Worcester Polytechnic Institute

Robotics Engineering Program

Autonomous Drone Pollination:

SUBMITTED BY:

Connor Craigie
Peter Guglielmino
Zeynep Şeker
Nathan Stallings

PROJECT ADVISORS:

Dr. Carlo Pinciroli (RBE/CS)
Dr. Greg Lewin (RBE)
Dr. Çağdaş Önal (RBE/ME)

Date Submitted : May 6th, 2021
Date Completed: May 6th, 2021

Table of Contents

1. Introduction

As the demand for agricultural products continues to grow with the global population, the need for pollinators increases. Of the 200,000 pollinating species, a diverse group of 20,000 bee species has the largest impact (*Jaramillo, 2018*). According to the Food and Agriculture Organization, it is estimated that 100 crops provide 90% of the food supplies for 146 countries. Of these 100 crops, 71 of these crops are bee-pollinated *(Kluser, 2007).* Along with pollinating essential crops, bees pollinate many types of wild plant species, supporting various global ecosystems.

Although there are many different pollinators, none are as successful as bees. The unique ecology of bees contributes to their overwhelming success as pollinators. Their complex societies containing 20,000 to 70,000 bees are divided into multiple roles allowing them to pollinate a variety of plant species throughout an area. Individual bees can communicate with other members of the hive using a dance to recruit other worker bees, increasing foraging effectiveness. Furthermore, the anatomy of a bee gives it great flexibility in flight and the ability to carry substantial loads *(Aslan, 2016)*. These characteristics give bees an advantage as pollinators over other insect species.

Unfortunately, over the past few years, there has been a growing concern about the global decrease in pollinators, especially bees. With increases in pesticide use, parasites, and other diseases, the declining bee population became a concern in 2006 with the onset of Colony Collapse Disorder (CCD), which caused a significant decrease in colonies. CCD occurs when the majority of the worker bees disappear leaving the queen alone. Because of CCD, US beekeepers have lost approximately 30% of their bee colonies *(Kaplan, 2021).* The decline of bee

populations coupled with the growing need for pollinators has motivated people to look for solutions to address this pollinator shortage.

To prevent bees from dying to parasites, solutions using genetic modifications are being investigated. Researchers are developing solutions that focus on protecting the bees from environmental threats. Rinderer *et al.* describe breeding programs from the University of Minnesota and the Russian Honey Bee breeders association that have genetically modified bees to make them resistant to the Varroa destructor parasite *(Rinderer, 2010)*. While this solution may protect bees from the Varroa destructor in the long run, it does not protect them from other predators. With globalization, many non-native species are brought to new countries, threatening the local species. One aggressive apex predator of honey bees is the Vespa mandarinia, commonly known as the "murder hornet". This species recently arrived in America and poses a new threat to the already declining bee population. The only solution for a predator like this is to eradicate them before they assert themselves as a prevalent species *(Tripodi, 2020)*. Another major threat to the bee population and the leading cause of CCD is Neonicotinoids, the most widely used group of insecticides. These chemicals are sprayed on plants and are absorbed into their pollen and nectar. Bees that ingest this polluted nectar become paralyzed and die. To mitigate the environmental damage of the insecticides, many governments around the world have limited the use of these and other pesticides, but this has not been enough to prevent the bee population from declining *(Hopwood et al. 2016)*.

Another solution to the declining bee population is to use robotics to assist the bees in pollination. While this solution does not address the declining bee population, it does help meet the pollination needs. Pollinating with robots is not a trivial task: a pollination robot will need sensors that observe its environment and detect flowers, and a means to pollinate these flowers

such as an end-effector or fan which transfers the pollen from flower to flower. Additionally, robots will need to be deployed at a large scale to efficiently supplement declining bee populations. Watson and his team from West Virginia University created *BrambleBee*, a wheeled ground robot equipped with a robotic arm and a pollinating end-effector. *BrambleBee* uses mapping and localization techniques to navigate around a greenhouse, in which it identifies and pollinates bramble plants *(Watson, 2018)*. While *BrambleBee* can replace the pollination that would be provided by a bee colony in this controlled environment, *BrambleBee* is limited to greenhouse environments as its current design cannot traverse rough and uneven terrain.

Aerial robots, such as drones, do not struggle with the problem of traversing rough terrain and can be used as another artificial pollinator. Two main ways of pollinating flowers with drones have been explored: wind-based and contact-based. Lee Kwan-hee *et al.* utilize the downwash effect of a drone to blow pollen in the air stimulating self-pollinating plants (*Arirang News, 2019*), while Chechetka *et al.* use direct drone-to-flower contact to collect pollen similar to animal-based pollination (*Chechetka, 2017).* While the previous systems are capable of pollination, they do not act autonomously, which limits the scalability of the solution. Furthermore, dispersing the pollen using air circulation is not a reliable way to pollinate all flowers in a region, and the drone designed by Checketka and his team required multiple passes at a flower to collect pollen. Because of the inconsistency in these pollination approaches, neither of these proposed systems can be implemented at a scale that can supplement the declining bee population in their current form.

The work discussed in this paper is our contribution to the problem of the declining bee population by supplementing it with artificial pollination utilizing a UAV. By creating an autonomous drone with onboard vision processing and an actuated end-effector, our proposed

solution allows for an unmanned drone to accurately transfer pollen between flowers. This solves the problem of inconsistent pollination and lack of automation that the previous pollinating drone systems experienced.

1.1 Problem Statement

To achieve effective autonomous pollination, a robot must be able to traverse and navigate its environment. The robotic system needs to detect its surroundings, allowing it to search for flowers within a region while avoiding obstacles. This system must be able to transfer pollen between flowers without damaging the flowers it pollinates. The robot should also avoid revisiting flowers that have already been pollinated.

To contribute to autonomous pollination, we designed a drone capable of autonomously navigating a region, detecting and targeting flowers to pollinate and transfer pollen between targeted flowers. This drone can localize itself relative to a home position and fly within a designated area using its onboard flight controller. Our system includes an onboard camera used for image processing to locate flowers within its field of view. The drone's camera is fixed to an actuated end-effector which is used for collecting pollen from one flower and transferring it to another.

1.2 Contribution

One contribution of our work is the development of a software architecture that can be deployed on a drone to perform autonomous pollination. Using a Pixhawk flight controller, we developed a ROS architecture that searches a region for flowers through a cyclical search pattern. While the drone is searching the environment, an OpenMV camera performs a flower

detection routine to locate and target detected sunflowers. A velocity PD control loop is used to center the drone over a detected flower as the drone lowers to attempt pollination.

Another contribution of our work is the development of a small drone model and end-effector design that can be implemented in future works. The proposed drone model is a 3D-printed, modular-based drone capable of supporting all required hardware for this task. The drone's minimized body profile is designed to lower the impact of downwash and allow for maneuverability around flowers. We discuss the creation of a servo-powered end-effector designed to house a pollen collection apparatus and OpenMV camera.

2. Related Works

Due to the decline in the bee population, people have started seeking alternatives to pollinate crops artificially. Some of these methods involve human labor while some rely purely on automated systems such as UAVs and on-ground robots. Although hand pollination by humans is possible, it is not a viable solution because it requires extra human labor and is not cost-efficient for large-scale crop production. Furthermore, manual pollination of flowers may lead to dangerous working environments and serious injuries. For example: to pollinate flowers on top of trees, workers either need to climb the tree or use a ladder or an automated lift, which increases the risk of work injuries. Although hand pollination is cost-efficient for a small-scale task such as gardening, it is not a feasible solution to replace the endangered bee population *(Darnaud, 2016)*. Due to the inefficiencies of hand pollination, engineers are researching alternative robotic pollination systems.

2.1 Pollinating Systems

Watson *et al.* developed a ground-based autonomous robotic pollination system consisting of a four-wheeled vehicle with a robotic arm and end-effector. This robot operates within a greenhouse environment and is equipped with multiple sensors, including 3D LiDAR, GPS, and multiple cameras. These sensors are used to localize the robot within its environment using a simultaneous localization and mapping algorithm, and the cameras help identify bramble flowers. The robotic arm, equipped with a custom-made end effector mimicking the behavior of a bee's pollen-collecting scopa, uses a path planning algorithm to accurately pollinate the flowers

(Watson, 2018). The project shows that a robotic system capable of accurately detecting and pollinating flowers in a controlled environment is feasible.

Lee Kwan-hee *et al.* developed a pollination drone (*Arirang News, 2019*). They used an off-the-shelf drone and performed teleoperated flights across plants growing inside greenhouses. These flights are all performed near plants whose flowers hold both sets of reproductive organs. The system relies on the downwash draft of the drone to stir the air and cause natural self-pollination. In its current form, the system is mainly being used for research purposes to study the most optimal altitude and flight speed for plant self-pollination *(Arirang News, 2019).* While this approach works for plants that contain both reproductive organs, it is not a viable solution for all flowering species.

Chechetka *et al.* designed and synthesized ionic liquid gels (ILGs) for artificial pollination. This adhesive material can effectively collect pollen grains and successfully pollinate *L.japonicum* flowers. This is achieved by attaching animal hairs to the bottom of a small, teleoperated UAV. This animal hair is coated with the ILG and flown up against flowers. Multiple passes were performed over the flowers to complete the pollination cycle *(Chechetka, 2017)*. Although the project's main goal is the development of ILGs, Chechetka *et al.* are the only group we identified that uses a drone that directly contacts a flower.

2.2 Drone Localization

Proper localization is imperative to autonomous navigation of any kind. For many applications that do not require precise movement, Global Positioning System (GPS) or Global Navigation Satellite System (GNSS) readings are adequate to position a system in proximity to a given target. Both GPS and GNSS readings commonly reach coordinate accuracy of meter-scale

(Chen, 2020). Our robot will require near-centimeter flight precision, and therefore GPS/GNSS on its own will not be accurate enough. Furthermore, because our drone requires relative position for localization within a field, the absolute position provided by the GPS is not usable on its own. Chen *et al.* identified other solutions for drone localization when surveying around high voltage electrical equipment. Due to the intense electrical current close to the drone, GPS and GNSS signals were not accurate enough for proper localization. Chen *et al.* focused on vision processing to identify a fixed image place on the surface. A well-defined checkerboard pattern is used as a fixed image. Their system uses the predefined properties of the pattern to identify its position and orientation in the local coordinate system of the drone. By using a pan/tilt camera the system can keep constant vision of the target at all times. The checkerboard's Cartesian position and rotation information are transformed into the drone's coordinate system through a set of simple homogeneous transformations which account for the camera's pan and tilt. This strategy allows for the robot to constantly calculate its relative position to the image on the surface *(Chen, 2020)*. The paper provides a relevant example of drone localization using real-time image processing which is more accurate than GPS or GNSS localization.

An additional problem with GPS localization is that it becomes inaccurate when the system is indoors. To accommodate indoor systems such as greenhouses, indoor localization is a crucial part of a robust pollinating drone. Chong Shen *et al.* have looked at other options for indoor localization(*Chong Shen*, 2016). They have tested sensors that track the external environment in addition to GPS satellite data. Their study implements a sensor fusion of GPS, Inertial Navigation System (INS), and optical flow. To compensate for unreliable GPS signals, Chong Shen *et al.* have implemented an extended Kalman filter that incorporates accelerometer data, gyroscopic data, optical flow velocity estimation, and sonar data measuring the linear

distance to the ground. This extended Kalman filter was designed to output an estimation of the system's Cartesian position, as well as its velocity vectors along the X, Y, and Z axis *(Chong Shen, 2016)*. This paper provides an example of using a Kalman filter to support indoor GPS signals with other localization sensors. These papers outline the fact that GPS will not be good enough on its own to provide the positioning data for the pollination system. However, even though GPS is not useful for precise positioning, it will be useful for region searching, as it can provide continuous position while visual feedback from the drone cameras will be able to produce relative position targets for the pollination system.

2.3 Drone End-Effectors

Attaching an end effector to a drone is not a novel concept. Kutia *et al.* developed, simulated, and built a proof-of-concept drone capable of collecting above-canopy leaf samples in forests *(Kutia, 2015)*. This approach consists of a drone whose end-effector interacts with plants in an outdoor environment. The team's original design consists of a single degree-of-freedom manipulator. Their preliminary testing found that this design is not suitable for collecting canopy samples for two reasons. First of all, the canopy would prevent the drone from getting close enough to the target sample without hitting the surrounding branches. The second restriction is the downwash produced by the rotors. These problems are solved with a 2 DOF manipulator allowing the drone to interact with the environment away from its body, negating the downwash and size constraints of the drone. They also found that the 2 DOF arm could compensate for unexpected movements of the branches or the drone itself due to external factors such as wind *(Kutia, 2015)*. The main contribution of this paper is to introduce a drone with an end-effector

that interacts with plants in an outdoor setting. Kutia *et al.* highlight the challenges they encountered and discuss the process they took to overcome these challenges.

A problem for aerial manipulator systems is controlling the UAV while minimizing the disturbances to the environment, such as strong gusts of wind. Zhang *et al.* developed a control system for a hex-rotor UAV and a 7 DOF robotic manipulator allowing for consistent performance at grasping tasks in high wind environments (*Zhang, 2019*). The team concluded that designing one controller for the hex-rotor and a separate controller for the manipulator would provide the best outcome. The hex-rotor's controller is composed of an H∞ controller and an acceleration feedback enhanced term. The H∞ controller ensures the stability of the UAV, while the acceleration feedback is used to compensate for wind disturbances. The manipulator uses a PID controller which adjusts the position and velocity of the end-effector. Zhang *et al.* found that these controllers improved the hex-rotor stability and the positional control accuracy of an aerial manipulator allowing tasks to be completed in high wind environments (*Zhang, 2019*). The paper exemplifies that designing two independent controllers is a feasible approach and helps maximize the system's performance in high wind environments.

Certain end-effectors may not be suitable for all the desired tasks of a drone, or the end-effector may get damaged, thereby putting the drone out of commission, Hricko *et al.* designed a device to efficiently and autonomously change between desired end-effectors (*Hricko*, 2018). Their desire for having a drone capable of supporting various sensory equipment for specific tasks led to the development of a modular end-effector design that can be changed to support different tasks. To minimize the weight of the end-effector, their design uses the main power source of the UAV to power any sensors or tools present on the attached end-effector. Their end-effector gripper design consists of an elliptical shape, enabling the gripper to grab the

end-effector module despite minor misalignments *(Hricko, 2018)*. The main contribution of the paper is to provide an example of a drone design that utilized a module focused end-effector. An easily replaceable end-effector allows for fast repairs and a method for quickly testing multiple end-effector designs.

2.4 Drone Vision

The ability to recognize individual flowers can improve pollination by accurately identifying targets for the pollination apparatus. Oppenheim *et al.* developed a drone with a vision algorithm capable of differentiating flowers from the background image *(Oppenheim, 2017)*. One method the team used is local-based techniques to detect flowers in a field, where each pixel is examined to determine whether it belongs to the target flower or the background by comparing their color properties with the color properties of the pixel's neighbors. The method converts the image to the Hue-Saturation-Value (HSV) color space, which differentiates the colors in the image. Unlike RGB, which is defined by how much red, green, and blue is in a single color, HSV differentiates between a color and the color's intensity, which allows for the detection of a desired color in various lighting conditions. The HSV value of each pixel is compared to its neighbors, identifying whether that pixel belongs to a cluster representing the target flower or the background. This also eliminates the noise of fluctuating shades due to changing daylight. The algorithm applies masks to the image and detects multiple flowers at once *(Oppenheim, 2017)*. Similarly, Dias *et al.* used deep learning to detect apple flowers in a grove *(Dias, 2018)*. Both teams used the HSV color model to disassociate brightness. By converting a colored image into an HSV file, the team binarizes the pixel HSV values and identifies clusters that have similar properties. By comparing the properties and the size of these

clusters, the team can associate whether the cluster belongs to the target flower *(Dias, 2018)*. These papers introduce techniques of flower recognition that adapt to changing lighting environments and emphasize the importance of using a color scheme that can identify a color in various lighting conditions.

2.5 Novelty

The novelty of the proposed autonomous pollination system discussed in this paper comes from the integration of previously developed functionalities into one unified solution. Techniques from computer vision, controls, mechanical design, trajectory generation, and localization are all involved in developing a robust autonomous pollination system. This problem also addresses the challenge of developing robots that interact with the natural world, and not a controlled laboratory setting.

3. Methodology

The goal of this section is to outline our team's requirements and design choices along with describing our development process. This chapter is broken into two sections, the first section outlines the problem formulation. The second section explains the system design aspects and associates them with our problem requirements.

3.1 Requirements and Problem Formulation

Because mimicking the behavior of a bee is a highly complex task, we simplified the problem down to designing a system capable of pollinating a flower rather than mimicking all aspects of a bee. The goal of this project is to create an autonomous system capable of locating flowers within a region, and transferring pollen between them. To accomplish this, we identified four major functions:

1. The system must be able to navigate a region near its starting location and search for flowers during flight.
2. The system must be able to locate and approach flowers.
3. The system must have a method of sequencing between multiple target flowers in the same region.
4. The system must have a method for collecting and depositing pollen.

We have also defined five metrics to evaluate our system: flower detection accuracy, pollination success rate, flower survival rate, attempted flower pollinations per cycle, and pollinatable flower species. We designed an experiment in *Section 4.2* to evaluate these metrics. We attempted to match 90% of a bee's capabilities in flower detection accuracy, pollination

success rate, and flower survival rate; however, we have excluded pollination attempts per cycle and pollinatable flower species, because matching 90% of a bees performance in these metrics is outside the scope of this project. By keeping the targeted number of types of flowers to one, and the number of pollination attempts during each flight to two, we can focus on the parameters which are more relevant to our defined problem. Although these numbers are much lower than that of a bee, the system will still resemble the actions of one. We decided to target a single species of flower because bees are flower loyal and pick a single species of flower to pollinate until the blooming season has ended *(Goodwin*, *2012)*. While bees can visit over 100 flowers per flight, we target only two flowers per pollination cycle because that is the minimum number of flowers needed for a flower to be successfully cross-pollinated *(Lihoreau, 2016)* The metrics and targets are listed in *Table 1* below:

Metric	**#**	**Measurement method**	**Bee's ability**	**Target**
Flower detection rate	1	Compare the number of flowers in the search region to the number of flowers detected by the system (%)	~100%	90%
Pollination success rate	2	Compare the number of flowers that the system attempted to pollinate to the number of them that are pollinated (%)	~100%	90%
Flower survival rate	3	Compare the number of artificial flowers that the system attempted to pollinate to the number of them that are not damaged (%)	~100%	90%
Pollination attempts per cycle	4	Count the number of flowers the system attempted to pollinate in a cycle (#)	100	2
Pollinatable flower species	5	Count the number of species the system is able to pollinate (#)	1,000+	1

Table 1: System Success Metrics

3.2 Design

The system described in *Section 3.1* requires the integration of multiple subsystems. While there a variety of designs can meet the outlined system requirements, we decided to design a drone with an end-effector and onboard camera to complete the task of autonomously pollinating sunflowers. We chose to pollinate sunflowers because of their consistent color patterns, their large size relative to other flowers, and a minimal amount of obstructive petals.

To focus on the implementation and integration of the subsystems, a QWinOut DIY F450 Drone Quadcopter Kit with a Pixhawk Flight Controller is used as a development platform. This drone was chosen because of its payload size of over a kilogram, giving sufficient design space to add onboard computers and an end effector. This drone also came with carbon fiber propellers and stall-resistant motors that are durable enough to withstand potential crashes, which will reduce the need for replacement parts. While the pollination system was being evaluated on the kit drone, a smaller drone was designed that implements all of the desired functionality while minimizing size and cost. A smaller drone would increase scalability because it would be cheaper to produce and easier to deploy in complex environments. The overall design of the drone relies on a Pixhawk 2.4.8 flight controller to control and stabilize the drone, a Raspberry Pi 3B running ROS Noetic to manage higher-level decisions and communicate between the different parts of the drone system, an OpenMV camera to detect flowers, a servo-driven end-effector to make contact with the flowers to pollinate them, and a GPS for position estimation.

3.2.1 End-Effector Design

The end-effector refers to the dynamic platform that houses a camera and a pollen collector. For this design, the system uses a makeup brush as the pollen collector, specifically an eyeshadow brush. The camera is placed with the optical axis parallel to the collector. The positioning of the camera and the collection device is depicted in *Figure 1*. This gives the camera a fixed frame aligned with the end effector. Having the camera's line of sight and the brush parallel to each other simplifies the drone's path computation since the offset between the camera and the brush is constant. This way no additional calculation is required to transform the position of the flower detected by the camera to the location of the brush. Since the only position difference between the camera and the tip of the brush is the length of the brush extended from the end effector, it can be assumed that the distance between the camera and the tip of the makeup brush will be constant.

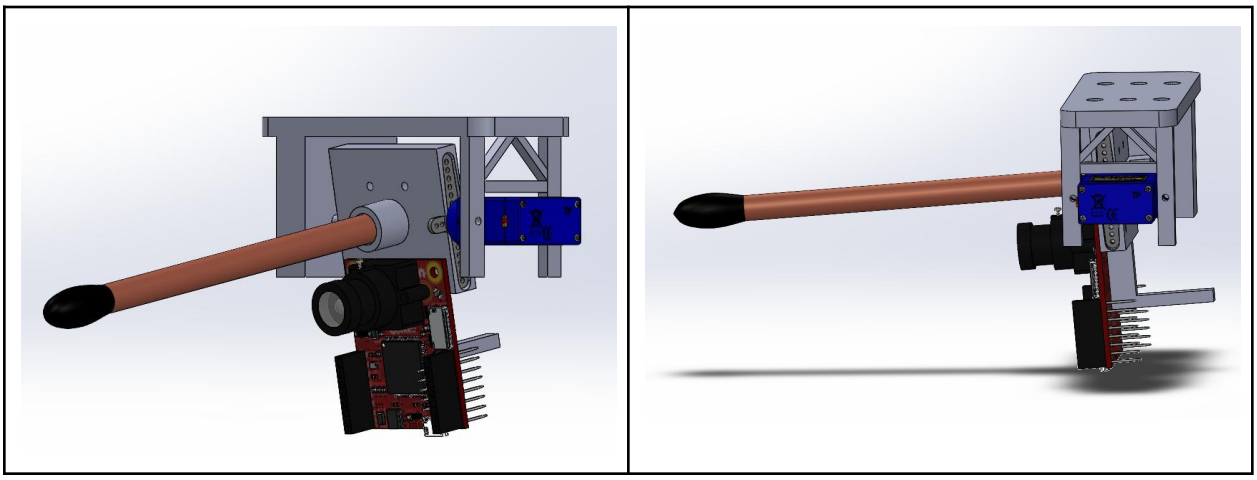

Figure 1: End Effector Design

Our designed end-effector is attached to the drone with four M4 bolts. Although the bolts act as a fixed joint between the drone and the end-effector, the end-effector itself is not

stationary. The plate that houses all of the subcomponents is actuated by a 9g servo as depicted in *Figure 1*. To maintain a constant horizontal position during flight, an additional hard stop is added to the back of the rotating plate, allowing the assembly to compensate for servo error. To avoid a constant gravitational moment on the servo, the center of rotation for the servo horn is collinear with the center of mass for the assembly. This allows the assembly to rotate without changing the drone's center of mass, which would change the dynamics of the drone.

3.2.2 Software Development

3.2.2.1 Software System Layout

The software system is organized into multiple subsections: end-effector control, vision processing, flight control, and optical flow/altitude detection. These subsections all interact with the primary onboard computer, the Raspberry Pi 3B.

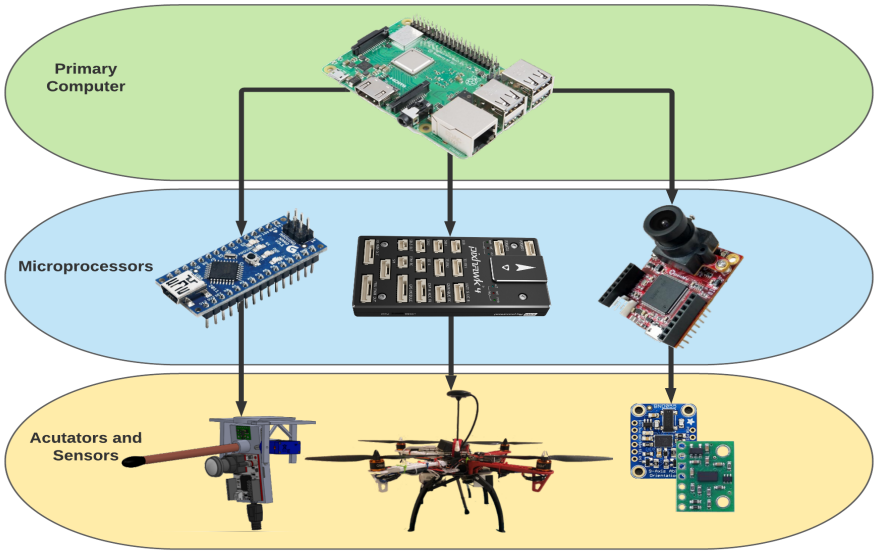

Figure 2: Software System Layout

End effector positioning is handled by an Arduino Nano. This microprocessor communicates to a Python node within our primary computer's ROS architecture. Target end

effector angles are transmitted over a USB cable using UART communication and received directly by the Arduino, which commands a servo to move the end effector.

Onboard vision processing is delegated to an OpenMV camera. This camera runs a custom flower detection algorithm and returns the centroids of detected flowers. The Raspberry Pi communicates directly with the OpenMV camera through UART protocol communicating over the Raspberry Pi's dedicated RX and TX GPIO pins. These pins are accessed similarly to a standard serial bus port. This allows for lightweight signal cables to carry data transmission to the camera, lessening the physical load that would be placed on the servo actuator by a full USB cable.

The Raspberry Pi communicates to the Pixhawk flight controller by sending MAVLink commands through a USB to UART adapter. The generated MAVLink commands originate from a MAVROS node which converts the high-level ROS messages to the MAVLink control messages that the Pixhawk understands. This MAVROS bridge is used to receive positional data from the Pixhawk which is used to generate the target position and velocity setpoints.

The final subsection is handled by a second OpenMV camera. This separate camera performs optical flow to calculate a body velocity estimate. The integrated microchip on the OpenMV camera communicates with a time of flight sensor and IMU using I2C which is used to estimate flight altitude and attitude. The velocity, altitude, and attitude estimates are sent directly with the Pixhawk flight controller through a UART connection.

3.2.2.2 Flight Control System

The drone uses a Pixhawk 2.4.8 flight controller to control the position, heading, and velocity of the drone as it flies around a given region looking for flowers. This flight controller is small, has a variety of interface ports, and is familiar to the design team. The Pixhawk is loaded

with PX4 flight controller firmware which has a variety of useful features, such as data logging, an Extended Kalman Filter to estimate the drone's position using a variety of sensors, and a MAVLink interface which is used to control the drone from an onboard computer.

In order to control the high-level decision-making of the drone, we use a Raspberry Pi 3B running ROS Noetic. ROS Noetic is the most up-to-date version of ROS and gives future teams the flexibility in the event there are ROS2 packages that they would need to use. The Raspberry Pi has multiple UART ports that can be used to interface with all of the desired peripherals we need for our pollination system, specifically the end effector and the Pixhawk flight controller.

To ensure that multiple nodes are not sending different commands to the Pixhawk, a PixhawkInterface node was created in ROS allowing the high-level ROS nodes to publish Pixhawk target setpoints and velocities. The Pixhawk interface takes in a custom message which contains a target setpoint or velocity, as well as other flight commands such as arming or initializing offboard mode. Because all communication to the Pixhawk is done through this node, if two nodes try to send competing position or velocity setpoints, the interface node will be able to control priority preventing conflicting messages.

3.2.2.3 State Machine Functionality

The main node in our ROS architecture implements the system's overall state machine. We designed the workflow of the state machine to resemble the target objectives of our experimental setup. The target objectives are shown in *Figure 3*.

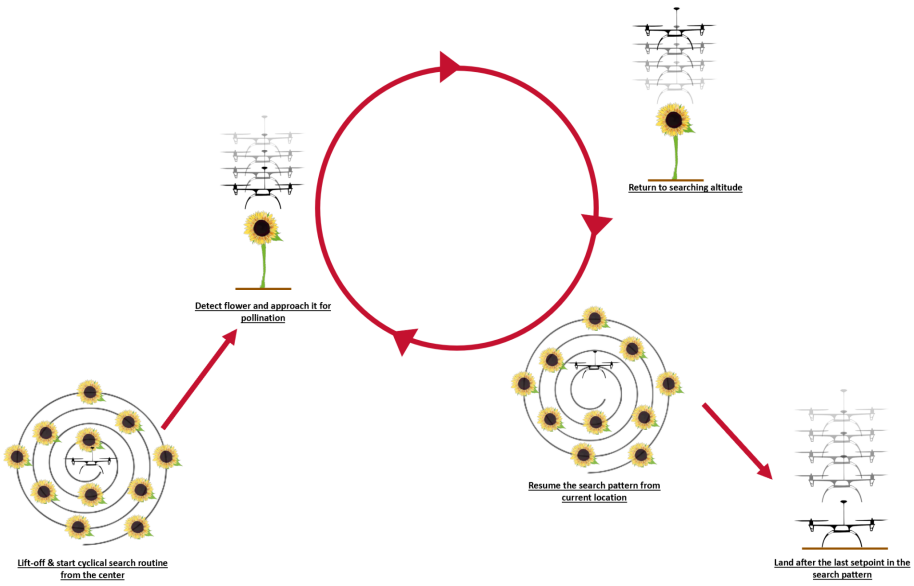

Figure 3: Visual Depiction of State Objectives

Upon power-up, the drone enters an initialization state in which it waits for the GPS signal required to begin in-flight localization. Because of IMU sensor drift that accumulates while waiting for a GPS signal, when the drone successfully gets a GPS signal, its local pose estimate can be offset from the startup origin by a few meters. To adjust for this, we record the drone's current pose and treat it as the cartesian origin before beginning the search pattern. Once this origin pose is recorded, the system sends a setpoint command of one meter above this new origin. Once these setpoints are received by the flight computer, the system is ready to enter offboard mode where it will begin autonomous flight.

Once the drone is properly initialized, the main node enters the search pattern generation state. This state sends a command to the SearchRegion node which generates a list of desired target setpoints that the drone follows when searching for flowers. After the waypoints are generated, the state machine idles until the drone is put into offboard mode by a user hitting the

offboard switch on a radio controller. Once the drone is signaled to enter offboard mode, the search setpoints are iterated through at a fixed frequency.

As the drone searches, the onboard vision system is constantly querying for flowers. If a flower is detected, the drone interrupts the search routine and enters the stabilization state. Here the drone switches from using positional setpoints and begins to use a local body velocity controller to center over the detected flower. Once the system has stabilized and centered over the flower, the drone descends until it reaches the set altitude of the flower. Pollination is considered to be complete when the flower's altitude is met. Once the system completes pollination, it stops using the local body velocity controller and resumes its search pattern. For a short time after pollination, the camera stops searching for flowers. This prevents the system from continuously locking on to the same flower. This cycle of searching, detecting, centering, and lowering continues until the system iterates through all of the desired setpoints. Landing is performed autonomously by the system once the search pattern is complete, or when the user intervenes and performs a manual landing.

3.2.2.4 Search Region Waypoint Generation

To successfully search a region, a list of setpoints needs to be generated that fills the target area. The target search area for the drone is defined as a five-meter radius circle. To search this region, the team implemented the function defined in *Eq 1* which will produce a spiral pattern within the target radius.

$$R(\theta) = \frac{a\theta}{2\pi} : \{0 < \theta < r_{max} \frac{2\pi}{a}\}$$

Eq 1: Polar Search Function

To parameterize elements of the search pattern such as spiral density and maximum search radius, *Eq 1* defines spiral search density constant *a*, which represents the linear distance

between each consecutive path of the spiral. Additionally, the maximum search radius, r_{max} is used to define the boundary of the continuous function. These equations generate the shape of the target search pattern, as visualized in *Figure 4*, but need to be transformed into discrete cartesian points to be used by the drone.

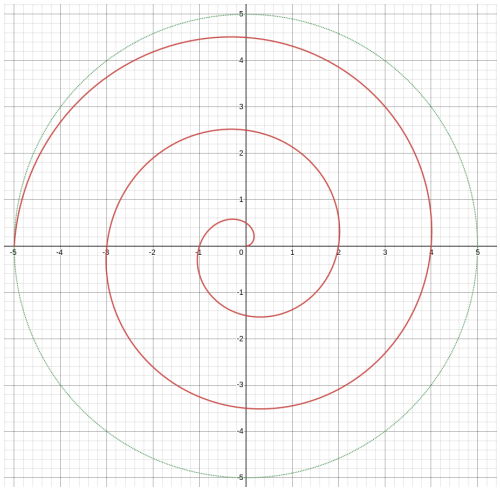	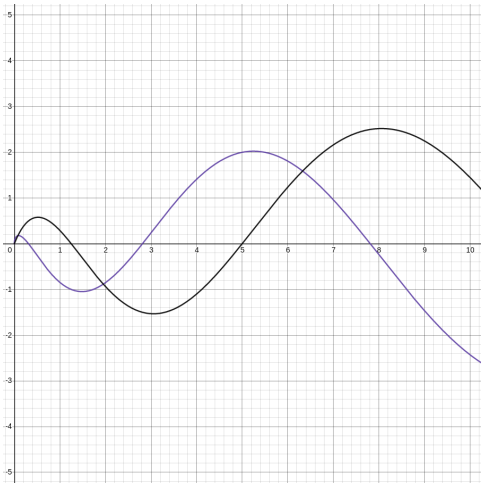
Positional output of polar function	Positional displacements of X(purple) and Y(black) over time.

Figure 4: Search Function Output

$$f_{xt}(t) = \frac{a \cdot c\left(\sqrt{\frac{\left(v_{max} \cdot t \cdot 2\pi\right)}{a}}\right)}{2\pi} \cdot \cos\left(c\sqrt{\frac{\left(v_{max} \cdot t \cdot 2\pi\right)}{a}}\right)$$

Eq 2: X Displacement per Time

$$f_{yt}(t) = \frac{a \cdot c\left(\sqrt{\frac{\left(v_{max} \cdot t \cdot 2\pi\right)}{a}}\right)}{2\pi} \cdot \sin\left(c\sqrt{\frac{\left(v_{max} \cdot t \cdot 2\pi\right)}{a}}\right)$$

Eq 3: Y Displacement per Time

By redefining the functions in the continuous time domain, as seen in *Eq 2* and *3*, we can add two additional parameters: c, an experimentally derived constant; and v_{max}, the system's target velocity.

The final implementation of this search pattern generates a list of discrete set points. When published at the desired frequency, these setpoints will result in the drone traveling at the desired search velocity. To search a given space, the drone iterates through this finite list of positional setpoints. Because the drone's path is determined by these setpoints, the system can easily exit and enter the search state during a pollination attempt.

3.2.3 Flower Detection and Targeting

While the drone navigates the environment, a crucial aspect of pollination is the detection and targeting of flowers. This section discusses the steps taken and the tools used to develop software capable of locating and centering over target flowers.

3.2.3.1 Flower Detection Algorithm

To autonomously pollinate flowers, our system needs to locate and target flowers. While various cameras could be used to accomplish this, we examined two possibilities, the Pixy2 and the OpenMV Cam H7. We chose these cameras because they have integrated vision processing, which distributes the computational load allowing for a less expensive onboard computer. This integrated vision processing allows the cameras to detect and track specified color blobs, which are groups of pixels with similar colors. We determined these cameras could be used to identify flowers by testing their ability to recognize artificial sunflowers. We opted to use the OpenMV camera for our drone because it provides better documentation and is easier to set up with the Raspberry Pi. The OpenMV camera also provides a large number of helpful functions and

camera settings that make detecting flower objects simpler than on the Pixy2. The camera and the Raspberry Pi communicate via a UART connection. We opted for UART because we did not need more than one master or slave, and there is no need to occupy multiple pins or set up a clock signal, as required by other communication protocols. While the HSV color space has been used in previous work, we designed an algorithm that uses the LAB colors space to detect flowers because HSV was not supported by our model of the OpenMV camera. Similar to HSV, LAB can identify a color in various lighting conditions as it focuses on the color's lightness (L), the red and green values (A), and the blue and yellow values (B). To ensure consistent color measurements throughout the day, we set specific color configuration values in the camera's register. The OpenMV desktop application has a threshold editor tool that defines a threshold for the color blob function to identify the yellow color of a sunflower. Once a yellow blob is found by the camera, the algorithm examines the interior of that colored blob. If the interior of the detected blob is not yellow, the algorithm classifies this instance of yellow as a flower. Data containing the flower's location, orientation, and size is then sent over the UART communication to the Raspberry Pi. Centroid-based object tracking code allows the camera to differentiate between multiple flowers and track them over a series of frames. This would be used to differentiate between two flowers located close to each other in the environment, but due to time limitations, this code was never implemented into the search routine.

3.2.3.2 Flower Stabilization

Once the camera detects a flower, the drone breaks free from the search routine and enters the flower stabilization state. This state implements a PD velocity controller that sends a velocity message to the Pixhawk in the East-North-Up (ENU) frame. Because MAVROS sends velocity setpoints in the ENU frame, our system transforms target velocities from the local body

frame to the global ENU frame. To do this, we rotate the local body velocity vector by the current heading of the drone's compass. To center over the flower, the controller takes the centroid of the detected flower as input. Since the frame size of the camera is known, the desired center-point can be established. This allows the controller to calculate the error between the flower's current centroid and the desired center-point of the camera. The proportional and derivative terms of the controller are used to calculate the desired velocities based on this error. Once the error is smaller than a defined threshold, the drone begins to lower over the centered flower while running the PD controller.

3.3 Drone Miniaturization

Beyond making an autonomous drone pollinating system, another goal of this project is to make a system that can effectively navigate complex environments and is cost-efficient enough to be deployed at a reasonable scale. Because of this goal, part of the project focuses on creating a miniaturized drone that is designed specifically to achieve the target pollination objectives and would be small enough to minimize potential downwash effects on real flowers.

The designed drone body, as seen in *Figure 5*, supports the necessary hardware discussed in *Section 3.2*. The design attempts to minimize both the distance between the rotors and the bodyweight, which will lower the cost of the drone frame and lower the downwash effect on the flowers. The miniaturized drone is designed by focusing on the development of the central body hardware stack and then building out the propellers and motors. By focusing on a central stack, the design can be easily modified to house the appropriate sensors and hardware, and it can be rearranged to alter parameters of the drone such as the center of gravity.

Because this design method focuses on supporting the central hardware stack, it is not constricted to a predefined profile. This means that the scale of the drone will grow as more functionality is added to the desired hardware stack, which can go against the goals of a small profile drone. However, this design strategy is a more sustainable development technique for the project while still in the prototyping stage.

The prototype drone body is manufactured with 3D printed ABS plastic. ABS was chosen due to its greater toughness and tensile strength when compared to PLA, our secondary option. Because the drone can become damaged during testing, we developed a modular design to avoid remanufacturing the entire drone body in the event of a crash. This design has detachable motor arms and end effector allowing for quick replacement if they are damaged.

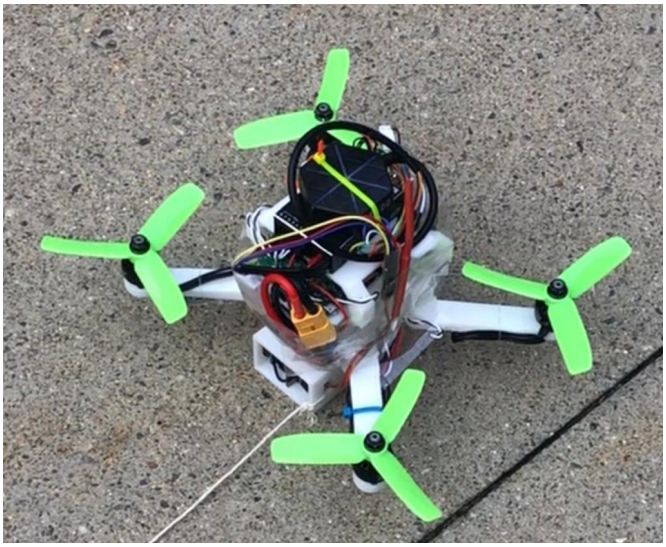

Figure 5: Proposed Drone Body

4. Experimental Evaluation

This section outlines the design of multiple experiments to evaluate our system. This section also includes the data that was collected from our experiment and an analysis of that data.

4.1 Sub-System Validation

This section discusses the experiments designed to test the performance of the various sub-systems developed. The results of these experiments are discussed and supported with collected data.

4.1.1 Search Patterns

To evaluate the effectiveness of the search pattern algorithms we developed for the drone, we ran a series of flight tests with different search patterns and setpoint densities. The two search patterns that were evaluated were a spiral pattern and a rectangular pattern as compared in *Figure 6.1* and *Figure 6.2*.

Testing was done in a field where both search patterns were deployed, one after the other. *Figure 6.1* and *Figure 6.2* show the drone successfully follows the desired setpoints with relative accuracy and both patterns could be used to search for flowers. The team eventually chose to use the spiral search pattern for the drone because the continuous nature of the spiral pattern prevents jerky motion that can result from the right angle turns found in the rectangular search pattern.

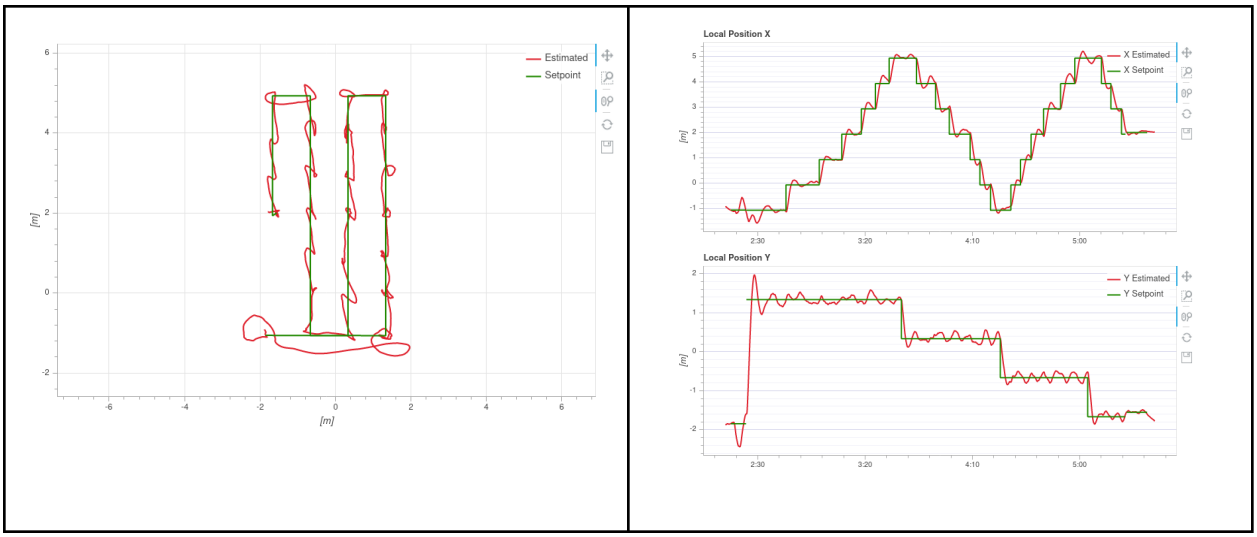

Figure 6.1: Drone Odometry Performing a Rectangular Search Pattern

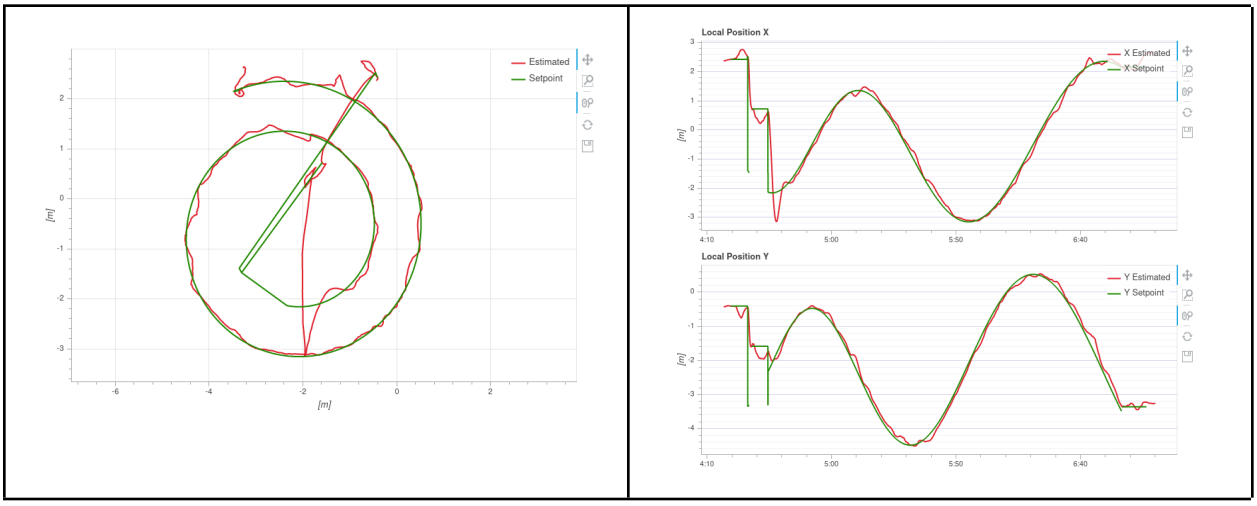

Figure 6.2: Drone Odometry Performing a Spiral Search Pattern

Once the spiral search pattern was chosen, we modified parameters such as spiral density, drone velocity, and setpoint publishing frequency. We found that the drone consistently detected nearby flowers with a spiral density of 1 meter, a target velocity of 0.5 m/s, and a publishing frequency of 5Hz.

4.1.2 End-Effector Validation

The end-effector was designed to collect and deposit pollen and dynamically track flowers as they move within the end-effector's field of view. This section discusses the tests performed to validate the ability of the end-effector to perform these functions.

4.1.2.1 Pollen Collector

To determine the efficacy of pollen collection using a makeup brush, we ran multiple tests. The first unit test evaluates a makeup brush's ability to collect pollen from flowers. We conducted this test by inserting a makeup brush into multiple flowers around Worcester, MA, and observed the amount of pollen collected. The pollen collected by the makeup brush from the flower is shown in *Figure 7*. Once we determined that a makeup brush was a viable pollen collector, we evaluated different types of makeup brushes.

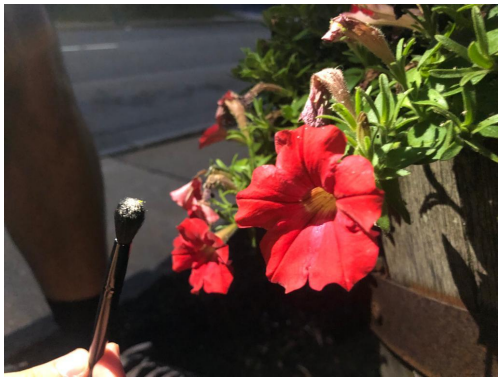

Figure 7: Hand Pollen Collection Tests

When choosing which brush to use, two brushes were tested. As seen in *Figure 8,* brush A has shorter bristles and a wider diameter, while brush B has longer bristles and a smaller diameter. Additionally, brush A has a denser set of bristles compared to brush B. To compare the amount of pollen these brushes can collect and determine the optimal brush for pollination, we replicated the pollination process by testing the brushes with eyeshadow instead of pollen.

Because eyeshadow is easier to visualize due to its distinct color and is more accessible throughout the year, all further tests were conducted using eyeshadow. After loading the cores of two artificial sunflowers with equal amounts of eyeshadow, we contacted brush A with one of the flowers and brush B with the other flower. Because eyeshadows have distinct colors that contrast with the brush bristles, we were able to visually compare the number of eyeshadow particles collected by each brush. During testing, we determined that having a larger area and denser bristles allowed the brush to collect more particles than a soft and narrow crease brush. As seen in *Figure 8* below, brush A has collected more colored powder.

Figure 8: Pollen Collector Comparison

4.1.2.2 End Effector Tracking

We developed a proportional controller for the end-effector to track a flower for pollination. The effectiveness of this flower tracking algorithm was evaluated by placing a flower within the view of the camera and moving it up and down, which tested the end-effector's ability to track the flower as it moved. This testing setup is shown in *Figure 9*. This test found

that the end-effector can successfully track flowers between 0 and 90 degrees before hitting the physical limitations of the end-effector. This test proved that a proportional control could be used to accurately track a flower moving relative to the camera frame to properly align the pollen collector with the center of the flower.

Figure 9: Dynamic End Effector Tracking

4.1.3 Flower Detection

We conducted a unit test to assess the centroid-based object tracking code developed on the OpenMV camera. This test consisted of a stationary flower being inserted into the camera's view. After a few frames, another flower is introduced and moved around to ensure proper object tracking. The object tracker is also capable of remembering objects that leave the camera view for a few frames, and this feature was tested by removing one flower from the camera's view for a few seconds before being re-inserted. The test showed that the desired features for object tracking were successfully implemented. *Figure 10* shows the algorithm differentiating between the stationary object, flower 0, and the moving object, flower 1.

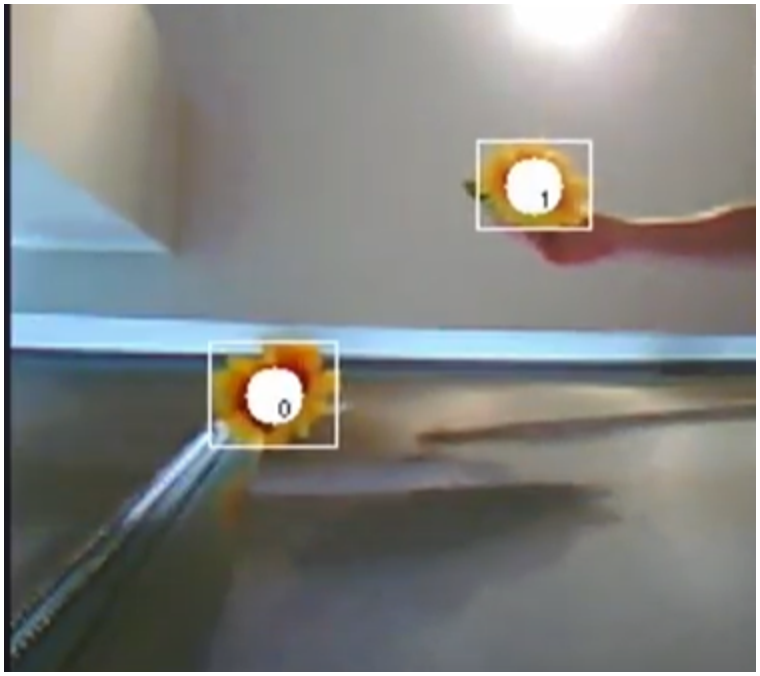

Figure 10: Flower Detection and Identification

4.1.4 Flower Centering

To test the flower centering algorithm, we manually controlled the drone to hover over a flower. Once the camera detects the flower, the drone will be set to offboard mode, and the PD controller will generate target velocities to stabilize the drone over the detected flower. This test was designed to tune the K_p and K_d gains until the controller gives the desired behavior. *Figure 11* shows the flight controller's X and Y setpoint velocities over time. The gains were periodically adjusted in the teleoperated state, and then the drone was switched into the autonomous stabilization state to evaluate the desired PD gains. We observed that the optimal gain values were the ones tested in *Figure 11.1*. The controller was able to center and stabilize the drone over a flower in wind speeds up to 20 m/s.

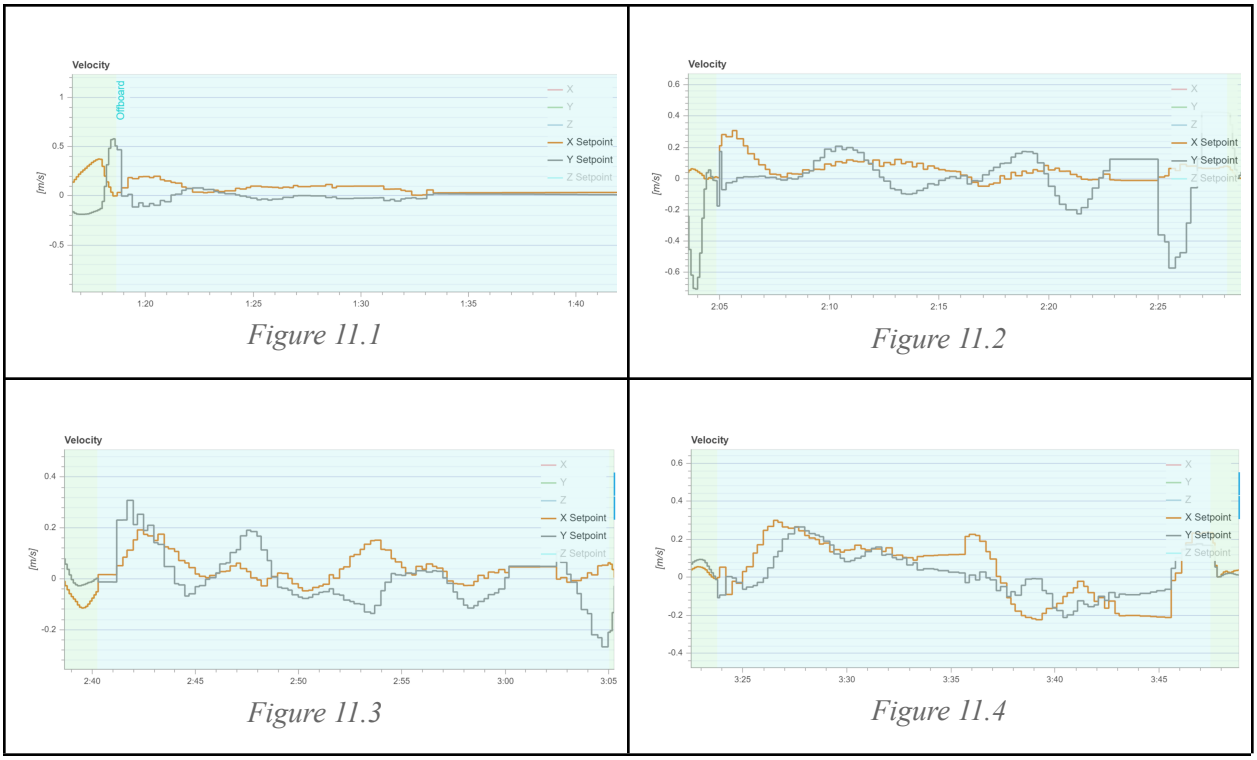

Figure 11.1

Figure 11.2

Figure 11.3

Figure 11.4

Figure 11: Drone Centering Velocity Setpoints per Time

4.2 Experimental Setup

To evaluate the success of the drone design, we conducted a single experiment to test our drone against the success metrics outlined in *Section 3.1*. The experiment took place outdoors with three sunflowers placed within a five-meter radius of the drone. We chose a five-meter radius because it allows us to evaluate the drone while it searches a region with no flowers but not take too long in between flower pollination attempts. We placed the sunflowers arbitrarily within this target region to evaluate how the system responds to an unknown environment. The sunflowers for this experiment were plastic sunflowers that were attached at fixed heights to metal rods in order to avoid downwash effects from the drone propellers disturbing the pollination process.

During this experiment, the drone's autonomous pollination routine was activated and it searched the region and attempted to pollinate every flower it detected. To measure successful pollination, each sunflower was seeded with a unique colored powder at the start of the experiment. This colored powder was used as a representation of real pollen. Successful pollination was recorded if there was colored powder on a flower it did not originate from. After the drone completed its autonomous routine, we investigated each flower that the drone had attempted to pollinate and looked for signs of damage. This experiment allowed us to test the drone's ability to find, target, and pollinate multiple flowers within a given region without damaging them.

4.3 Discussion of the Results

The team successfully developed a drone capable of performing the low-level objectives required of a pollinator. To evaluate the drone systems, the team ran unit tests which validated the systems ability to dynamically track the flower by repositioning the end effector, transfer pollen between two flowers, generate positional waypoints for region searching, stabilize over a flower with computer vision feedback, and descend to a target height to pollinate a flower.

These subsystems were integrated into an autonomous pollinating system that was evaluated with the experimental design outlined in *Section 4.2*. This experiment was run 14 times over two days. It was found that the system discovered an average of 2.14 flowers per test and attempted to pollinate 96% of all detected flowers. Flowers that were tested on had a 93% survival rate. The only non-successful metric was pollination success rate, as no flower was confirmed to be pollinated. This data is included in *Table 2*.

Timestamp	Flower Survival	Flower Detections	Attempted Pollinations	Successful Pollinations	CUT SHORT
4/24, 00:11	TRUE	1	1	0	**
4/24, 00:17	TRUE	1	1	0	**
4/24, 00:24	TRUE	1	1	0	**
4/24, 01:19	TRUE	2	2	0	
4/24, 01:24	TRUE	1	1	0	
4/24, 01:32	TRUE	1	1	0	
4/24, 01:39	FALSE	2	2	0	
4/24, 01:53	TRUE	2	2	0	
4/24, 02:05	TRUE	3	3	0	
4/24, 02:41	TRUE	3	3	0	
4/24, 13:30	TRUE	3	2	0	
4/24, 13:33	TRUE	5*	5	0	
4/24, 13:41	TRUE	2	2	0	
4/24, 13:54	TRUE	1	1	0	
4/24, 13:56	TRUE	1	1	0	**
4/24, 14:04	TRUE	2	2	0	
4/24, 14:10	TRUE	2	2	0	
4/24, 14:13	TRUE	1	1	0	
		AVG INCLUDING TESTS CUT SHORT (#)**		Attempted Pollination Given a Flower Detected (%)	
		1.89	1.83	97.1	
		AVG EXCLUDING TESTS CUT SHORT (#)		Attempted Pollination Given a Flower Detected (%)	
		2.14	2.07	96.7	

*Detected the same flower multiple time

**These tests were cut short in order to preserve battery consumption.

Table 2: Autonomous Pollination Routine Results

Compared to the success metrics outlined in *Section 3.1*, our pollination system was not completely successful, but it did achieve most of desired metrics. The drone was able to reliably detect the sunflowers in the search region, and would rarely have false positives or miss a flower once fully calibrated. By detecting on average 2.14 flowers per test. With this average, the

system's success rate at attempted pollination is 97.1% given a flower was detected. Referring back to *Table 1*, the system has proven successful for metric 1. Unfortunately, the designed drone system was unable to successfully pollinate any flowers due to challenges in sensing the distance between the pollen collector and the surface of the flowers. Because the distance was unknown, the drone was not able to consistently descend to a point where contact with the flower was made. This means we did not achieve success for the second metric. In terms of flower survival rate, our drone was able to repeat multiple trials without damaging the fake flowers. The flower survival rate was recorded to be 93%, resulting in metric 3 being completed. When looking at flowers pollinated per cycle, we see from our tests that our drone has enough battery life to be able to attempt to pollinate at least three flowers. On average we attempt to pollinate 2.07 flowers per test. This surpasses our goal of two flowers, as defined by success metric 4. Finally, our drone needed to be able to detect and pollinate at least one flower species, and since it was designed to be able to pollinate sunflowers, and was able to control itself over a sunflower, we can say we achieved success for metric 5.

5. Conclusion and Future Work

Our proposed autonomous pollination system is a quadrotor capable of housing onboard computers and vision systems that are used to autonomously navigate the desired region and search for flowers. To develop this system, we purchased a drone kit containing a Pixhawk flight controller and installed a Raspberry Pi onboard computer, an OpenMV camera for flower detection, and an actuated makeup brush that acted as a pollen-collecting end-effector. We developed a software architecture integrating all of these components allowing the drone to autonomously search a parameterized spiral search region. While searching this region, the drone uses the OpenMV camera to search the ground for flowers, and when a flower is detected, the drone will begin to autonomously stabilize over the flower before descending for a pollination attempt. After reaching the target altitude, the drone will consider the pollination attempt complete and return to the search altitude to resume searching for flowers. This work also proposes a miniaturized drone that is capable of housing all of the required sensors and actuators for the pollination system validated using the drone kit.

While we did achieve success for 4 of the 5 metrics outlined at the start of the project, the one metric we failed to meet was arguably the most important. Because our system was unable to pollinate any flowers, we cannot conclusively say that our system is a full pollinating system; however, we have shown that our system has the functionality required for a full pollinating system. This means that the platform we have developed may be expanded upon in future work to a point where it can reliably pollinate target flowers and become a fully autonomous pollinating system.

5.1 Future Work

Future work should focus on the flower pollination aspect of this project. Due to time constraints, we were unable to tune the system to successfully pollinate flowers. One major obstacle was the drone's ability to predict its current height with the limited range of the time of flight sensor equipped on the drone. Future work should implement a more accurate rangefinder to estimate the drone's altitude and avoid challenges caused by the drone flying outside of the maximum range of the time of flight sensor. The dynamically tacking end-effector should also be implemented to increase pollination chances. An additional sensor could be added to the makeup brush to confirm contact with a flower has been made. Future work could also implement the object-tracking code for the camera and improve the flower detection code allowing it to detect multiple species of flowers.

5.2 Lessons Learned

Our team learned multiple valuable lessons from this experience. The first was the importance of proper unit testing during development. Designing and conducting tests to validate desired behavior for sub-components is essential before combining subsystems into the final system design. Similarly, another lesson we learned was the importance of developing a proof of concept with easily accessible tools and products. After a proof of concept has been established, the system can be modified to better fit the needs of the project. Another lesson we learned is to evaluate the strengths and weaknesses of each team member and utilize past experiences to complete the project.

5.3 Covid-19 Statement

One unavoidable aspect of our project was the ongoing challenges presented by the global Covid-19 pandemic. Due to health guidelines, our team was unable to meet in person for parts of the year and had to take precautions with regards to working on hardware together. Another challenge the team faced was delays in the global supply chain, which increased the shipping time to get parts from overseas and increased the length of the prototyping cycle. This resulted in failed ideas taking longer to resolve which impacted the number of features that could be developed for the final demonstration.

6. Citations

Arirang News. (2019). Scientists use drones to artificially pollinate strawberries [Video file]. Retrieved from https://www.youtube.com/watch?v=oKMddPi1e-c

Aslan, C. E., Liang, C. T., Galindo, B., & Topete, W. (2016). The Role of Honey Bees as Pollinators in Natural Areas. Retrieved February 25, 2021, from https://www.fs.fed.us/psw/publications/liang/psw_2016_liang001_aslan.pdf

Birds Eye Aerial Drones. (2019). Plan Bee, A Bees Best Friend - plan bee smart phone controlled pollinator aerial drone. Retrieved September 25, 2020, from https://birdseyeaerialdrones.com/plan-bee-drone/

Bouktir, Yasser & Haddad, Moussa & Chettibi, Taha. (2008). Trajectory planning for a quadrotor helicopter. 2008 Mediterranean Conference on Control and Automation - Conference Proceedings, MED'08. 1258 - 1263. 10.1109/MED.2008.4602025.

Chechetka S, Yu Y, Tange M, Miyako E Chem, 2 (2017) Materially Engineered Artificial Pollinators. Retrieved from, https://www.sciencedirect.com/science/article/pii/S2451929417300323

Chen, C., Tian, Y., Lin, L., Chen, S., Li, H., Wang, Y., & Su, K. (2020). Obtaining World Coordinate Information of UAV in GNSS Denied Environments. Sensors, 20(8), 2241. doi:10.3390/s20082241

Chen, Y., Zhao, H., Mao, J. *et al.* Controlled flight of a microrobot powered by soft artificial muscles. *Nature* 575, 324–329 (2019). https://doi.org/10.1038/s41586-019-1737-7

Chong Shen, Zesen Bai, Huiliang Cao, Ke Xu, Chenguang Wang, Huaiyu Zhang, Ding Wang, Jun Tang, Jun Liu, "Optical Flow Sensor/INS/Magnetometer Integrated Navigation System for MAV in GPS-Denied Environment", *Journal of Sensors*, vol. 2016, Article ID 6105803, 10 pages, 2016.https://doi.org/10.1155/2016/6105803

Dias, P. A., Tabb, A., & Medeiros, H. (2018). Apple flower detection using deep convolutional networks. *Computers in Industry*, *99*, 17–28. https://doi.org/10.1016/j.compind.2018.03.010

Darnaud, G. (2016, April 19). Shrinking bee populations are being replaced by human pollinators. Retrieved March 16, 2021, from https://www.globalcitizen.org/en/content/life-without-bees-hand-human-pollination-rural-chi/

Goodwin, M. (2012). *Pollination of crops in Australia and New Zealand*. Barton, A.C.T.: RIRDC.

Hehn, M., & D'Andrea, R. (2011). Quadrocopter trajectory generation and control. IFAC Proceedings Volumes (IFAC-PapersOnline), 44(1 PART 1), 1485–1491. https://doi.org/10.3182/20110828-6-IT-1002.03178

Hricko, Jaroslav & Havlík, Štefan. (2019). Exchange of Effectors for Small Mobile Robots and UAV: Proceedings of the 27th International Conference on Robotics in Alpe-Adria Danube Region (RAAD 2018). 10.1007/978-3-030-00232-9_32.

Hopwood, J., A. Code, M. Vaughan, D. Biddinger, M. Shepherd, S.H. Black, E. Lee-Mader, and C. Mazzacono. 2016. *How Neonicotinoids Can Kill Bees: The Science Behind the Role These Insecticides Play in Harming Bees.* 2nd Ed. 76 pp. Portland, OR: The Xerces Society for Invertebrate Conservation.

Jaramillo, J., Maus, C., & Breukelen-Groeneveld, C. (2018, September). The Importance of

Insect Pollinators for Agriculture. Retrieved March 07, 2021, from https://www.cropscience.bayer.com/sites/cropscience/files/inline-files/BEEINFOrmed_7_The-Importance-of-Insect-Pollinatorsjlouz8q1.pdf

J. R. Kutia, K. A. Stol and W. Xu, "Canopy sampling using an aerial manipulator: A preliminary study," *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, Denver, CO, 2015, pp. 477-484, doi: 10.1109/ICUAS.2015.7152326.

Kaplan, K. (2021, February 2). Ars honey bee health. Retrieved March 01, 2021, from https://www.ars.usda.gov/oc/br/ccd/index/#:~:text=Typical%20average%20annual%20losses%20jumped,continued%2C%20averaging%20about%2030%20percent.

Kluser S, Peduzzi P. (2007), "Global Pollinator Decline: A Literature Review", UNEP/GRID, Geneva.

L. Meier, D. Honegger and M. Pollefeys, "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, 2015, pp. 6235-6240, doi: 10.1109/ICRA.2015.7140074.

Lihoreau, Mathieu, et al. "Monitoring Flower Visitation Networks and Interactions between Pairs of Bumble Bees in a Large Outdoor Flight Cage." *PLOS ONE*, vol. 11, no. 3, 2016, p. 1. *PLOS ONE*, https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0150844. Accessed 09 03 2021.

Oppenheim, D. , Edan, Y. , Shani, G. (2017). 'Detecting Tomato Flowers in Greenhouses Using Computer Vision'. World Academy of Science, Engineering and Technology, Open Science Index 121, International Journal of Computer and Information Engineering, 11(1), 104 - 109.

Rinderer, T., Harris, J., Hunt, G., & Guzman, L. (2010). Breeding for resistance to Varroa destructor in North America*.

Tripodi A, Hardin T. 2020. New Pest Response Guidelines. *Vespa mandarinia* Asian Giant Hornet. United States Department of Agriculture. (13 April 2020)

Watson, Ryan & Gross, Jason & Kilic, Cagri. (2018). Design of an Autonomous Precision Pollination Robot.

Zhang, G., He, Y., Dai, B., Gu, F., Yang, L., Han, J., Liu, G. Aerial Grasping of an Object in the Strong Wind: Robust Control of an Aerial Manipulator. *Appl. Sci.* 2019, *9*, 2230.

7. Appendix

7.1 Main Node Containing High Level State Machine

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "std_msgs/Float64.h"
#include "drone_architecture/flower.h"
#include "drone_architecture/pixhawkInterface.h"
#include "drone_architecture/endEffectorControl.h"
#include "drone_architecture/flowerData.h"
#include "drone_architecture/initializeSearch.h"
#include "drone_architecture/nextWaypoint.h"
#include "PID_Driver.cpp"
#include <geometry_msgs/Twist.h>
#include <geometry_msgs/TwistStamped.h>
#include <geometry_msgs/PoseStamped.h>
#include <mavros_msgs/HomePosition.h>
#include <mavros_msgs/CommandHome.h>
#include "mavros_msgs/State.h"
#include <geometry_msgs/Pose.h>
#include <sensor_msgs/NavSatFix.h>
#include <geographic_msgs/GeoPoint.h>
#include <drone_architecture/updateGains.h>
#include <iostream>
#include <unistd.h>
#include <math.h>
#include <sys/time.h>
#define PI 3.14159265

using namespace std;

enum Option
{
  sendPositionalMessages, offBoard, arming, initStateMachine, searchForFlower, cyclical_search,
lockOnFlower, centerFlower, allignServo,
  lowerToFlower, pollinateFlower, returnToSearchHeight, resetServoPosition, updateHome
};

int choice = initStateMachine;
drone_architecture::flowerData flowerDataMessage;
drone_architecture::flower targetFlower;
mavros_msgs::State current_state;
int desiredID = 0;
PID_Driver pid_velocity_x(0.00375, 0, 0.0001);
PID_Driver pid_velocity_y(0.00375, 0, 0.0001);

// globals used in search path generation
// struct timeval tp;
double start_search_time;
double search_pattern_freq = 5;
double waypoint_count = 0;
// globals used for velocity setpoints
geometry_msgs::Pose lastSetpoint;
// globals used for flower pollination status
double time_flower_pollinated = 0;
double time_last_flower_seen = 0;

bool TESTING = false;

double getTime()
```

```
{
        struct timeval tp;
        gettimeofday(&tp, NULL);
        return (double)tp.tv_sec + (double)tp.tv_usec / 1000000;
}

void flowerCallback(const drone_architecture::flowerData& msg)
{
        if(!current_state.mode.compare("OFFBOARD") || TESTING)
        {
        if(getTime() - time_flower_pollinated > 10)
        {
                // cout << "see flower Case" << endl;
                // Update that a flower has been seen
                time_last_flower_seen = getTime();

                if(choice == searchForFlower)
                {
                choice = lockOnFlower;
                flowerDataMessage = msg;
                }
                else
                {
                try
                {
                        targetFlower = msg.flowerArray[desiredID];
                }
                catch(const std::exception& e) //Flower disapeared for some reason
                {
                        choice = lockOnFlower;
                        flowerDataMessage = msg;
                }
                }
                if(choice == cyclical_search)
                {
                cout << "!!!!!!!!!!!!!!! FLOWER DETECTED !!!!!!!!!!!!!!!!" << endl;
                choice = lockOnFlower;
                }
        }
        }
}

geometry_msgs::PoseStamped currentPosition;
sensor_msgs::NavSatFix globalPosition;
geometry_msgs::PoseStamped droneOrigin;
mavros_msgs::HomePosition home_pose;
bool have_home{false};

void gainsCallback(const drone_architecture::updateGains& msg)
{
        pid_velocity_x.setGains(msg.p, msg.i, msg.d);
        pid_velocity_y.setGains(msg.p, msg.i, msg.d);
}

void state_cb(const mavros_msgs::State::ConstPtr& msg)
{
        current_state = *msg;
}

void pixhawkCallback(const geometry_msgs::PoseStamped::ConstPtr& msg)
{
        currentPosition = *msg;
}
```

```
void global_pose_cb(const sensor_msgs::NavSatFix::ConstPtr& msg)
{
	globalPosition = *msg;
}

float compassHeading;
void compassCallback(const std_msgs::Float64& msg)
{
	compassHeading = msg.data;
}

void State_Machine(ros::Publisher EndEffector_pub, ros::Publisher pixHawk_pub, ros::Publisher
search_pub, ros::ServiceClient next_waypoint_client)
{
	switch (choice)
	{
	case initStateMachine:
	{
		// wait for FCU connection
		if(ros::ok() && current_state.connected || TESTING){
		drone_architecture::endEffectorControl endEffectorMessage; //reset servo
		endEffectorMessage.m_XCentroid = 0;
		endEffectorMessage.m_YCentroid = 110;
		endEffectorMessage.m_reset = true;
		droneOrigin = currentPosition;
		EndEffector_pub.publish(endEffectorMessage);

		choice = sendPositionalMessages;
		}
	}
	break;

	case updateHome:
	{

	}
	break;

	case sendPositionalMessages:
	{
		cout << "sendPositionalMessages Case" << endl;
		drone_architecture::pixhawkInterface InterfaceMessage;
		geometry_msgs::Pose position_message;
		position_message.position.x = currentPosition.pose.position.x;
		position_message.position.y = currentPosition.pose.position.y;
		position_message.position.z = currentPosition.pose.position.z + 2;
		position_message.orientation = currentPosition.pose.orientation;

		InterfaceMessage.pose = position_message;
		InterfaceMessage.flight_command = "POSITION";
		for(int i=0; i < 10; i++)
		{
		pixHawk_pub.publish(InterfaceMessage);
		}

		// Record the positional setpoint message
		lastSetpoint = position_message;
		choice = searchForFlower;
	}
	break;

	// case offBoard:
	// {
	//	cout << "offBoard Case" << endl;
```

```
        //      drone_architecture::pixhawkInterface InterfaceMessage;
        //      InterfaceMessage.flight_command = "OFFBOARD";
        //      pixHawk_pub.publish(InterfaceMessage);
        //      choice = arming;
        //      droneOrigin = currentPosition;
        //      sleep(5);
        // }
        // break;

        // case arming:
        // {
        //      cout << "arming Case" << endl;
        //      drone_architecture::pixhawkInterface InterfaceMessage;
        //      InterfaceMessage.flight_command = "ARM";
        //      pixHawk_pub.publish(InterfaceMessage);
        //      choice = searchForFlower;
        //      sleep(5);
        // }
        // break;

        case searchForFlower:
        {
                // cout << "searchForFlower Case" << endl;
                // cout << current_state.mode << endl;

                if(!current_state.mode.compare("OFFBOARD") || TESTING)
                {
                // ====================================
                // Start timing the search for flower state
                // start_search_time = time(nullptr);
                // gettimeofday(&tp, NULL);
                start_search_time = getTime();

                // Input Variables
                double r_max = 9;             // maximum radius of search in meters
                double v_max = 0.25;          // frequency of generatet setpoints in Hz -- must be
larger than 2Hz
                double f_hz = 5;              // maximum velocity in m/s
                double search_height = 0;   //1.5;   // search height in meters

                // Publish the initialization msg
                drone_architecture::initializeSearch InitSearch;
                InitSearch.region_type = "SPIRAL";
                InitSearch.x_axis = r_max;
                InitSearch.y_axis = v_max;
                InitSearch.height = search_height;
                InitSearch.density = f_hz;
                InitSearch.currentLocation = currentPosition.pose;
                search_pub.publish(InitSearch);
                cout << "=========== SENDING SEARCH INFO ============" << endl;

                // send to search algorithm state
                choice = cyclical_search;
                // ====================================
                }
                else
                {
                choice = searchForFlower;
                }
        }
        break;

        case cyclical_search:
        {
```

```
                double elapsed_time = getTime() - start_search_time;

                if(elapsed_time > 1/search_pattern_freq)
                {
                // Reset the timer
                // cout << "======= GO TO NEW WAYPOINT ========" << endl;
                // cout << "Target Freq: " << search_pattern_freq << "   Actual Freq: " <<
1/elapsed_time << "   Latency: " << elapsed_time - 1/search_pattern_freq << endl;
                start_search_time = getTime();

                // Make the next waypoint service call
                drone_architecture::nextWaypoint next_waypt_srv;
                next_waypt_srv.request.go_to_pose = true;
                next_waypt_srv.request.next_pose = true;
                next_waypt_srv.request.publish = true;

                if(next_waypoint_client.call(next_waypt_srv))
                {
                        // cout << "Waypoint Recieved?: " << (int)next_waypt_srv.response.arrived
<< endl;
                        cout  << "WAYPOINT COMMAND SENT  " << waypoint_count << "/" <<
next_waypt_srv.response.num_pts << endl;
                        waypoint_count++;
                }
                else
                {
                        cout << "ERROR NEXT WAYPOINT SERVICE FAILURE" << endl;
                }
                }
        }
        break;

        case lockOnFlower:
        {
                // int numberOfFlowers = flowerDataMessage.flowerArray.size();
                // desiredID = rand() % numberOfFlowers;
                cout << "lockOnFlower Case" << endl;

                // Stop waypoints from being called
                drone_architecture::nextWaypoint next_waypt_srv;
                next_waypt_srv.request.go_to_pose = false;
                next_waypt_srv.request.next_pose = false;
                next_waypt_srv.request.publish = false;

                if(next_waypoint_client.call(next_waypt_srv))
                {
                // cout << "Waypoint Recieved?: " << (int)next_waypt_srv.response.arrived << endl;
                }
                else
                {
                cout << "ERROR NEXT WAYPOINT SERVICE FAILURE" << endl;
                }

                choice = centerFlower;

                // Enter center on flower and perform velocity control
        }
        break;

        case centerFlower:
        {
                int setpointX = 130;
                int setpointY = 120;
                float x_velocity = pid_velocity_x.setVelocity(setpointY, targetFlower.m_Y);
```

```
                float y_velocity = -pid_velocity_y.setVelocity(setpointX, targetFlower.m_X);
                // float x_velocity = 1;
                // float y_velocity = 0;

                drone_architecture::pixhawkInterface InterfaceMessage;
                geometry_msgs::Twist velocity_message;

                int heading_offset = (int)compassHeading - 90;
                int theta = (heading_offset)%360;
                float x_vel_enu = x_velocity * cos(theta * (PI/180)) - y_velocity * sin(theta *
(PI/180));
                float y_vel_enu = -1*(x_velocity * sin(theta * (PI/180)) + y_velocity * cos(theta
* (PI/180)));
                float z_vel_enu = 0;

                float max_vel = 1.5;
                if(x_vel_enu > max_vel)
                {
                x_vel_enu = max_vel;
                }
                if(x_vel_enu < -max_vel)
                {
                x_vel_enu = -max_vel;
                }
                if(y_vel_enu > max_vel)
                {
                y_vel_enu = max_vel;
                }
                if(y_vel_enu < -max_vel)
                {
                y_vel_enu = -max_vel;
                }

                // Check allignment to see if it is time to lower
                float vel_threshold = 0.15;
                // A magic distance offset, who knows why it is this way
                float magic_float = -1.4;

                if(abs(x_vel_enu) < vel_threshold && abs(y_vel_enu) < vel_threshold){
                // Eneter a state of lowering
                z_vel_enu = -0.2;
                }
                // Some check to assure we have lowered enough
                if(currentPosition.pose.position.z - droneOrigin.pose.position.z+magic_float < 0)
                {
                cout << "!!!!!!!!!!!!!!!!!!!!!! FLOWER POLLINATION DETECTED
!!!!!!!!!!!!!!!!!!!!!!" << endl;
                time_flower_pollinated = getTime();
                choice = cyclical_search;
                }

                velocity_message.linear.x = x_vel_enu;
                velocity_message.linear.y = y_vel_enu;
                velocity_message.linear.z = z_vel_enu;

                InterfaceMessage.twist = velocity_message;
                InterfaceMessage.flight_command = "VELOCITY";

                double elapsed_time = getTime() - start_search_time;

                if(elapsed_time > 1/search_pattern_freq)
                {
                pixHawk_pub.publish(InterfaceMessage);
                start_search_time = getTime();
```

```
                cout << "Center Flower Velocities (Robot Frame): " << x_velocity << ", " <<
y_velocity << endl;
                cout << "Elevation From Trigger: " << currentPosition.pose.position.z -
droneOrigin.pose.position.z+magic_float << endl;;
                }

                // Bail out if a flower has not been seen for n seconds
                double time_since_last_flower = getTime() - time_last_flower_seen;
                double max_seconds_without_flower = 2.5;
                if(time_since_last_flower > max_seconds_without_flower)
                {
                choice = cyclical_search;
                }
        }
        break;

        case allignServo:
        {
                cout << "allignServo Case" << endl;
                drone_architecture::endEffectorControl endEffectorMessage;
                endEffectorMessage.m_XCentroid = targetFlower.m_X;
                endEffectorMessage.m_YCentroid = targetFlower.m_Y;
                endEffectorMessage.m_reset = false;
                EndEffector_pub.publish(endEffectorMessage);
                choice = allignServo;
        }
        break;

        case lowerToFlower:
        {
                cout << "lowerToFlower Case" << endl;
                choice = lowerToFlower;
        }
        break;

        case pollinateFlower:
        {
                cout << "pollinateFlower Case" << endl;
                choice = pollinateFlower;
        }
        break;

        case returnToSearchHeight:
        {
                cout << "returnToSearchHeight Case" << endl;
                choice = returnToSearchHeight;
        }
        break;

        case resetServoPosition:
        {
                // 50 is 0
                // 96 is 45
                // 143 is 90
                cout << "resetServoPosition Case" << endl;
                choice = resetServoPosition;
        }
        break;

        default:
        {
                cout << "INVALID INPUT" << endl;
        }
        break;
```

```
        }
}


int main(int argc, char **argv)
{
        ros::init(argc, argv, "Main_Node");
        ros::NodeHandle n;

        ros::Publisher EndEffector_pub =
n.advertise<drone_architecture::endEffectorControl>("EndEffectorControl", 1);
        ros::Publisher pixHawk_pub = n.advertise<drone_architecture::pixhawkInterface>("/command",
1);
        ros::Publisher search_pub =
n.advertise<drone_architecture::initializeSearch>("/search_initialize", 5);

        ros::Subscriber Flower_sub = n.subscribe("FlowerData", 1, flowerCallback);
        ros::Subscriber Pixhawk_sub = n.subscribe("mavros/local_position/pose", 1,
pixhawkCallback);
        ros::Subscriber Compass_sub = n.subscribe("mavros/global_position/compass_hdg", 1,
compassCallback);
        ros::Subscriber state_sub = n.subscribe<mavros_msgs::State>
                ("mavros/state", 10, state_cb);
        ros::Subscriber update_gains_sub = n.subscribe("updateGains", 1, gainsCallback);

        ros::ServiceClient next_waypoint_client =
n.serviceClient<drone_architecture::nextWaypoint>("/next_waypoint");

        ros::Rate loop_rate(20);
        ros::spinOnce();

        while (ros::ok()) {
        State_Machine(EndEffector_pub, pixHawk_pub, search_pub, next_waypoint_client);
        ros::spinOnce();
        loop_rate.sleep();
        }
}
```