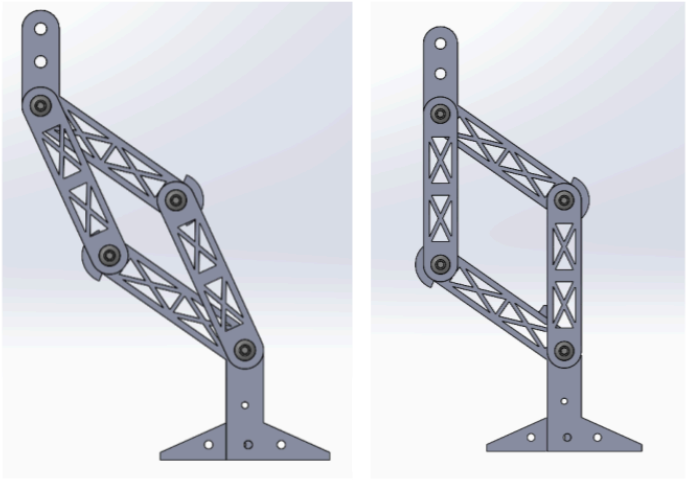


Figure 21: The left image shows the leg being overextended, in a position where the force applied by the drone's weight would not collapse the linkage. The right image shows the leg's fullest extension with the addition of the hard stops in the linkage.

3.2.4 Tendons

The tendons in the grasping mechanism are made up of monofilament fishing line. The ends are rigidly attached to the end of the talon on either side of the grasping mechanism. Assembly proved difficult as maintaining consistency between the two leg assemblies was difficult due to the tendons being tied to the end of the talons. This means some trial and error was performed to get the legs even with each other.

When the tendons were installed in the drone we pursued several options of where the tendon should be taut. We attempted stringing the tendons with the legs fully extended, however that proved to require too much force to collapse the talons. Stringing the tendons with the legs with the ankle and knee joints horizontally aligned, meant the four-bar linkage could not create enough displacement between the joints to collapse the talons fully. The final design was strung to be in-between these two positions to reduce the amount of force required to collapse the talons, while ensuring the displacement between the joints was large enough the talons could create a fully enclosed grasp.

3.2.5 Final Design

The final design of the grasping mechanism featured 95A TPU talons based off of the Doyle et al. work, with a PLA four-bar linkage. Fully extended it is 21cm tall, fully collapsed it is 14.4cm tall. The talons are 18.5cm long.. The tendons are made up of monofilament fishing line, strung as shown in Figure 22. The grasping mechanism is bolted onto the underside of the drone's body, underneath the battery.

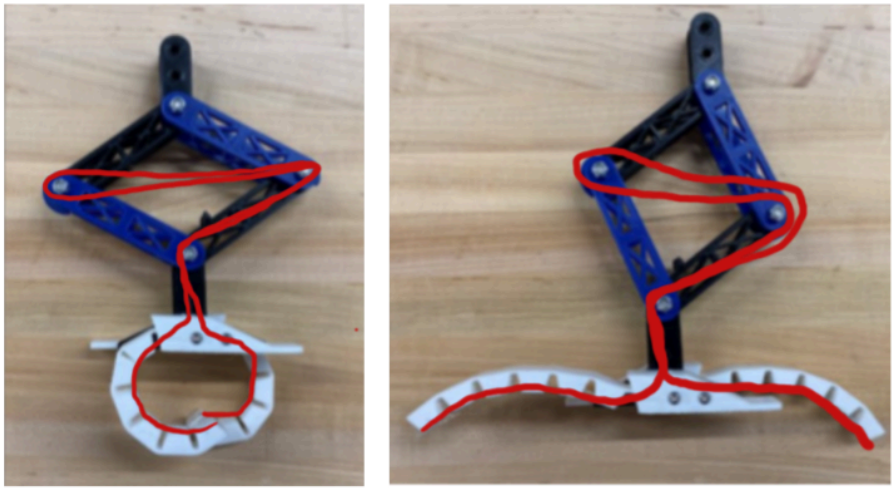


Figure 22: A complete leg subassembly, collapsed and open, the tendons are shown in red.

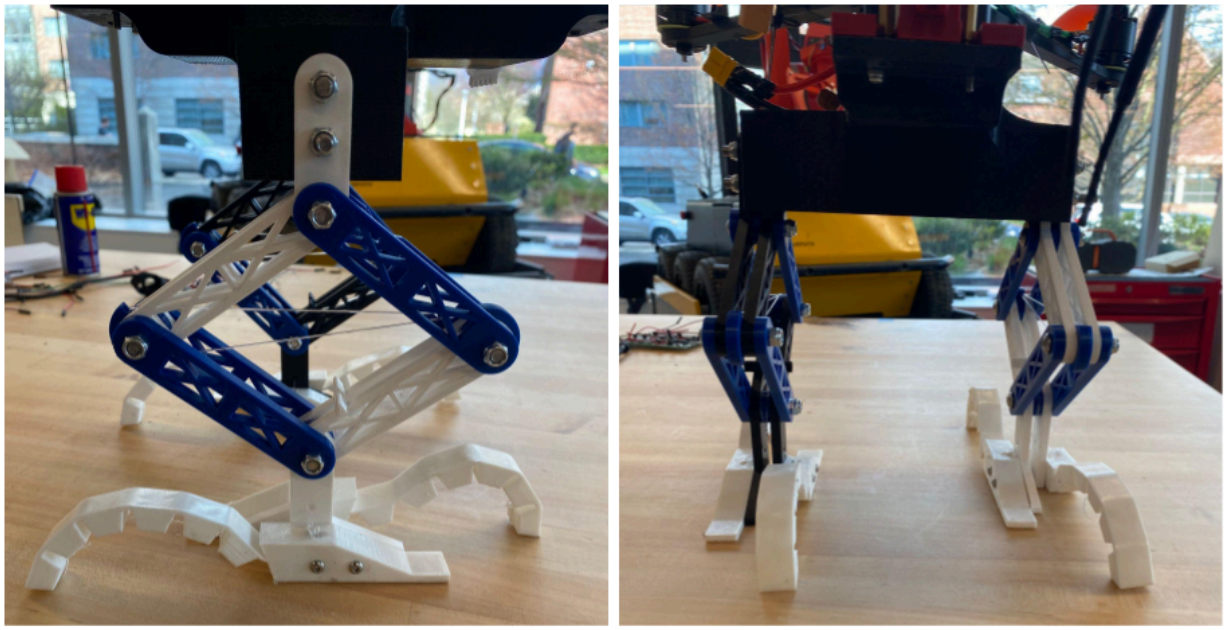


Figure 23: The final grasping mechanism, attached to the drone body. The left image shows a side view of the system, the right image shows the front view of the grasping mechanisms and lower drone body.

3.3 Vision Model

To ensure that the drone stays autonomous during flight, we needed to use a computer vision model. For a complete autonomous pipeline, the model must be able to search for a tree, find and evaluate the branches on that tree, choose a branch that is suitable for the drone to land on, and finally direct the drone to fly and land on the branch. Due to the nature of the project, we decided to reduce the scope of the vision model to two key components: branch identification and branch detection. We define branch identification as a process that determines if an image

contains a branch. Branch detection would then assume that an image already contains a branch, and compute its location relative to the drone.

3.3.1 Camera Choice

This type of computer vision model requires the use of a specific type of camera that is able to return both a color image and depth data. We considered the use of two different cameras due to our limited budget.

The PS4 Camera uses two lenses to deliver stereo depth for motion tracking. This output is received as one large image of three appended images as shown in Figure 24. The camera performs very poorly outdoors, as it was intended to be used in an indoor environment while playing games that required motion tracking or VR technology. We were able to address this by attaching sunglasses to the lenses of the camera, however this caused a distorted effect on the resulting image. The biggest issue that this camera caused was the lack of support for ARM architecture. Due to this problem, we decided to choose a different camera that was compatible with the Raspberry Pi.

Figure 24: PS4 camera output.

The Intel RealSense L515 Depth Camera featured LIDAR depth technology as well as BGR color output. The camera is able to return accurate depth measurements of objects at distances of up to 9 meters while indoors. This camera is also not rated for outdoor use, in some cases not returning any depth measurements of objects whose distances are less than 2 meters. This camera was compatible with the Raspberry Pi, however it required additional effort compared to most USB cameras.

We decided to use the RealSense Depth Camera due to its compatibility with the Raspberry Pi. Since the camera was unusable in an outdoor setting, our testing of any autonomous code took place indoors. While using this camera, we decided to place it on the drone rotated by 90°. This ensures that the camera is able to gather more data vertically than horizontally, allowing the drone to see the branch during the landing process.

3.3.2 Branch Identification Methods

In order to correctly identify that an image contains a branch, we decided to apply various machine learning models. Each model was trained using different datasets described in this section, with the results of each model described in section 4.3.

YOLO, as described in section 2.3.1, is the industry standard object detection method for use in robotics. In order to train a YOLO model, we created a custom dataset of 239 images gathered using the PS4 camera. Each image was annotated by hand, creating a polygon around each branch.

We also considered using standard binary classification models such as Logistic Regression, Naive Bayes, SVM, k-Nearest Neighbors, and Random Forest. Each model was trained using a mixed dataset that included images of various objects from CIFAR 100 as well as resized images of branches from the paper in the previous section. All images from CIFAR 100 were labeled with a 0, and all resized images of branches were labeled with a 1. The data was shuffled and trained using 10-fold Cross Validation. Performance of each model was determined by its average precision.

3.3.3 Branch Detection

The branch detection model assumes that the main object in an image is already a branch. We define the main object of an image to be the largest foreground object within a certain distance to the camera. The distance chosen for all tests was 3 meters, as further distances risk the loss of data due to pixelation.

The raw depth data given by the camera displays the distance in meters of each individual pixel of an image. The data is filtered such that any pixel with a distance larger than 3 meters is set to a value of 0 meters, removing any noise from distant objects. The raw depth data is then cropped from 1024x768 to 960x540 in order to fit the resolution of the color data, and converted to a grayscale image with closer objects appearing as lighter values. The model then blurs the resulting image using a 10x10 kernel and returns an image as seen in Figure 25.

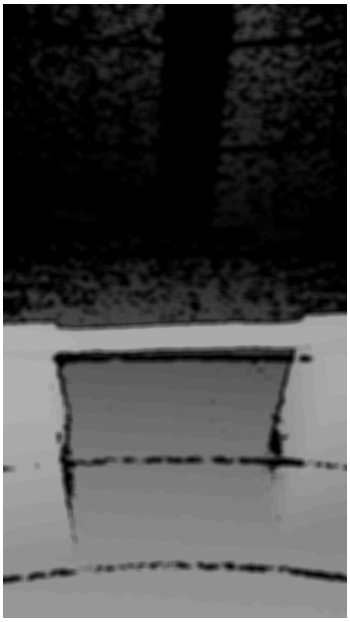


Figure 25: Raw depth data as grayscale.

After modification to the depth data, the model filters the color image by the resulting depth data. Any pixel in the grayscale image with a value of black will remain as black. Pixels that appear closer in the depth image will retain their color, while pixels that are further will appear darker.

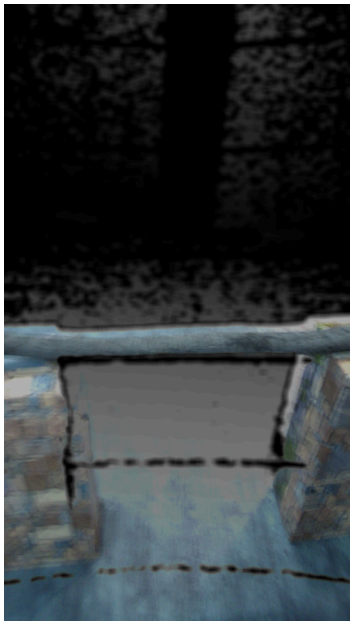


Figure 26: “Mixed” depth and color.

After generating the “mixed” image, Canny Edge Detection is applied and a contour plot is generated. The model then bounds each contour with a minimum area rotated rectangle. The

contour that corresponds to the rotated rectangle of largest area will be chosen as the main object in the image. This contour will always be a branch if the initial assumption is correct.

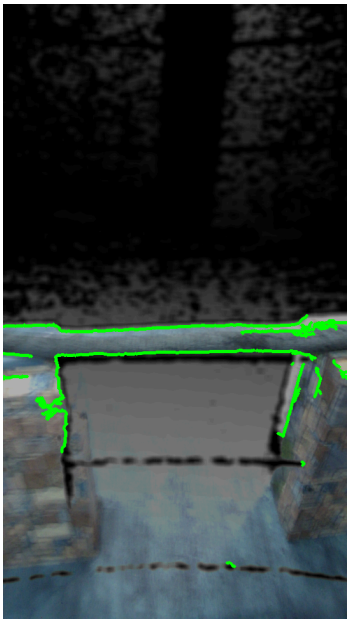


Figure 27: Contours drawn on image.

Once a contour of the branch is found, several operations can be used to extract the necessary data for autonomous decision making. By applying contour moments (see section 2.3.3), the centroid of the contour can be found giving an exact location of a singular pixel on the branch. The model uses the raw depth data to return the distance to the centroid as well as compute the necessary angle direction by using the FOV of the camera. The camera has an FOV of 70° x 55°, but due to cropping the estimated FOV is about 65.63° x 38.67°. Both angles are calculated using equations derived from the ratio of the FOV to the number of pixels in the image.

$$vAngle = (65.63° / 960px) * cY$$
$$hAngle = (38.67° / 540px) * cX - (38.67°/2)$$

Note that the centroid's x and y pixel values are defined as cX and cY respectively. The vertical angle from the horizontal can be computed directly assuming that the camera is facing down at an angle of 24.37° from the horizontal. The horizontal angle needs to be shifted, allowing any centroid in the left half of the image to return a negative angle while centroids in the right half of the image return positive angles. After the direction, vertical angle, and horizontal angle are computed, they are then returned to the main loop of the program for use in directing the drone.

3.4 In-Flight Communications

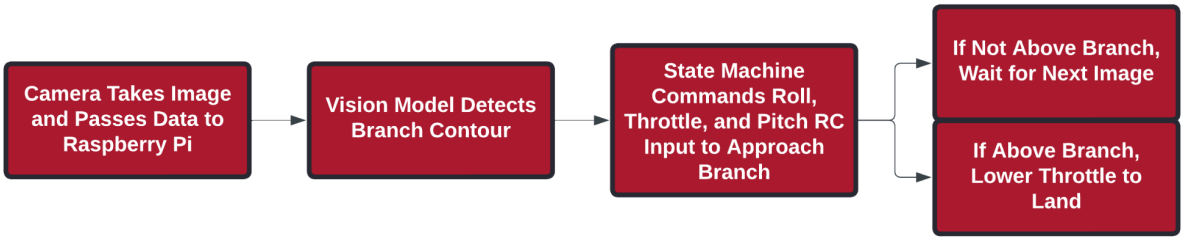


Figure 28: Overview of autonomous flight pipeline.

In order to facilitate internal communication, we designed a system such that all components talk to a central computer which decides the proper action to take at every moment. We chose a Raspberry Pi 4B to serve as the central computer, mounted along the back-center beam of the drone frame. To form an autonomous pipeline, every two seconds we take an image using an Intel RealSense L515 depth camera mounted to the front of the drone. These data are fed to the Raspberry Pi, which identifies a branch contour using an algorithm described in Section 3.3 and provides the proper distance and heading to reach that branch's centroid. The Raspberry Pi then determines the necessary roll, throttle, and pitch to reach this branch and sends a MAVLink packet to a Mamba F405 MK2 v2 flight controller, which signals the drone motors to follow the specified heading. If the drone is already above a branch, we lower the throttle to attempt landing; if it is not, we wait until the next image is delivered. We additionally simulate these conditions to ensure safe flight.

To determine whether successful in-flight communications were achieved, we establish three main goals:

1. All components communicate without protocol or version failures.
2. Autonomous pipeline successfully takes an image, receives it from the vision pipeline, and calculates proper angles and directions from that image.
3. Autonomous flight model provides safe inputs with proper force and direction to simulated and actual drone.

3.4.1 Raspberry Pi as a Flight Computer

The Raspberry Pi 4B serves as the central computer onboard the drone, processing incoming data from the Intel RealSense camera and sending outgoing data to the Mamba flight controller. The Raspberry Pi 4B was the most powerful Raspberry Pi board available at time of design, and has the necessary USB and GPIO connections to ensure stable power and communication. As Pi systems use ARM-style architecture, we selected Raspbian Buster to serve as the operating system.

3.4.1.1 Incoming and Outgoing Connections

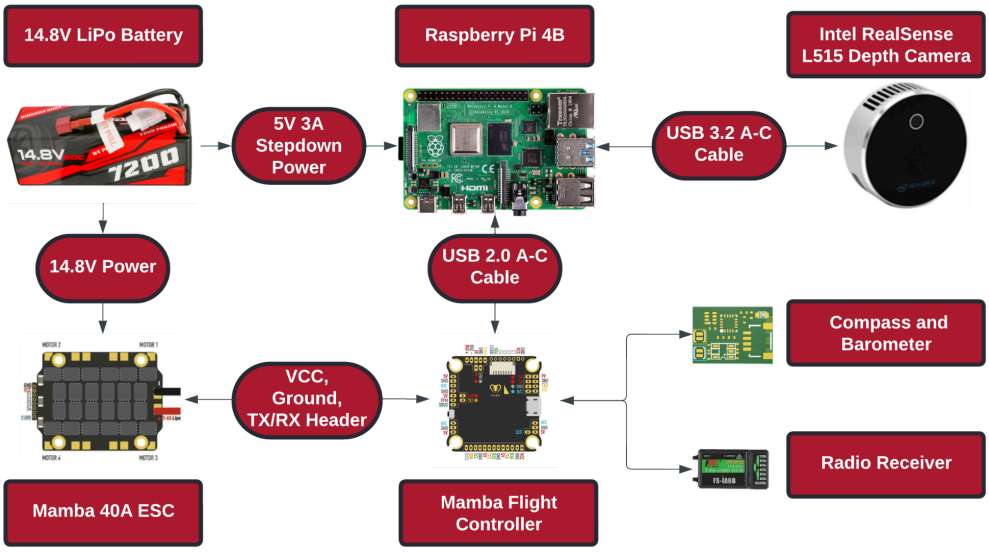


Figure 29: Overview of in-flight component interactions.

- 5V, 3A power is supplied from the drone battery via a 5V step-down to GPIO pins #2 (5V) and #6 (ground).
- A 32GB microSD card is mounted to allow persistent data.
- A USB 3.2 port connects to the Intel RealSense camera via a USB A-C connector.
- A USB 2.0 port connects to the Mamba flight controller via a USB A-C connector.
- The Pi's internal network card connects to WPI-Wireless wifi.
- In testing, two USB ports are used to connect to a keyboard and mouse.
- In testing, an expansion bus is added to permit HDMI connection to an external display.

3.4.1.2 Raspbian Buster Operating System

Raspbian Buster is the industry-standard OS for Raspberry Pi systems, but trial-and-error is necessary to ensure all components can properly communicate. Since the Raspberry Pi uses the ARM instruction set, we needed to compile many packages from source to ensure they would work with specific technologies and software dependencies. Additionally, not every package was version compatible, so it was necessary to find *old-enough* versions of packages that would still work with *new-enough* technologies.

The main limitation was the Intel RealSense camera, which relies on Intel's librealsense library and pyrealsense2 bindings. Intel's librealsense in turn relies on Google's protobuf serialization library and the OpenGL graphics library. Although Intel recommends Ubuntu Linux

for development, we found that version incompatibilities were too great to overcome – certain packages were not available on newer builds of Ubuntu, and certain packages were no longer available (or were never available) on older builds. Pivoting to Raspbian, Raspberry Pi's default OS, allowed us the greatest flexibility for the Pi's architecture.

Newer versions of Raspbian encountered the same difficulties as Ubuntu Linux – many packages were unavailable or had significant incompatibilities. Since Intel no longer develops librealsense with the L515 camera in mind, we decided to install an older version of Raspbian OS that would have been widely available when the L515 was released in 2019. We immediately encountered a bootloader block on the Raspberry Pi 4B which did not allow OS systems older than September 2020 to be installed, as that was when our specific Raspberry Pi was manufactured. Ultimately, we decided to install the December 2020 64-bit build of Raspbian Buster to allow the greatest flexibility with all packages. This decision allowed us to experiment with pre-installed Python version 3.7, which we found to be compatible with librealsense and older versions of protobuf and OpenGL.

3.4.2 Intel RealSense L515 as a Depth Camera

The Intel RealSense L515 depth camera's purpose is to capture color and depth image data. From color data, we can visualize an image and verify the chosen contour is indeed a tree branch. From depth data, we can determine the distance and angle to the centroid of the chosen branch contour. The RealSense camera can capture color data as quickly as 60 times per second and depth data as quickly as 30 times per second. As the data is fed to a Raspberry Pi with limited computational power, accepting data too fast can result in a bottleneck where data is not processed quickly enough to provide accurate headings to the drone. To ensure flight safety, we artificially limit depth and image data to one frame every two seconds.

3.4.2.1 Requesting and Obtaining Image

We request an image from the RealSense camera via Intel's pyrealsense2 API, which relies on a pipeline stream of images. To balance image quality with speed, we request color images at 1280x720 resolution and 6 frames per second, and depth images at 1024x768 resolution and 30 frames per second. These represent medium-quality images at the lowest framerate available. Despite these settings, we further artificially limit framerate by only selecting images from the pipeline every 2 seconds – even though the camera provides more images than necessary, the Pi does not access them and therefore does not become bottlenecked.

For indoor flight, we additionally set two hyperparameters – *color camera exposure* to 1250 microseconds and *visual preset mode* to short range. In this fashion, we allow indoor images to be bright enough to be analyzed and prompt the camera's LIDAR depth sensor to focus especially on objects nearby.

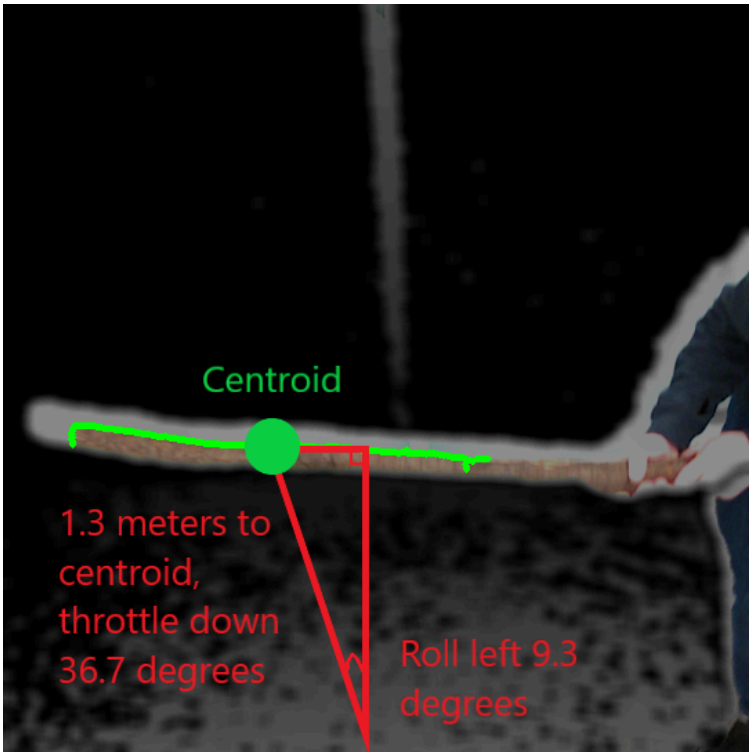


Figure 30: Distance, "throttle," and "roll" angles from our drone to a centroid.

Every two seconds, we take the most recent image frame from the pipeline and store color and depth data as numpy arrays. We scale down the depth array by a scaling factor of 0.00025, a parameter specific to the RealSense L515, to ensure output units are in meters. These arrays are then passed to the vision model, which delivers the distance to the centroid of the branch in the image as well as the "throttle" and "roll" angles that represent the up/down angle and left/right angle respectively from our drone to that centroid.

The image in Figure 30 is a visualization of this process with color and depth images overlaid, and a green line drawn to visualize the contour detected by the vision model. In practice, computations are done on the arrays themselves and no images are saved to the Raspberry Pi to avoid computational expense. We detect a distance of 1.3 meters from our drone to the centroid, represented by the hypotenuse of the triangle. This hypotenuse is angled downwards at 36.7° from horizontal, indicating our drone is above the branch in the image. This can be verified since the branch appears in the lower half of the taken image. Additionally, our drone is offset -9.3° from the centroid, indicating we need to roll left 9.3°. Using trigonometry, we can calculate other ratios as follows:

$$Distance\ Forward = sin(throttleangle) * distance = sin(36.7°) * 1.3 = 0.8m$$
$$Distance\ Down = cos(throttleangle) * distance = cos(36.7°) * 1.3 = 1.0m$$
$$Distance\ Left = sin(|rollangle|) * distance = sin(|-9.3°|) * 1.3 = 0.2m$$

These distances indicate we will need to travel forward (using pitch) a distance 0.8m, travel down (using throttle) a distance 1.0m, and travel left (using roll) a distance 0.2m to reach the centroid of the branch.

3.4.3 Mamba F405 MK2 v2 as a Flight Controller

While the Raspberry Pi serves as the central computer, the Mamba F405 MK2 v2 flight controller serves as the hardware component necessary for sending proper signals to each ESC and onwards to each motor. We selected Ardupilot to serve as the autopilot on the flight controller in order to accept messages in the MAVLink protocol from the Raspberry Pi and pass along signals each ESC could understand.

3.4.3.1 Selecting a Flight Controller

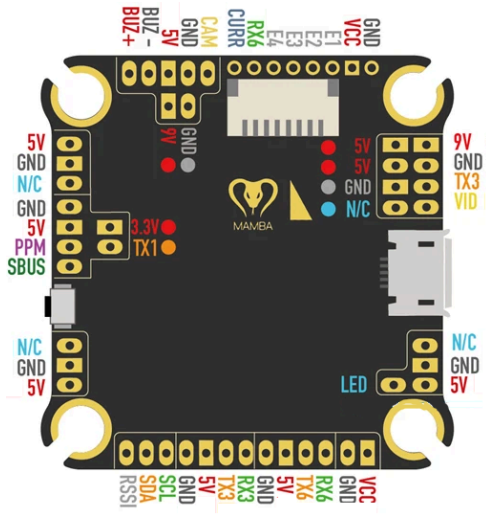
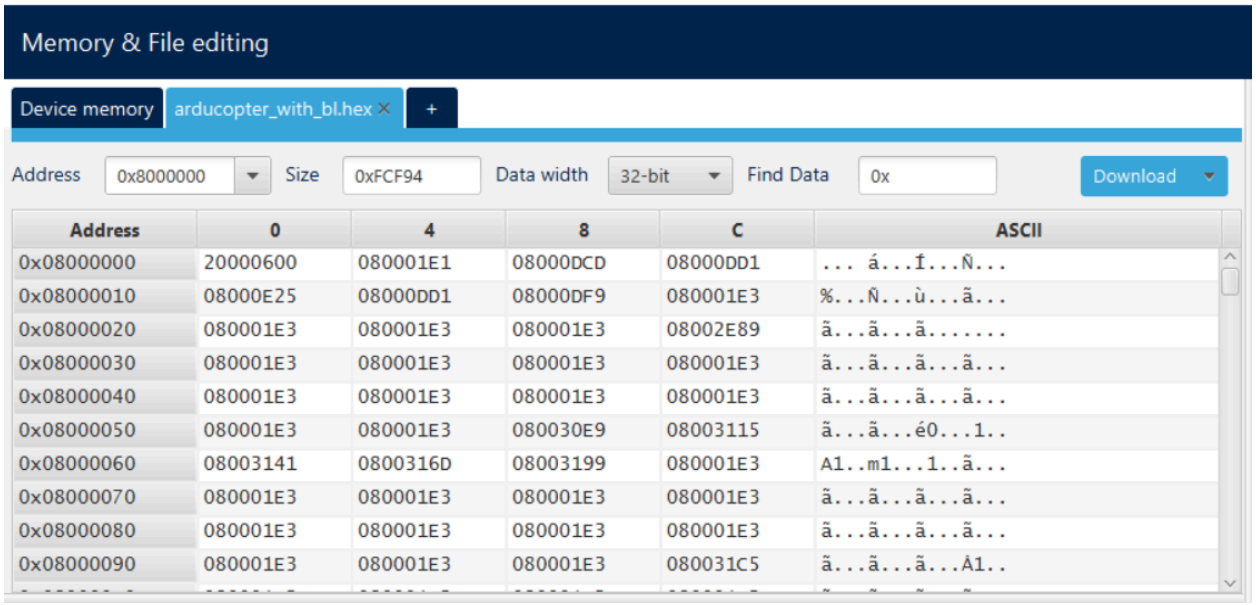


Figure 31: Mamba F405 MK2 v2 (*Mamba F405 MK2 Flight Controller*, n.d.).

We chose the Mamba F405 MK2 v2 flight controller for its affordable cost, surplus UART connections, USB-C UART accessibility, and compatibility with a 4-in-1 ESC platform that would be compatible with our chosen motors. Additionally, the Mamba was well-documented within the Ardupilot autopilot community.

3.4.3.2 Selecting Flight Controller Autopilot



Address	0	4	8	C	ASCII
0x08000000	20000600	080001E1	08000DCD	08000DD1	... á...Í...Ñ...
0x08000010	08000E25	08000DD1	08000DF9	080001E3	%...Ñ...ù...ã...
0x08000020	080001E3	080001E3	080001E3	08002E89	ã...ã...ã.....
0x08000030	080001E3	080001E3	080001E3	080001E3	ã...ã...ã...ã...
0x08000040	080001E3	080001E3	080001E3	080001E3	ã...ã...ã...ã...
0x08000050	080001E3	080001E3	080030E9	08003115	ã...ã...é0...1..
0x08000060	08003141	0800316D	08003199	080001E3	A1..m1...1..ã...
0x08000070	080001E3	080001E3	080001E3	080001E3	ã...ã...ã...ã...
0x08000080	080001E3	080001E3	080001E3	080001E3	ã...ã...ã...ã...
0x08000090	080001E3	080001E3	080001E3	080031C5	ã...ã...ã...Á1..

Figure 32: Flashing Arducopter to flight controller with STM32CubeProgrammer.

Of the flight controller autopilots available, we selected Ardupilot for its wide adoption within the drone flying community, its well-documented simulation and flight parameterization options, and its support of the inputs necessary for autonomous flight.

This design choice required that we flash our flight controller with the Ardupilot bootloader, as the Mamba F405 comes preinstalled with Betaflight. As seen within Figure 32, we installed STM32CubeProgrammer, a tool documented by the Ardupilot community for this purpose, and flashed Arducopter 4.4.2 onto the Mamba. Arducopter is the edition of Ardupilot specific to quadcopters such as our drone, and version 4.4.2 was the latest version at the time of our install in October 2023.

3.4.3.3 Input to Flight Controller

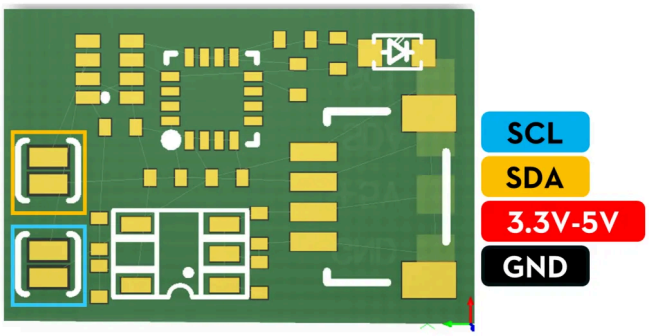


Figure 33: Flywoo NANO v1.0 compass/barometer (*Flywoo BQNANO V1.0 Model W/ Compass & Baro*, n.d.).

In flight, Ardupilot relies on sensor input to stabilize the drone. Although the Mamba F405 contains an internal ICM42688P IMU, it does not contain a built-in compass nor barometer. These inputs allow Ardupilot to approximate local direction and elevation, so to maximize stabilization we purchased the Flywoo BQNANO v1.0 Compass/Barometer chip and mounted it along the front of the drone, connected via I2C. We then calibrated the chip and all other sensors using Ardupilot Mission Planner's built-in setup tools to ensure safe, stable flight.

Figure 34: Flysky FS-i6X radio transmitter (*FS-i6X | Flysky*, n.d.).

While moving towards eventual autonomous flight, we began testing using a radio controller that allowed us to provide roll, pitch, throttle, and yaw directly as PWM values from 1000 (minimum) to 2000 (maximum). We selected the Flysky FS-i6X as our radio transmitter, paired with a FS-iA6B radio receiver, for its affordable cost and compatibility with the Mamba F405 when connected via IBUS. We modified the wiring in the radio transmitter to allow switch #10 (in the top right of the transmitter) to act instead on channel #6 so it could be received by our 6-channel receiver, and configured Ardupilot to interpret it as an "arm-disarm" killswitch that we could pull to immediately halt our drone's propellers in an unsafe situation.

3.4.3.4 Output from Flight Controller

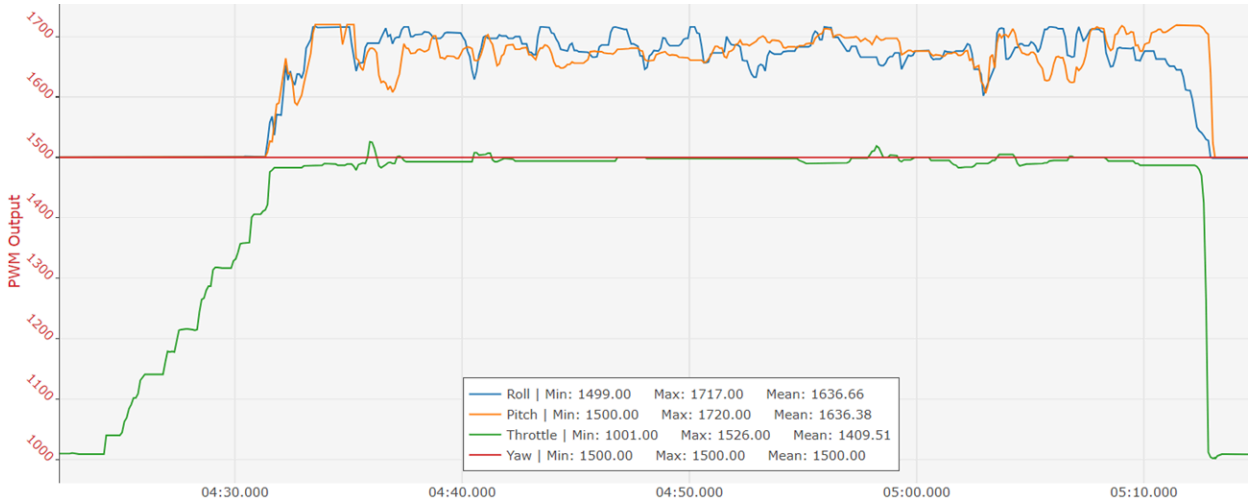


Figure 35: PWM values for roll, pitch, throttle, and yaw from a successful teleoperated flight.

Ardupilot's Mission Planner ground station provides invaluable data feedback that we use to ensure our drone is safe to fly. These include preflight checks of radio connection, battery voltage, and proper sensor inputs – refusing to allow our drone to arm if any conditions are off from expected. We view this output on our local computers either via USB-C connection to the flight controller or through a network connection mirrored from the Raspberry Pi. Ardupilot also saves data logs to the flight controller, which provide us with PWM values for roll, pitch, throttle, and yaw input from the radio controller achieved during teleoperated flight. Graphed on Figure 35 above, we can observe which inputs resulted in which behavior, and mimic those same values to display the same behavior during autonomous flight.

3.4.4 Model for Autonomous Flight

Compared to teleoperated flight, autonomous flight incurs several limitations. As we fly indoors, we do not have access to GPS global position data. Additionally, although we do have a RealSense L515 depth camera, it is not compatible with Ardupilot and thus does not provide local position offsets to the flight controller itself. In this fashion, we are not able to utilize Ardupilot's built-in movement commands since Ardupilot itself is unable to ascertain its location or the objects immediately surrounding it.

In light of these constraints, with distances and angles to a branch's centroid and PWM values for roll, pitch, throttle, and yaw that correspond to values observed during teleoperated flight, we still have all of the components necessary to permit autonomous flight.

3.4.4.1 State Machine for Autonomous Flight

An autonomous drone begins by establishing connection from the Raspberry Pi to Ardupilot via "/dev/ttyACM0," a connection string unique to the Raspberry Pi over the mavutil Python library. We then wait for Ardupilot to acknowledge our connection with a "heartbeat"

packet, and proceed with startup. To ensure flight stability, we establish a state machine with four states, starting in preflight:

- PREFLIGHT – a drone is waiting to be armed or has been disarmed
- TAKEOFF – a drone is arming itself and revving throttle to hover
- SEARCHING – a drone is taking images and adjusting PWM values to locate a branch
- LANDING – a drone is landing by bringing throttle to minimum then disarming

In the takeoff phase, our model sends the ARM MAVLink command via mavutil, waits for Ardupilot to acknowledge our request, then initiates takeoff by progressively revving throttle higher in increments of 100 PWM every second until HOVER_THROTTLE, a constant specific to the physics of each drone, is reached. We take 1500 PWM to be our default HOVER_THROTTLE, as it is the median of our 1000 PWM minimum and 2000 PWM maximum.

```
self.drone.mav.rc_channels_override_send(self.drone.target_system,
                                         self.drone.target_component,
                                         self.roll,
                                         self.pitch,
                                         self.throttle,
                                         self.yaw,
                                         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

Figure 36: Sending a “flight pulse” via RC_OVERRIDE in the searching phase.

In the searching phase, after taking an image with the pipeline introduced in Section 3.4.2.1, we use the calculated distance forward, down, and left to change pitch, throttle, and roll respectively by the RC_OVERRIDE MAVLink packet, keeping mind of the HOVER_THROTTLE value of 1500 PWM and scaling up or down from 1500 by (10 * (distance in meters)), where 10 is a scalar dependent on observed behavior of a particular drone. This mimics the same radio controller (RC) input we provide in teleoperated flight, so by observing how large particular teleoperated inputs are required to move our drone a fixed amount, we determine the proper scalar for each input to duplicate observed behavior. We additionally introduce a constant maximum change (10 PWM), so that no one outlier image causes our drone to fly in an unsafe manner. These “flight pulses” are sent every two seconds after the vision model has processed an image. If we are within a certain constant horizontal distance from the centroid of a branch, we will attempt landing by lowering throttle. If not, we wait until the next image to move further.

In the landing phase, our model progressively lowers throttle by increments of 100 PWM every second until the minimum throttle PWM of 1000 is reached. We then initiate disarming through the DISARM MAVLink command, and return to the preflight phase.

3.4.4.2 Simulating Autonomous Flight

Figure 37: A simulated drone on the ground in Mission Planner SITL.

Before flying our drone autonomously, we needed to ensure it would fly safely by moving in the direction and angles we expect when provided with certain RC_OVERRIDE packets. To evaluate safe flight, we simulated our drone under Ardupilot's Mission Planner SITL physics engine. We chose this engine since it is within the same software as we use to fly our actual drone and contains the same data output. Although this simulation does not model the physics of landing on a branch, it allows us to test takeoff, flight behavior, and landing. Each PWM value sent to the simulated drone is a scaled value from the one that would be sent to the actual drone, as the simulated drone operates under ideal conditions without the weight or size constraints of our drone design and grasping mechanism.

Since Mission Planner SITL is not designed for Microsoft Windows, the simulation operates under a pipeline from Windows Subsystem for Linux (WSL):

1. On Windows, start an XWindows instance to allow data to be transferred to Windows from WSL
2. In WSL, run a simulated vehicle as indicated in Ardupilot's SITL documentation
3. On Windows, open Mission Planner and connect via TCP to the simulated drone
4. Access Mission Planner's MAVLink Mirror feature and open a UDP Client on port 14540
5. Run the drone state machine with connection string "udp:localhost:14540"

4. Results

4.1 Base Drone

4.1.1 Drone Hardware

In terms of our set thrust to weight goal of 2:1 we ended up overshooting with a ratio of 8:1. The center of mass ended up being within tolerance as seen in Figure 38. During testing battery life was estimated to be 10 minutes. This extended our initial goal of 5 minutes by roughly double.

In order for the quadcopter to be stable in flight and also stable on the ground the center of mass generally needs to lay underneath the flight controller and is as centered as possible in the roll and pitch axis. This was achieved in the roll axis as seen in Figure 38a while the pitch axis is tilted slightly towards the camera side of the quadcopter as seen in Figure 38b.

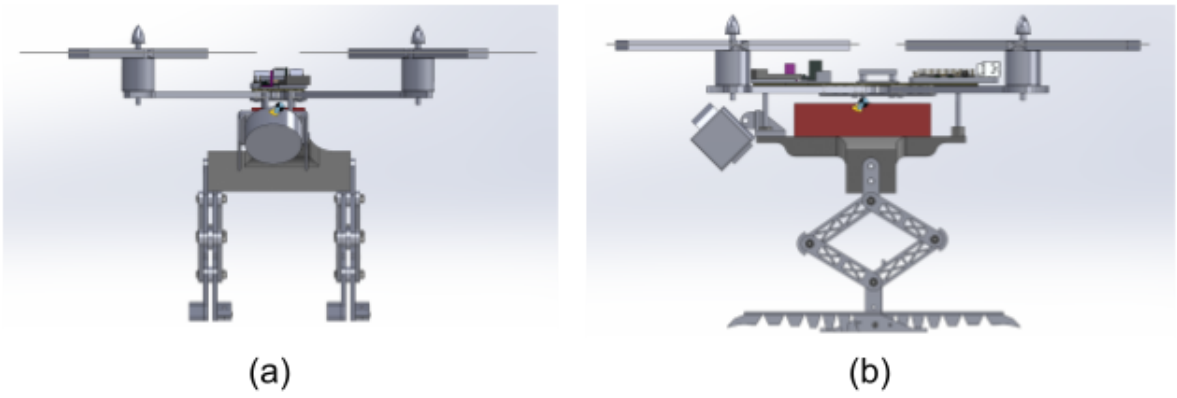


Figure 38: Center of mass of the quadcopter in a front view and a side view.

4.1.2 Drone Flight

Overall the drone was able to achieve multiple stable flights via teleoperation. One of these flight's input parameters is shown in Figure 35 the quadcopter's throttle is slowly increased from 1000 PWM to our take off speed of 1483 PWM. Once the quadcopter is airborne the pitch and roll are then tweaked in order to keep the drone stable while attempting to land on a tree branch which were manually adjusted by the operator using a remote controller.

4.2 Grasping Mechanism

The grasping mechanism is able to support the drone on flat and natural surfaces, even returning to standing equilibrium from pitch of -8.5° to 8° from horizontal. It is capable of fully

grasping objects with diameters between 31 and 60mm. The grasping mechanism as a whole weighs 224g, making it 12.4% of the drone's entire weight. The drone is capable of maintaining a stable perch on a branch as shown in Figure 39, without power.

Figure 39: The quadcopter perched on a branch without power.

4.3 Vision Model

Due to the limitations of both models described in section 3.3.2, we decided to use the binary classification model for branch identification. Training each individual model with 10-fold cross validation our results show average accuracy, precision, recall, and f1-score for each model.

Table 3 - Binary Classification Model Average Scores:

Model	**Accuracy**	**Precision**	**Recall**	**F1**
Logistic Regression	0.891	0.674	0.315	0.429
Naive Bayes	0.785	0.350	0.761	0.480
SVM	0.890	0.684	0.288	0.406
kNN @ k=2	0.967	0.852	0.906	0.878
Random Forest	0.979	0.984	0.853	0.913

We found that the Random Forest Classifier performed the best on the training data provided with a precision of 98.4%. While testing with additional data, the model was unable to identify

any new images as branches. The model severely overfit to the data provided during training and was unable to accept any images taken from the RealSense camera.

Although branch identification did not succeed, the branch detection algorithm was able to accurately detect the location of a branch, assuming it was the main object in the image. We tested the model in real time in a “walking simulation” where we physically walked the drone towards a tree branch to simulate the model’s use in flight. When the branch was the main object in the image, the model correctly identified the contour associated with the branch. It then correctly returned the distance to the branch, as well as the angle to the centroid of the contour.

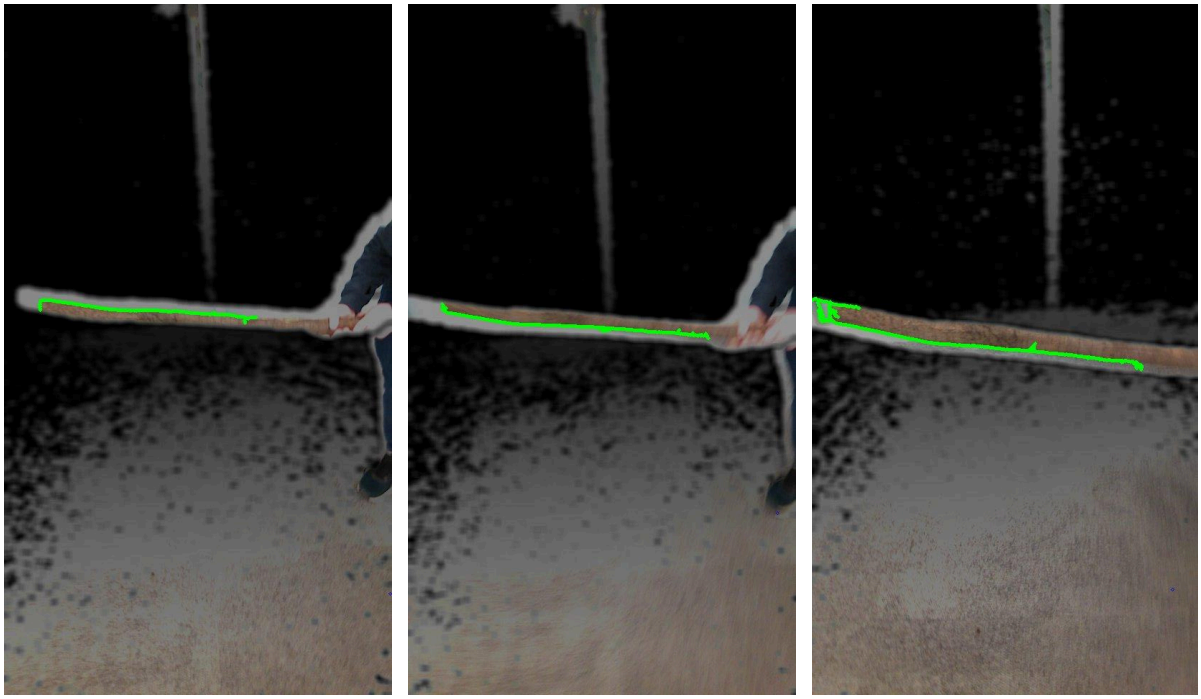


Figure 40: The vision model correctly detecting the branch in 3 consecutive images

4.4 Communication and Autonomous Simulation

In the design of our in-flight communications, we established the following three metrics on which to evaluate our results:

1. All components communicate without protocol or version failures.
2. Autonomous pipeline successfully takes an image, receives it from the vision pipeline, and calculates proper angles and directions from that image
3. Autonomous flight model provides safe inputs with proper force and direction to simulated and actual drone.

4.4.1 Success of Communication Protocols

The quadcopter contains components with precise interactions specific to particular versions of Python. Through the following table, we establish which protocols we experimentally determined were compatible with one another when communicating between the Raspberry Pi and the RealSense vision camera:

Table 4 – Raspberry Pi and Depth Camera Communication Iterations:

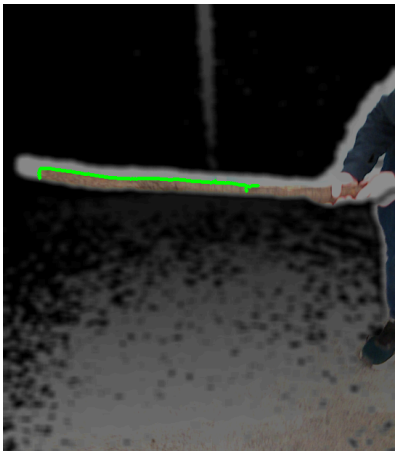
Operating System	**Python Version**	**pyrealsense2 version**	**L515 officially supported?**	**Successful Communication**
Ubuntu Desktop 22.04.4 LTS	Python 3.10	2.54.1, from source using Ubuntu walkthrough	Not tested by Intel, but recognized	No – unable to compile due to dependency incompatibilities
Raspberry Pi OS Bookworm (October 2023)	Python 3.11	2.54.1, from source using Raspberry Pi walkthrough	Not tested by Intel, but recognized	No – unable to compile due to dependency incompatibilities
Raspbian Buster (September 2019)	Python 3.7.3	N/A	N/A	No – unable to pass bootloader
Raspberry Pi OS Buster (December 2020)	Python 3.7.3	2.54.1, from source using Raspberry Pi walkthrough	Not tested by Intel, but recognized	Yes – when built from source and with precompiled wheels

Thus, with a system *old enough* to support Python 3.7 while *new enough* to interpret Intel's RealSense camera library, we establish successful communication without protocol or version failures between the Raspberry Pi and the camera.

Additional communication was successfully achieved via Arducopter 4.4.2 and configuration of Arducopter to permit I2C compass and barometer input and IBUS radio controller input.

4.4.2 Success of Autonomous Pipeline

We evaluate the success of our autonomous pipeline by successfully taking an image, receiving it from the vision pipeline, and calculating proper angles and directions from that image frame.



```
Getting a frame...
Throttle Angle: 36.708984375;
Roll Angle: -9.309444444444445;
Horizontal distance: 0.8062165155750499;
Vertical distance: 1.0812685444984425;
Hypotenuse distance: 1.3487500640621874
```

Figure 41: Our autonomous pipeline takes an image, receives it from the vision pipeline, and calculates proper angles and directions.

Table 5 – Calculation of Angles and Directions After Image Frame

Image Number	Throttle Angle (deg)	Roll Angle (deg)	Hypotenuse Distance (m)	Horizontal Distance (m)	Vertical Distance (m)
Image 1	25.839	17.187	1.859	0.810	1.673
Image 2	36.709	-9.309	1.349	0.806	1.081
Image 3	35.137	-1.289	1.254	0.722	1.026
Image 4	34.453	-8.880	0.970	0.550	0.800
Image 5	3.281	-2.005	1.216	0.070	1.214

From the vision model's provided throttle angle (up/down), roll angle (left/right), and hypotenuse distance (distance from drone to branch centroid), we calculate the horizontal distance a branch is in front of the drone as well as the vertical distance a branch is below the drone. Based on the "walking test" conducted in Section 4.3, we collect these angles and a hypotenuse distance for five image frames and calculate horizontal and vertical distances based on these ratios. Visual representations of Images 2, 3, and 4 can be seen in Figure 40 above.

Thus, we can be sure the autonomous pipeline successfully calculates proper angles and directions from a provided image.

4.4.3 Progression of Autonomous Flight Model

We evaluate the success of our autonomous flight model by ensuring it provides safe inputs with proper force and direction to both the simulated and actual drone. Since we had not fully fine-tuned our drone for teleoperated flight by project submission in April 2024, we denote autonomous flight on the drone as a future expansion and instead focus on evaluating the success of simulated flight.

Figure 42: Simulated flight within Mission Planner SITL.

The simulated flight in Figure 42 denotes a mission providing the same throttle as observed by our drone in teleoperated flight (1500 PWM), but more narrow roll and pitch (1450-1550 simulated PWM compared to 1400-1600 teleoperated PWM). This mission simulates the drone taking off until it reaches hover throttle, then “finding a branch” through taking five sequential simulated images that provide a simulated shrinking distance and angle in front of its initial hover throttle location, to the left and down from the drone’s hover position. We observe an altitude of 8.03 meters at “image four” where the drone rolls a slight angle left at ground speed 0.05m/s and throttles downwards at vertical speed -0.70m/s.

These data show that the same observed RC input to our quadcopter in teleoperated flight results in a drastically increased range of movement – the simulated drone gains and changes altitude at a much greater rate than the actual drone, which would reach approximately 2.0m in altitude at the same hover throttle. We narrow the PWM range for roll and pitch, but observe that

the drone moves significantly further left than would be expected from the same PWM on our actual drone. In this fashion, we observe that the direction commanded by the autonomous pipeline is properly displayed by the simulated drone moving left and down, but the force provided results in much greater changes than the actual drone displays on teleoperated flight with those inputs.

Table 6 – Calculation of Roll, Pitch, Throttle, and Yaw From Distances and Angles Based on Walking Simulation

Image Number	Roll (PWM)	Pitch (PWM)	Throttle (PWM)	Yaw (PWM)
HOVER	1500	1500	1500	1500
Image 1	1510	1516	1490	1500
Image 2	1496	1516	1490	1500
Image 3	1500	1510	1490	1500
Image 4	1498	1510	1490	1500
Image 5	1500	1501	1490	1500

In practice, we observe autonomous PWM ranges through the walking simulation from Section 4.3, with observed image frame angles and distances as in Table 5. Visual representations of Images 2, 3, and 4 can be seen in Figure 40 above. Relative to safe hover values of 1500 PWM, we simulate the PWMs necessary to have our drone fly downwards as throttle is decreased to 1490. We also mimic our drone flying right, then addressing an overcorrection by moving slightly left, relative to the initial throttle roll before stabilizing at 1500. We finally observe pitch increasing sharply to move forward, then decreasing to near 1500 as horizontal distance to a branch decreases.

5. Conclusions

While each individual subsystem functioned independently, we were unable to fully integrate them into a drone capable of autonomous flight to grasp a tree branch. The drone is able to fly under indoor conditions, but experiences side-to-side motion that makes controlled flight challenging. Further fine-tuning is necessary to ensure stable teleoperated flight before progressing to the autonomous model. The grasping mechanism is able to perch on a tree branch when provided with an additional downward force. The weight of the drone alone is unable to fully collapse the legs of the grasping mechanism, but during the landing sequence the drone's propellers could be activated to provide additional downward force. The branch identification model is unable to identify tree branches due to the limitations of each of the proposed models. On the other hand, the branch detection model is successful in identifying the location of a branch and returning its relative distance and direction. The autonomous model is able to simulate flight in the expected directions, but this flight does not mimic the physics of our drone and requires greatly scaled inputs to simulate the same behavior observed in teleoperated flight.

5.1 Future Work

To improve upon each subsystem, in future iterations of this project we recommend that researchers consider the following:

1. **Drone Hardware:** Utilize alternative sensor inputs such as outdoor-rated depth cameras and GPS components for use in outdoor flight.
2. **Grasping Mechanism:** Explore adding a locking mechanism to the legs to ensure a more robust grasp.
3. **Branch Identification:** Investigate developing a model that is capable of classifying branches either by using alternative approaches or by reducing the effects of the limitations on previous approaches.
4. **Branch Detection:** Utilize contours to calculate the ideal landing spot on a branch.
5. **Autonomous Pipeline:** Integrate newly released computers such as the Raspberry Pi 5 that can allow for more efficient computation without a self-imposed two second camera delay.

References

Ahmad, N., Ghazilla, R. A. R., Khairi, N. M., & Kasi, V. (2013). Reviews on various inertial measurement unit (IMU) sensor applications. *International Journal of Signal Processing Systems*, *1*(2), 256-262.

Alex. (2017, October 27). *The Complete Guide to Buying an FPV Quadcopter Frame*. Unmanned Tech Blog. https://blog.unmanned.tech/fpv-quadcopter-frame-buying-guide/

Alkadhim, S. A. S. (2019). Communicating with raspberry pi via mavlink. *Available at SSRN 3318130*.

Aryee, B. N. A. (2001). Ghana's mining sector: its contribution to the national economy. *Resources Policy*, *27*(2), 61–75. https://doi.org/10.1016/s0301-4207(00)00042-8

Atoev, S., Kwon, K. R., Lee, S. H., & Moon, K. S. (2017, November). Data analysis of the MAVLink communication protocol. In *2017 International Conference on Information Science and Communications Technologies (ICISCT)* (pp. 1-3). IEEE.

Brand, I., Roy, J., Ray, A., Oberlin, J., & Oberlix, S. (2018, October). Pidrone: An autonomous educational drone using raspberry pi and python. In *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 1-7). IEEE.

The Buzz on Native Bees: U.S. geological survey. The Buzz on Native Bees | U.S. Geological Survey. (n.d.). https://www.usgs.gov/news/featured-story/buzz-native-bees

Canny, J. (1986). A computational approach to edge detection. In *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8*(6), 679-698.

Chao, H., Cao, Y., & Chen, Y. (2010). Autopilots for small unmanned aerial vehicles: a survey. *International Journal of Control, Automation and Systems*, *8*, 36-44.

Chi, W., Low, K. H., Hoon, K. H., & Tang, J. (2014). An Optimized Perching Mechanism for Autonomous Perching with a Quadrotor. 2014 IEEE International Conference on Robotics and Automation (ICRA). https://doi.org/10.1109/icra.2014.6907306

DJI. (2018, August 23). DJI Introduces Mavic 2 Pro And Mavic 2 Zoom - DJI. DJI Official. https://www.dji.com/newsroom/news/dji-introduces-mavic-2-pro-and-mavic-2-zoom

Doyle, C. E., Bird, J. J., Isom, T. A., Kallman, J. C., Bareiss, D. F., Dunlop, D. J., King, R. J., Abbott, J. J., & Minor, M. A. (2013). An avian-inspired passive mechanism for quadrotor perching. IEEE/ASME Transactions on Mechatronics, 18(2), 506–517. https://doi.org/10.1109/tmech.2012.2211081

Drone Eger. (2020, January 1). HOW TO CHOOSE MOTOR AND PROPELLER FOR QUADCOPTER. One-Stop Drone Parts Store. Save BIG. https://www.droneassemble.com/how-to-choose-motor-and-propeller-for-quadcopter/

Donkor, A. K., Nartey, V. K., Bonzongo, J. C., & Adotey, D. K. (2006). Artisanal Mining of Gold with Mercury in Ghana. Www.wajae.org; African Journals Online. https://www.ajol.info/index.php/wajae/article/view/45666/29146

Encyclopædia Britannica. (n.d.). Perching Mechanism of a Pigeon. Encyclopædia Britannica. Retrieved April 22, 2024, from https://www.britannica.com/animal/bird-animal/Flight#/media/1/66391/44424.

Flywoo BQNANO V1.0 Model w/ Compass & Baro. (n.d.). Flywoo. Retrieved April 21, 2024, from https://flywoo.net/products/flywoo-bqnano-v1-0-model-w-compass-baro-0-6g

FS-i6X | Flysky. (n.d.). Flysky. Retrieved April 21, 2024, from https://www.flysky-cn.com/fsi6x
Hadi, G. S., Varianto, R., Trilaksono, B. R., & Budiyono, A. (2014). Autonomous UAV system development for payload dropping mission. *Journal of Instrumentation, Automation and Systems*, *1*(2), 72-77.
Hell, P., Mezei, M., & Varga, P. J. (2017, January). Drone communications analysis. In *2017 IEEE 15th International Symposium on Applied Machine Intelligence and Informatics (SAMI)* (pp. 000213-000216). IEEE.
Hilson, G., & McQuilken, J. (2014). Four decades of support for artisanal and small-scale mining in sub-Saharan Africa: A critical review. *The Extractive Industries and Society*, *1*(1), 104–118. https://doi.org/10.1016/j.exis.2014.01.002
Hollowell, W. (2024, January 9). *How to choose motor and propeller for quadcopter*. Medium. https://medium.com/@hollowell0869197/how-to-choose-motor-and-propeller-for-quadcoter-97630a9e92d7
Gustafsson, J., & Mogensen, D. (2023). Streamlining UAV Communication: Investigating and implementing an accessible communication interface between a ground control station and a companion computer.
Intel. (n.d.). *Intel® RealSense™ LiDAR Camera L515*. Intel® RealSense™ Depth and Tracking Cameras. https://www.intelrealsense.com/lidar-camera-l515/
Liang, O. (2023b, February 27). *The Ultimate Guide to FPV Drone Propellers: How to Choose the Best Props for Your Quadcopter*. Oscar Liang. https://oscarliang.com/propellers/
Kan, M., Okamoto, S., & Lee, J. H. (2018, March). Development of drone capable of autonomous flight using GPS. In *Proceedings of the international multi conference of engineers and computer scientists* (Vol. 2).
Kawabata, S., Nohara, K., Lee, J. H., Suzuki, H., Takiguchi, T., Park, O. S., & Okamoto, S. (2018). Autonomous flight drone with depth camera for inspection task of infra structure. In *Proceedings of the International MultiConference of Engineers and Computer Scientists* (Vol. 2).
Kjærgaard, M. B., Blunck, H., Godsk, T., Toftkjær, T., Christensen, D. L., & Grønbæk, K. (2010). Indoor positioning using GPS revisited. In *Pervasive Computing: 8th International Conference, Pervasive 2010, Helsinki, Finland, May 17-20, 2010. Proceedings 8* (pp. 38-56). Springer Berlin Heidelberg.
Kumar, G. P., & Sridevi, B. (2019). Simulation of Efficient Cooperative UAVs using Modified PSO Algorithm. *WSEAS Trans. on Information Science and Applications*, 94-99.
Mamba F405 MK2 Flight Controller. (n.d.). Ardupilot. Retrieved April 21, 2024, from https://ardupilot.org/copter/docs/common-mamba405-mk2.html
Molina, J., & Hirai, S. (2017a). Kinematic Analysis of a Novel Skew-Gripper for Aerial Pruning Tasks. *Proceedings of the 3rd International Conference on Mechatronics and Robotics Engineering*. https://doi.org/10.1145/3068796.3068802
Molina, J., & Hirai, S. (2017b). Pruning Tree-Branches Close to Electrical Power Lines Using a Skew-Gripper and a Multirotor Helicopter. *2017 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*. https://doi.org/10.1109/aim.2017.8014169

Nadan, P. M., Anthony, T. M., Michael, D. M., Pflueger, J. B., Sethi, M. S., Shimazu, K. N., Tieu, M., & Lee, C. L. (2019). A bird-inspired perching landing gear system1. *Journal of Mechanisms and Robotics*, *11*(6). https://doi.org/10.1115/1.4044416
Pütsep, K., & Rassõlkin, A. (2021, October). Methodology for flight controllers for nano, micro and mini drones classification. In *2021 International Conference on Engineering and Emerging Technologies (ICEET)* (pp. 1-8). IEEE.
Randall, B. (2022, June 6). *The value of birds and bees*. Farmers.gov. https://www.farmers.gov/blog/value-birds-and-bees
Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
Roderick, W. R., Cutkosky, M. R., & Lentink, D. (2021). Bird-inspired dynamic grasping and perching in arboreal environments. *Science Robotics*, *6*(61). https://doi.org/10.1126/scirobotics.abj7562
Sanjukumar, N. T., & Rajalakshmi, P. (2022, September). Obstacle Detection and Collision Avoidance on UAV using Rangefinder Sensor with Kalman Filter Technique. In *2022 IEEE Global Conference on Computing, Power and Communication Technologies (GlobConPT)* (pp. 1-6). IEEE.
Silva, R., Junior, J. M., Almeida, L., Gonçalves, D., Zamboni, P., Fernandes, V., Silva, J., Matsubara, E., Batista, E., Ma, L., Li, J., & Gonçalves, W. (2022). Line-based deep learning method for tree branch detection from Digital Images. *International Journal of Applied Earth Observation and Geoinformation, 110,* 102759.
Yang, Z., Lin, F., & Chen, B. M. (2016, October). Survey of autopilot for multi-rotor unmanned aerial vehicles. In *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society* (pp. 6122-6127). IEEE.
Mohsan, S. A. H., Othman, N. Q. H., Li, Y., Alsharif, M. H., & Khan, M. A. (2023). Unmanned Aerial Vehicles (UAVs): Practical aspects, applications, Open challenges, Security issues, and Future Trends. *Intelligent Service Robotics*, *16*(1). https://doi.org/10.1007/s11370-022-00452-4
Oscar Liang. (2023a, April 28). *A Comprehensive Guide to FPV Drone Frames*. Oscar Liang. https://oscarliang.com/fpv-drone-frames/#Plus-Frame
Sinsky, C. A., Bavafa, H., Roberts, R. G., & Beasley, J. W. (2021). Standardization vs Customization: Finding the Right Balance. *The Annals of Family Medicine*, *19*(2), 171–177. https://doi.org/10.1370/afm.2654
SunnySkyUSA. (n.d.). *SunnySky X2814 Brushless Motors*. SunnySky USA. Retrieved April 23, 2024, from https://sunnyskyusa.com/products/sunnysky-x2814-brushless-motors?variant=45672358991
Syracuse schools. (2024). *Universal Pre-K / Homepage*. Www.syossetschools.org. https://www.syossetschools.org/SyoUPK
Tatu Gens. (2017, October 11). *What is difference between X-Frame and H-Frame?* GensTattu. https://genstattu.com/blog/what-is-difference-between-x-frame-and-h-frame
United States Department of Agriculture. (n.d.). *Honey Bees*. Www.usda.gov. https://www.usda.gov/peoples-garden/pollinators/honey-bees

United States Department of Agriculture (USDA), National Agricultural Statistics Service (NASS), & Agricultural Statistics Board, Honey Bee Colonies (2023).
Vector Technics. (2023, January 23). *What is "Kv" of UAV BLDC Motor ?* Www.linkedin.com. https://www.linkedin.com/pulse/what-kv-uav-bldc-motor-vectortechnics
Walker, I. D., Dawson, D. M., Flash, T., Grasso, F. W., Hanlon, R. T., Hochner, B., Kier, W. M., Pagano, C. C., Rahn, C. D., & Zhang, Q. M. (2005). Continuum robot arms inspired by cephalopods. *SPIE Proceedings*. https://doi.org/10.1117/12.606201

Appendix A: A Comparison Between Premade and Custom Drones for Environmental Robotics

Environmental robots is the use of robotic systems to help, protect, improve, and survey various dangerous or hard-to-reach environments. These environments could potentially be hazardous to humans due to potential pollutants, disrupt the natural ecosystem, or the risk while traveling to these locations is high. These hard-to-reach or potentially dangerous places could provide vital information on how to reduce our emissions, counteract current global climate, or observe an endangered species and it is critical we find a cost effective way to access these locations. Hopefully the findings below can progress this goal. This research can also be used to evaluate both healthy and unhealthy ecosystems to determine what flora and fauna are present, endangered, or even absent. This type of surveying can help researchers get a more accurate state of a given environment.

Currently, not one robot fits all solutions which means for each task a different type of robot will be better suited for a task. For example, a quadcopter has great maneuverability and is able to fly allowing for superb aerial surveillance. However, these quadcopters do not have the longest battery life or the largest payload meaning large sensors or studies that require a long period to pass would not necessarily be a strong suit. On the flip side, a stationary robot that collects and sorts out trash and plastics is able to be as large as it needs to be and does not necessarily depend on battery power allowing it to run for as long as needed. The downside of this solution is that it takes a significant number of resources to construct and or move these types of robots.

Just like how the type of robot shapes what the project can do, the way these robots are developed and sold can have serious implications on the research and development process. The main choices are commercial off-the-shelf (COTS) and custom designs. When deciding which type of project is needed, the main criteria that researchers must consider are:

Scope: does the project need a specific hardware or software requirement?

Customizability vs Scalability: How much modification does a robot need in order to complete the mission and how easily can the solution be sold/used by the masses

Development time: what is the time frame of a project?

Cost: how much money or manpower can be spent on development vs research

Technical knowledge requirement: How skilled does the operator/customer need to be in order to pilot the robot?

Use case 1: Environmental Mapping

Figure 43: DJI Mavic 2 Pro (DJI, n.d.)

Mining is one of the most important yet environmentally destructive industries on earth. This is due to the nature of needing to dig into the ground and extract resources at high volumes. This is especially true in Ghana where up to 40% of the country's economy comes directly from mining, more specifically gold mining (Ayree, 2000).

Artisanal gold mining is small-scale gold mining. There are a variety of methods used however they all have some common practices. These methods primarily differ with the initial extraction of material. Once the (mainly) gold chunks are extracted they are combined with mercury inorder to form a gold mercury amalgamate (Donkor et al., 2006). This removes any undesired material and increases the size of the gold pieces. Once all of the gold mercury amalgamate is sorted out the mercury is buried away. This leaves the miners with pure gold which is the basis of their income. During this process however the miners are exposed to mercury as most of the time there is a degree of handling the mercury gold amalgamate. The potentially more damaging exposure happens when the mercury is boiled away as it can get inhaled and it can spread through the environment. Once a mining site is closed the equipment

and materials are generally cleaned up and stored for the next mining operation. This includes as much of the mercury as the miners can manage, however the mercury that gets mixed with the soil and water of the mining processes is left as environmental contaminants (Donkor et al., 2006). This mercury contamination is generally near rivers and other bodies of water as that is where the majority of gold is found. This process leads to a severely contaminated environment and water supply for humans, animals, and plants. The sad reality is that the miners know they are poisoning themselves and those around them, however, they need the income in order to survive (Hilson et al.,2014).

Due to safer mining practices not being financially feasible on the scale of these artisans, a quicker and more efficient remediation effort could be put into place so that the miners can still work while limiting the mercury contamination the environment is exposed to. One way to do this would be to use a drone to map mine sites, identify where highly contaminated soil is, and help direct remediation efforts. This can be combined with the local knowledge of the land in order to minimize the exposure and time required to remediate. In order for the drone to be most effective it must be able to autonomously map these mining sites and identify where areas of contamination are.

Use Case 2: Environmental Supplements

Figure 44: The BranchBot quadcopter

According to the United States Department of Agriculture bees are responsible for pollinating around 130 types of plants sold in grocery stores and help generate 15 billion dollars each year (United States Department of Agriculture, 2017). Even though pollinators are incredibly valuable their populations have been decreasing at an alarming rate. This is partially due to climate change, habitat loss, and a variety of diseases (for more information see the Introduction, section 1). The main objective of this project is to use robotics to help supplement declining bee populations so that the ecosystem does not collapse while they are recovering. In order to do this as unobtrusively as possible a custom quadcopter was created in order to perch on tree branches and hold a beehive. While this is the main purpose of the quadcopter, it can also be used as a mobile environmental station that can perch in trees.

Scope: does the project need a specific hardware or software requirement?

In general, it is easier to create and distribute a software system than it is a hardware system due to the physical manufacturing that is required. However if the hardware does have the physical capacity to complete the mission the software will not be able to change it. When deciding whether or not to use a premade drone for an application this needs to be taken into account. The next step is to determine what software is needed to complete the mission. During this step, software should be checked to see if they are compatible with each other as this could cause issues down the line. Both case studies experience both similar and drastically different errors in order to get the quadcopters off the ground. For example, for both projects outdated versions of Python and firmware needed to be used in order to integrate with the oldest hardware devices. In the Ghana project's case, the DJI Mavic Pro 2's firmware was no longer actively supported by DJI which meant that older versions of supporting software had to be used like Android Studio and OpenCV. In the case of BranchBot the Intel RealSense L515 camera recently lost some support from Intel which caused the version of Python run on the Raspberry Pi to be slightly outdated. The difference however is that the Mavic Pro 2, experienced no major hardware issues during testing. The primary reason for this is the robust design of the quadcopter and the failsafe in the code that made it this safe. For BranchBot due to the custom build and code on our flight controller, there were significant issues with the integration of software and hardware. This caused the quadcopter to break on multiple occasions that cost the team money

and time. Primarily we found out partway through the project that the quadcopter needed additional sensors (a barometer and RC receiver).

These types of issues can be dealt with by anyone with enough time, however, those with a software background generally can debug and streamline software problems more easily than mechanically focused people. The inverse is true as well, that hardware-focused people would be able to identify and correct hardware issues significantly easier than a software-oriented person would be able to. This means that if developing a custom drone it would be highly beneficial to have a diverse team with both software and mechanical members on it or generally well-rounded people. On the flip side, if using a COTS solution it would be recommended that the team is primarily composed of software-oriented individuals.

Customizability vs Scalability: How much modification does a robot need in order to complete the mission and how easily can the solution be sold/used by the masses.

Scalability is the ability to repeatedly manufacture the same product over and over again. For this to work the product has to be relatively standard with possible small changes the user can request. The main benefit of having a very scalable product is that the customer knows exactly what they're going to get and they know it will be decent for the use case they have in mind (Sinsky, et al., 2021). This can be seen with the DJI Mavic Pro 2 that was used in the Ghana gold mining project as the quadcopter was an excellent system for flying around and taking pictures however the minimum shutter speed of the camera being 2 seconds made the system not 100% ideal for the mission at hand. On the same note, the DJI drone was incredibly robust and hard to break under normal conditions. This is due in part to the robust engineering and collision avoidance built into the quadcopter. There are also part replacements for this quadcopter online that should theoretically work two if part of the frame is damaged. The drawback to this method however is that if there are customizations that the user would like to make it may be more difficult to do so, even if it is within the use case of the drone, for example, rapid photo taking. This could be seen in the Ghana project when the camera would only take a photo every two seconds and there was not much that could be done about it. Along with a convoluted way of accessing and saving the image led to a slowdown in both development and

execution of the end product. Another benefit of standard designs is that the time between ordering and receiving the product is minimal as the drone design is already made and it either just needs to be manufactured, or it is already made.

The main draw of a custom solution is that it can be used to solve highly unique problems or challenges where a premade drone or part could not work without extensive modification (Sinsky, et al., 2021). A custom drone allows the creators to optimize their drone to supplement the features that are critical while ignoring other features that may be conventionally important. For example, with BranchBot the goal of the project was to identify tree branches and then land on said tree branches. No pre-made drone that we could find had either the capacity to add to the design to create the grasping mechanism or have a grasping mechanism already attached. This led to the creation of a custom drone as it allowed us to easily modify the design of a standard quadcopter to add mounting points for the gripping mechanism.

Eventually, both drones would ideally be mass-producible and usable across a country due to the nature of their work. However, the difference is that using a pre-made drone for the Ghana project would allow those who already have a compatible system to use the software created for the Mavic Pro 2 to be able to download the app and then use it on their system alleviating the initial cost of buying one of these systems. For branch bot, the customers would have to buy the entire system however it would be able to complete the very niche problem it was set out to solve.

Development time: what is the time frame of a project?

In general, it would be recommended that the shorter the time frame the more COTS components that are used. This is because research and development of both hardware and software takes a significant amount of time and effort. This can be seen through the creation of our custom quadcopter for branch bot. From the time that we decided to create a custom drum to the time that we were first able to fly it took roughly 14 weeks. After these 14 weeks, some issues still had to be sorted out on the hardware's end. For the pre-bought DJI Mavic Pro 2, it only took five weeks or so for the custom application to be able to have the drone take off. This

time frame was not represented in the amount of time that the programs had as the Ghana project had a significantly loser time frame starting at the beginning of the summer and going to roughly the end of the calendar year where branch bot had exactly one school year to complete the project. In an ideal world these time frames would be swapped as custom projects tend to take more time and are more prone to canonical failure leading to significant delays. This was seen in BranchBot during the development of our quadcopter when we were soldering on a Bluetooth controller; this controller was to be used as a secondary communication system to ensure constant communication. During this process, however, the wires were connected improperly causing our flight controller to burn out and break. This led to us needing to spend an additional $70 to replace the parts and spend additional time waiting and redoing a significant amount of work to get the flight controller back to where it was both with hardware and software. This sort of failure never occurred in the Ghana project, however, if something did break it would have caused significantly more delays as the Mavic's parts are not as readily available as the custom parts and are significantly more expensive.

Cost (money and man hours): How much money can be spent on development vs research?

In general, buying a premade quadcopter incurs a heavy upfront cost with minimal costs over time while custom drone cost is distributed over a greater period. This is partially due to continual upgrades, needing to replace parts or a change in scope. There's always the possibility as well that unforeseen circumstances come up and new sensors are needed to achieve the original mission. One example of this in the branch bot project was the addition of a barometer. Initially, we thought that the flight controller did not need a built in altitude sensor to perform its mission. This is primarily due to there being a camera that could theoretically determine the height off the ground using computer vision. However, this was not the case and we ended up having to buy a separate component and integrate it with the flight controller. This ended up costing our team roughly $10 and three to four days' worth of time. This type of issue would not necessarily come up with a pre-bought product however there is a premium in the upfront cost. The DJI Mavic Pro 2 was released in stores in 2016 and has been discontinued. The initial cost of the drone was $1500 which incorporates the material cost, the research cost and the

package cost of a solid cohesive unit that is reliable (DJI, 2018). Compare this with BranchBot which costs roughly the same amount of money to research and develop and would cost roughly $900 to replicate as is currently (see cost breakdown in Table 5). This $900 would be spent for a less generically good quadcopter however it has the solution to a very specific problem that needs to be addressed. This cost increase is primarily due to the increase in research and the iterations of failed designs and the corresponding cost to each of those iterations. Although these costs are not necessarily reflected in the final cost of the quadcopter there is a total cost that it is reflected in and it costs roughly the same as the DI and Mavic Pro 2

Table 7: Cost breakdown of BranchBot

Item	Amount	Cost per Item	Cost
SD card	1	$15.00	$15.00
PS4 camera adapter	2	$12.00	$24.00
FeeTech FB90 Servo	1	$9.00	$9.00
Calipers	1	$24.00	$24.00
Frame	1	$140.00	$140.00
Motors	4	$30.00	$120.00
Flight controller	1	$89.00	$89.00
Propellers	2	$4.50	$9.00
Main battery	1	$117.00	$117.00
Raspberry Pi battery	1	$25.00	$25.00
USB 3.1	1	$7.00	$7.00
12 gauge wire	1	$4.25	$4.25
3D printing (arm)	50	$1.00	$50.00
Raspberry pi battery	1	$18.00	$18.00
3d printing (Quad)	58	$1.00	$58.00
Bluetooth adapter	2	$10.00	$20.00
Mcaster-carr	1	$65.00	$65.00
Fc v2	1	$70	$70.00
XT60 to JST	1	$9.00	$9.00

adapter			
Converter Step Down	1	$12.00	$12.00
12 gauge wire	1	$5.00	$5.00
GPS Module	1	$20.00	$20.00
Remote controller	1	$50.00	$50.00
Barometer	1	$10.00	$10.00
lipo battery charger	1	$7.00	$7.00
More propellers	3	$7.20	$21.60
Replacement propellers	1	$15.00	$15.00
Replacement propellers	1	$15.00	$15.00
USB 3.1 1.5 ft	1	$8.00	$8.00
Mcmaster	1	$43.00	$43.00
Propellers	2	$12.00	$24.00
Raspberry Pi 4B	1	$75.00	$75.00
Intel Realsense L515	1	$589.00	$589.00
Total			$1574.25

Technical knowledge requirement: How skilled does the operator/customer need to be in order to pilot the robot?

In the end, what matters is the end-user experience. For the Ghana drone, the end user will likely be a worker at a goldmine who does not necessarily have any technical background to tinker with or modify the drone with. This leads to the end application and drone being very user-friendly and straightforward to use. This is so this drone operator can easily navigate and receive all the results that they need. For BranchBot, which is more in the prototype stage, the end-user has a significantly higher technical background as they should already know basic mechanical and software concepts and have applied them in other areas of study. This means that

the quadcopter can be left in a state that needs more fine-tuning and tinkering in order to run. In the end, the branch prop should end up in a similar situation as the Ghana project application however it is significantly earlier in the design cycle of it.

Results

Through my analysis, both projects made the correct choice. This is because the Ghana-based mission only required a quadcopter with a good camera and the ability for steady, smooth flight. A premade video quadcopter allows for this. I would recommend switching to a different premade quadcopter that has more built-in computer vision accessibility. This is primarily because, during the development process of the app, there were significant hurdles to automating this quadcopter that were never quite overcome such as the two seconds between photoshops and the time it takes to download photos to use for computer vision. With the correct premade drone this is doable. For BranchBot no premade drones that either land on tree branches already or are modifiable to be able to do so within our price range. This meant that we had to create a custom drone with the materials we had at hand. When starting with a new environmental robotics project I'd recommend searching for a pre-made drone within your price range and seeing if that can suit your needs. If so and you have the budget I would recommend using it because more time can be spent ironing out the mission. If one does not easily exist or the software versions are incompatible with what software you want to implement on the quadcopter I would recommend creating your quadcopter or at least starting from a kit. A kit allows for some degree of standardization and framework to allow for a jump start in the development process. It will allow you then to switch out your flight controller or add customer electronics where needed to complete the mission.
