ROB498: Autonomous Drone for Nuclear Plant Safety

Team Pu239: Ryan Zazo, Kanav Singla, Kajanan Chinniah, Siddarth Narasimhan

1 Introduction

The goal of our capstone project is to build an autonomous drone for use in nuclear plant inspections. Drones are a very popular site-inspection tool because of their agility and maneuverability in cluttered environments, ability to offer a new perspective from different viewpoints and provide new avenues for navigation. To be useful in such inspection tasks, the drone must be able to maintain a stable hover while awaiting further instructions, follow paths described by waypoints and autonomously avoid obstacles. This report will detail our design procedure, the results of our tests, and what we learned in the process.

2 Design Philosophy

Central to every design is the underlying philosophy that guided it. Our team's design philosophy revolves around three core values: safety, simplicity & elegance, and attention to detail. These values have not only shaped our approach to the design process but have also guided our team organization and individual responsibilities.

Safety: This is the most critical aspect of our guiding principles; to avoid injuries and harm during operation. We achieve this by always testing our code in simulation first before executing on the real drone. This approach helps us prevent accidents while we are in the debugging stage and allows us to test our algorithms in a risk-free environment. Additionally, one of our teammates, Ryan, is a certified and licensed pilot for aerial vehicles. To mitigate potential risks and harm done to property, we have restricted the operation of our drone exclusively to Ryan, which allows us to execute safe maneuvers during unforeseen malfunction or erratic behavior during flight. To ensure the safety aspect of our design philosophy, we organized our team by assigning specific roles and responsibilities related to safety. For instance, one team member was responsible for conducting risk assessments, another for maintaining safety guidelines and protocols, and Ryan was designated as the drone pilot.

Simplicity and Elegance: This value entails formulating solutions that are intuitive, easy to implement, and robust. In doing so, we can streamline our workflow as everyone can contribute to the development of each component without having a significant knowledge gap. With this in mind, we were more focused on implementing algorithms from scratch instead of using off-the-shelf solutions that require more background knowledge and understanding. Taking the path of least resistance also allowed us to complete our milestones in a timely manner and gave us enough time to build and test both in simulation and real-life. In doing so, we were able to build solutions that were robust while exceeding the design requirements. Our team organization followed a modular approach, where each member was responsible for a specific aspect of the design process, such as obstacle detection or path planning. By assigning these responsibilities, we ensured that each team member could focus on simplicity and elegance within their domain.

Attention to Detail: Attention to detail means that we always ensure every part of our solution is clean and efficient. Our goal here was to focus on creating good documentation, following best software practices, and ensuring rigorous testing of the design requirements. We wrote our code following OOP principles and used GitHub for documentation, versioning, and to do code reviews within the team. Doing this allows us to build a stronger and collaborative workflow so that we minimize our mistakes and improve the overall quality of our solution. We established a system of peer reviews and regular meetings to discuss progress and address any concerns. Additionally, we assigned a team member as a quality assurance lead, responsible for overseeing the attention to detail aspect of our design philosophy. To guide this, we also reviewed "The Design of Everyday Things", by Don Norman [1]. This source provided us with valuable insights into the importance of user-centered design and helped shape our approach to creating an effective and efficient drone system.

In conclusion, our team's design philosophy, grounded in safety, simplicity & elegance, and attention to detail, has guided our solution process and team organization. By assigning specific roles and responsibilities related to these core values and by distributing the workload, we were able to work more cohesively as a team throughout the term.

3 Background and Related Work

In developing our drone system, we made use of various software, hardware, and firmware frameworks. Our choices were informed by the available hardware, our team's prior experience, and the requirements of the project.

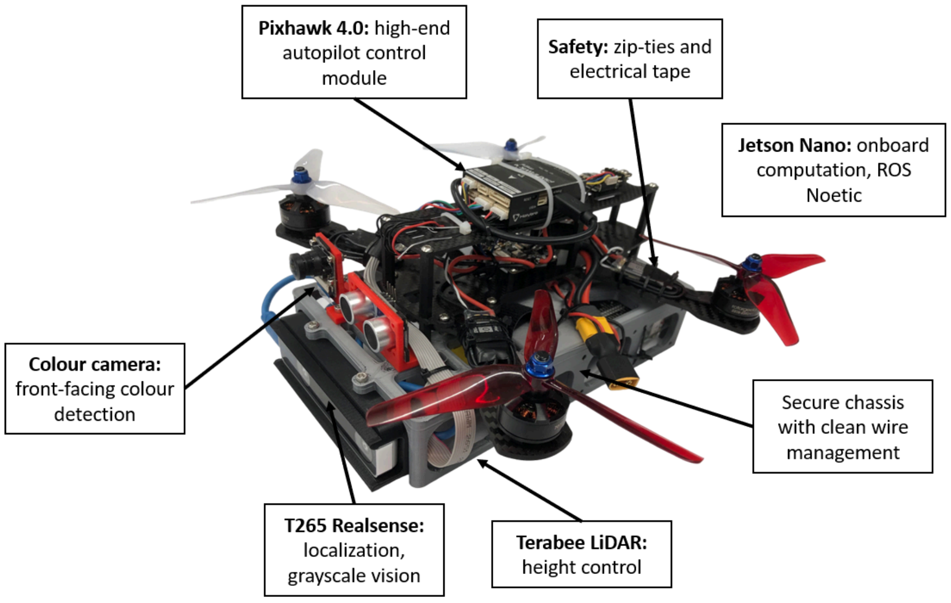
Pixhawk 4.0: high-end autopilot control module
Safety: zip-ties and electrical tape
Jetson Nano: onboard computation, ROS Noetic
Colour camera: front-facing colour detection
Secure chassis with clean wire management
T265 Realsense: localization, grayscale vision
Terabee LiDAR: height control

Figure 1: *Diagram of our drone with hardware components and features. The ultrasonic sensor was removed and has not been used for any of the challenge tasks. This photo was used as it was the most clear one available.*

3.1 Hardware Components

Our hardware components included a Jetson Nano, a Pixhawk-4 Mini, an Intel Realsense T265 tracking camera, a Sony IMX color camera, and a Terabee EVO distance sensor. These components were chosen due to their compatibility, performance, and suitability for the specific tasks involved in the project.

A few modifications to the drone mounting frame were made to complement our approach. An additional frame was designed to remount the colour camera and face the front of the drone, as shown in Figure 2. This simplified our software approach for Challenge Task 4's obstacle recognition and avoidance.

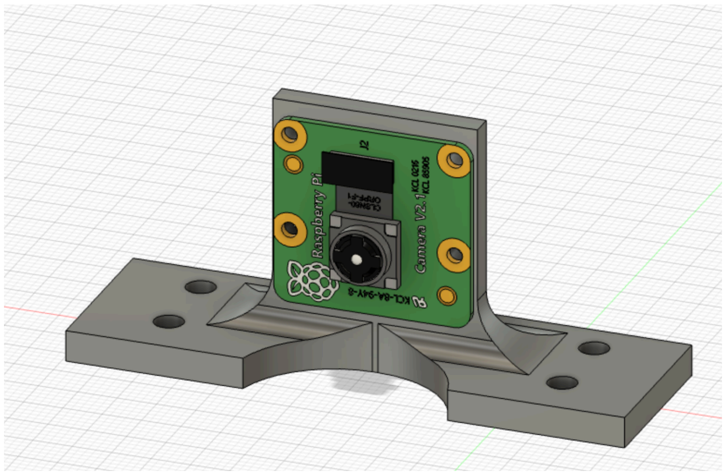

Figure 2: *Our 3D designed mount, to allow for a front-facing colour camera.*

3.2 Software and Firmware Frameworks

Instead of using the standard Ubuntu 18.04 image for the Jetson, we instead opted to upgrade to an Ubuntu 20.04 image. This would allow us to use more updated and supported software including ROS Noetic and Python 3, both of which would grant us access to a more extensive pool of resources and enable us to build higher quality programs. We flashed a custom patched version of Jetpack 4.6.1 with Ubuntu 20.04 [2], which comes preinstalled with essential libraries such as PyTorch and OpenCV.

For communication between our drone and the Jetson Nano, we utilized the MAVROS bridge, which connects the Jetson Nano to the Pixhawk4 mini. This bridge allows for command transmission and telemetry data reception from the drone over the ROS network. Furthermore, for localization, we chose the AurterionVIO package, which integrates the T265 tracking camera's visual odometry estimation into the Pixhawk's internal pose EKF. Additionally, we used the Terabee EVO distance sensor to provide ground distance data for the Pixhawk EKF's height estimate.

4 Design Objectives and Solution Details

In this section, we explain our methodology for completing the challenge tasks. Our objectives (**O**) and metrics (**M**) used to measure compliance are detailed. We also include our results at the end of each subsection.

4.1 Challenge Task 2

4.1.1 Challenge Description

The second challenge task aims to have the drone hover at a fixed altitude over an area of interest for a specified duration. To perform well in the task, the drone must remain within a particular zone, called the primary zone, to avoid loss of points. The task consists of 3 parts:

1. Upon receiving a LAUNCH command, the drone should ascend to the desired altitude and begin to hover
2. Upon receiving a TEST command, the drone should remain within the primary zone for 30 seconds
3. Upon receiving a LAND command, the drone should land on the ground

This task has two modes of operation; flying with Vicon data available and flying using only on-board sensing.

4.1.2. Objectives and Task Functional Analysis Metrics

O1.1: The drone should ascend to the specified altitude once the "LAUNCH" command is sent

- **M1.1.1:** Distance between the drone's altitude, measured from the Vicon, and the specified altitude.
    - **Criteria:** Smaller distances are preferred.

O1.2: The drone should remain at the desired altitude (e.g. hover) over the area of interest for a specified duration once the "TEST" command is sent.

- **M1.2.1:** The variation between the drone's current x and y positions and the starting position, all measured using the Vicon system.
    - **Criteria:** The drone's x and y positions should remain within $\pm$ 15 cm of the starting position during the 30 second measurement period using both on-board sensing and Vicon data.
- **M1.2.2**: The variation between the drone's current heading and the starting heading, all measured using the Vicon system.
    - **Criteria:** The drone's heading should should remain within $\pm 5^{o}$ relative to its initial position during the 30 second measurement period using both on-board sensing and Vicon data.
- **M1.2.3:** The drone's altitude, measured using the Vicon system.
    - **Criteria:** The drone's altitude should be of 150 ± 10 cm during the 30 second measurement period using both on-board sensing and Vicon data.

O1.3: The drone should successfully land once the "LAND" command is sent

- **M1.3.1:** The drone successfully and safely descends to the ground once "LAND" is sent.
    - **Criteria:** Descent should happen as soon as the command is sent.

4.1.3. Challenge Task 2 Design Overview

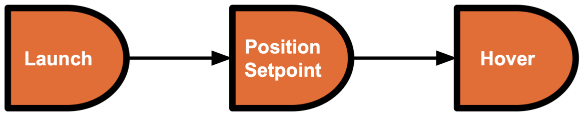

Figure 3: *Challenge Task 2 solution pipeline. Send launch command and then move position to setpoint using MAVROS was how we achieved this task.*

The Realsense T265 serves as our main odometry source, providing us a stable localization estimate with little drift. The Terabee range sensor was used for altitude measurements, as it provides an accurate and stable estimate for height and was found to be more reliable than the T265. We also configured the Pixhawk's internal Extended Kalman Filter and PID controller to fuse multiple sensors including the Realsense's IMU and Visual Odometry estimation in QGroundControl. In order to ascend the drone to the desired altitude, we simply sent the desired pose using topic /mavros/position_setpoint/local, supplied by MAVROS.

Our codebase was designed to be object oriented with multiple nodes to perform multiple independent tasks. For this task, we created a single node, called our “CommNode”. CommNode listens to the services described in the challenge documentation (LAUNCH, TEST, LAND and ABORT) and publishes and subscribes to MAVROS topics required to receive drone telemetry and send waypoint commands.

4.1.4. Challenge Task 2 Results

Our evaluation of challenge task 2 was split into two categories. First, we verified our solution in simulation. We tested all metrics in simulation and were able to satisfy them. We confirmed that the drone was responsive to the "LAUNCH" command (**O1.1**), and that the drone reached the desired altitude and hover in place for 30s by reading the output of the pose topic, /mavros/local_position/pose (**O1.2**). After the verification in simulation, we tested our implementation on the real drone. As seen in Figure 4, our approach successfully satisfied all our metrics, suggesting that we satisfied all the high level objectives. Due to our high performance with sensors only, we chose to perform our Vicon test with on-board sensors only as well, and achieved similar results.

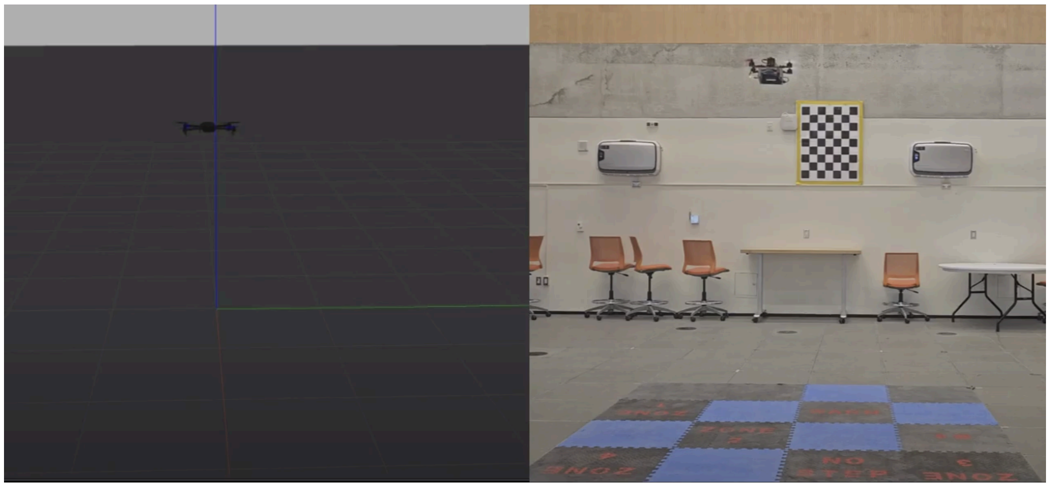
Figure 4: *Successful implementation of challenge task 2 in both simulation and real life.*

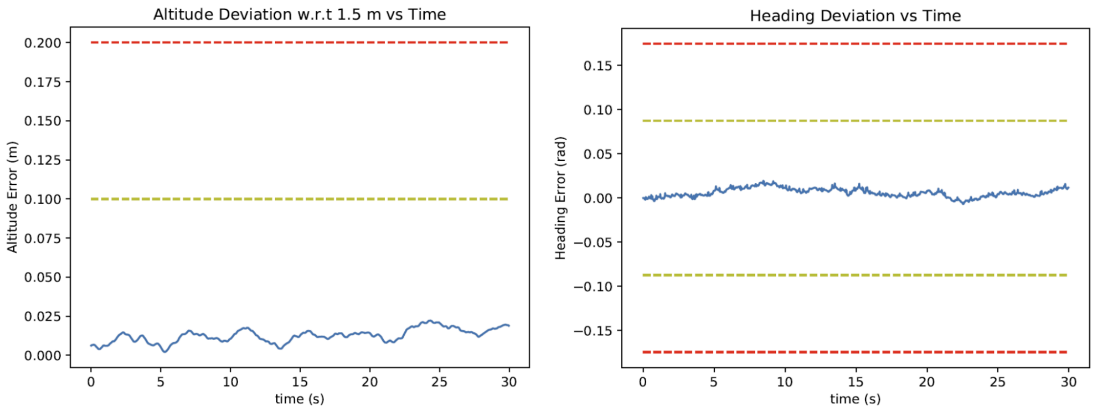

Figure 5: *Depicts our performance for Challenge Task 2. Depicts our performance when testing using on-board sensors only. Our drone's maximum altitude deviation was $\pm$ 2.5 cm relative to 1.5 m, and our maximum heading deviation was about $\pm$ 0.05 rad.*

4.2 Challenge Task 3

4.2.1 Challenge Description

The third challenge task aims to have the drone navigate a set of waypoints sequentially. Each waypoint is a 3D point in space that the drone must visit. This task consists of 3 parts:

1. The drone must takeoff after receiving a LAUNCH command, as in Challenge Task 2
2. Follow the set of waypoints published when receiving the TEST command.
3. The drone must land upon receiving the LAND command upon completing the trajectory.

Once again, this task contains two modes (1) - flying with Vicon data available and (2) - flying with on-board sensing only.

4.2.2. Objectives and Task Functional Analysis Metrics

This section summarizes the objectives we had for our drone, and the metrics we used to ensure that the drone satisfied each one. The objectives from Challenge Task 2 apply (**O1.1**, **O1.2** and **O1.3**) but have been omitted for the sake of brevity.

O2.1: The drone must visit every waypoint sequentially and in order

- **M2.1.1:** Within 35 cm of a waypoint when using both on-board sensors and Vicon data.
  - **Criteria:** Drone must be able to reach a waypoint using either on-board sensors and Vicon Data.
- **M2.1.2:** Closest distance to each waypoint after a trajectory.
  - **Criteria:** The drone must be within 35cm of the center of each waypoint.
- **M2.1.3**: Number of waypoints traversed in order
  - **Criteria:** All 7 waypoints must be traversed in order.

O2.2: The drone should navigate through all the waypoints quickly

- **M2.2.1:** Time to traverse all waypoints.
  - **Criteria:** The drone must navigate through all waypoints in under 60 seconds
  - **Constraint:** This must be accomplished under 120 seconds.

4.2.3. Challenge Task 3 Design Overview

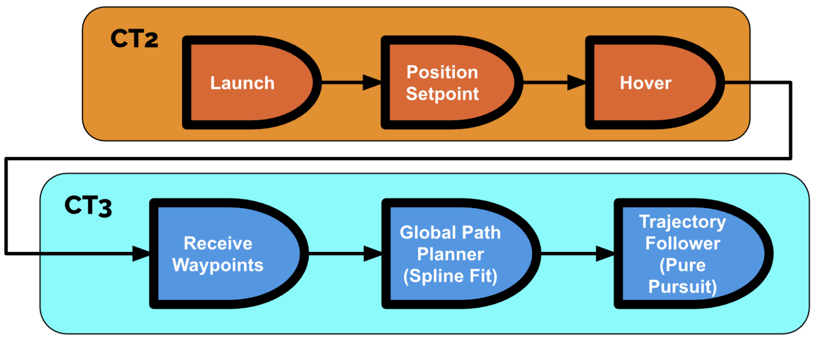
CT2
Launch
Position Setpoint
Hover
CT3
Receive Waypoints
Global Path Planner (Spline Fit)
Trajectory Follower (Pure Pursuit)

Figure 6: *Challenge Task 3 pipeline as an extension to Challenge Task 2.*

We begin by first launching and wait until we achieve a stable hover. Once we are satisfied with the hover and receive the TEST command, we take the last published Vicon frame, and then compute the transformation between the drone's frame (based on our local pose) and the world frame.

Once this transformation is computed, we transform all the waypoints into the drone's local frame, and then we perform a cubic spline fit to generate a trajectory traversing all the waypoints. We leverage scipy's interpolation module to perform this cubic spline fit. We chose to do a spline fit in order to ensure that the drone follows a smooth path when trying to navigate to each waypoint, making it kinematically easy to follow and limits the possibility of a jerk occurring, which can lead to an increase in error or accidents.

Finally, for our controller, we leverage a simple position-based pure pursuit controller. This is done by continuously updating the setpoint that our drone must follow to be a "look ahead" distance away. We chose this approach because it was a fairly simple controller to implement, and allowed us to tune our controller for high speed relatively easily. However, one major downside of our controller was the fact that it doesn't employ any guarantee that we'll reach a waypoint, which sometimes led to us missing waypoints. In order to solve this, we added a guarantee by checking if we're close to a goal, and if we are, we chose to target the goal and “wiggle” near it in the shape of a box, in an attempt to make sure that we reached the goal.

4.2.4. Challenge Task 3 Results

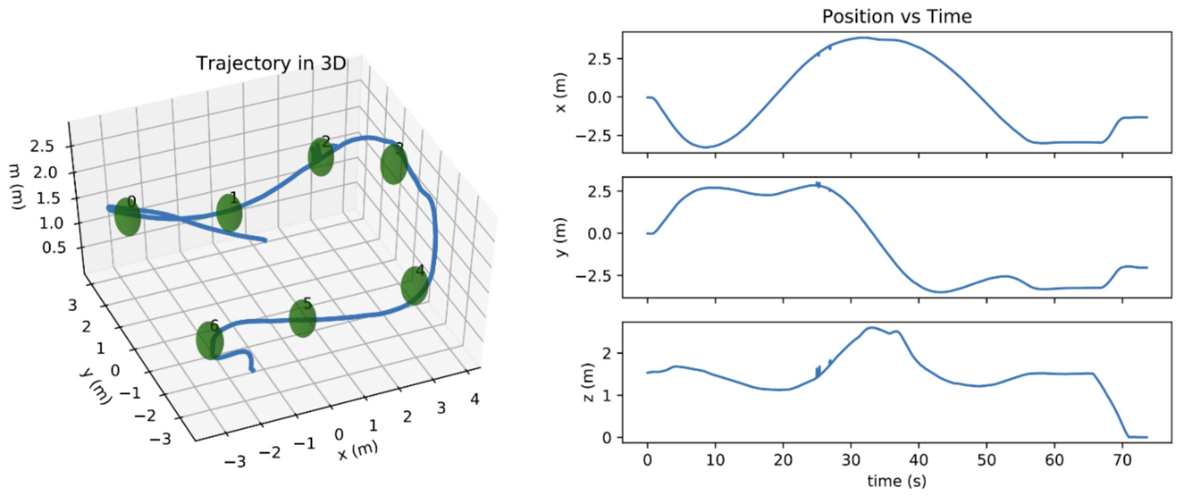

Figure 7: *Trajectory of our algorithm under simulation alongside plots to see how our position changes in the 3 axes*

Before testing our algorithm using our physical drone, we opted to first our drone in a simulated environment. Our simulations performed as we’d expect, following the desired trajectory perfectly.

We then decided to try our algorithm on the drone itself. We first tried a practice run using the practice waypoints. Our drone ended up flying too high, as there was a mistake in the Vicon Frame to drone frame transformation, which resulted in the vertical distance being added twice and as such made our drone fly an extra 1.5m higher than it should have.

As stated earlier, we compute the frame transformation between the drone and the world frame by first hovering, and once we are satisfied with the hover and receive the TEST command, we take the last published Vicon frame, and then compute the transformation between the drone’s frame (based on our local pose) and the world frame. This method has its drawbacks, as it is possible for the Vicon frame to be misaligned with the current frame being used. Using interpolation or timestamp matching could have reduced the error we have experienced.

After fixing the transformation, we opted to undertake the Vicon-less test, which resulted in our drone following a smooth trajectory and reaching all 7 waypoints. See the above two graphs for our drone's trajectory. We were able to conclude that our drone has met all of our objectives for this challenge task.

4.3 Challenge Task 4

4.3.1 Challenge Description

Similar to Challenge Task 3, this task aims to have the drone navigate a set of waypoints sequentially, but with obstacles placed at arbitrary locations in the environment. The obstacles added to the environment are four yellow cylinders with a diameter of 30 cm. The positions of these obstacles are unknown prior to flying. Additionally, these obstacles have colored letters painted on them, with red letters denoting that the drone should fly to the left of the obstacle (clockwise) and green denoting that the drone should fly to the right of the obstacle (counter-clockwise). The red obstacles also have black bands. Finally, the direction constraint only needs to be followed if the drone is within a 1.0 m radius from the center of the obstacle.

4.3.2. Objectives and Task Functional Analysis Metrics

O3.1: The drone must visit every waypoint sequentially

- **M3.1.1:** Within 35 cm of a waypoint when using both on-board sensors and Vicon data.
  - **Criteria:** Drone must be able to reach a waypoint using either on-board sensors and Vicon Data.
- **M3.1.2:** Closest distance to each waypoint after a trajectory.
  - **Criteria:** The drone must be within 35cm of the center of each waypoint.
- **M3.1.3**: Number of waypoints traversed in order
  - **Criteria:** All 7 waypoints must be traversed in order.

O3.2: The drone should navigate through all the waypoints quickly

- **M3.2.1:** Time to traverse all waypoints.
  - **Criteria:** The drone must navigate through all waypoints in under 75 seconds
  - **Constraint:** This must be accomplished under 135 seconds.

O3.3: The drone should avoid obstacles autonomously

- **M3.3.1**: Number of collisions per flight
  - **Constraint:** There should be no collisions in the flight

O3.4: The drone should respect the direction denoted by an obstacle

- **M3.4.1**: Number of violations per flight
  - **Constraint:** There should be no violations.

4.3.3. Challenge Task 4 Design Overview

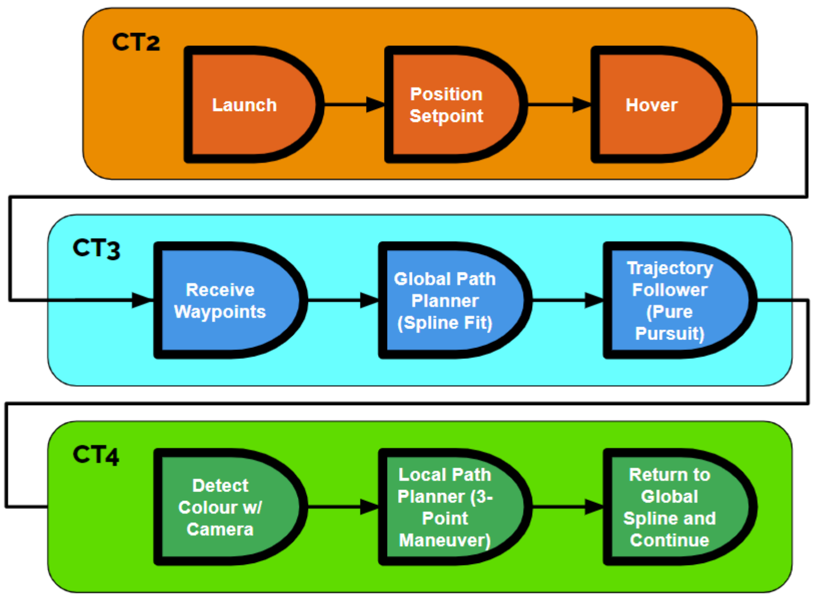
CT2
Launch
Position Setpoint
Hover
CT3
Receive Waypoints
Global Path Planner (Spline Fit)
Trajectory Follower (Pure Pursuit)
CT4
Detect Colour w/ Camera
Local Path Planner (3-Point Maneuver)
Return to Global Spline and Continue

Figure 8: *Our Challenge Task 4 pipeline depicting how we build on top of our previous pipelines*

The pipeline builds from our Challenge Task 3 pipeline by adding obstacle avoidance. While navigating, we leverage the color camera to detect the yellow obstacles, and use the diameter of the obstacle to estimate the distance we to the object.

Our obstacle detection algorithm runs in its own node, called ColorDetector. Our algorithm to detect if we are in range of the yellow obstacles can be summarized as:

1. Receive an image from the camera node.
2. Undistort the image and convert it to the HSV color-space.
3. Perform a histogram equalization on the Saturation Channel of the HSV image and recombine the image with the new S channel.
4. Perform an in-range threshold to keep pixels that are between particular H, S and V values. This forms a binary mask consisting of yellow pixels only.
5. Get the contours of the binary mask and search for the largest contour (by area). During search, we filter candidate contours by ensuring that they are greater than a minimum pixel width, and that it has a larger width than it does height.
6. If there are no contours that are valid, then return no detected obstacle.
7. Otherwise, find the minimum area rectangle of the contour, and find the maximum width of it. This will be the disparity value.
8. Calculate the depth of the obstacle using the focal length of the camera, the diameter of the object (30 cm) and the disparity from before.
9. If computed depth is less than some distance threshold (e.g. 1.25 m), then return that we are not within range of the obstacle. Otherwise, perform direction detection.

This algorithm returns either True, we are within 1.25 m of the object, or False, we are not. We chose to undistort the images we received to remove any imperfections within the image, and ensure that our depth estimation can be as accurate as it can be. We leveraged the HSV color space for yellow detection because we found that it was easier to tune [3], as "similar-looking" colors have similar hue values, which is unlike the RGB color space. By performing histogram equalization on the saturation channel, it was then easier to filter out grays in the background that can be confused with yellow when the image is dark, and ensures that our tuning works in a wider range of lighting conditions. We chose to flag an obstacle being detected at 1.25m rather than 1m so that our planning algorithm can have a bit more space to plan around the obstacle.

If we are within 1.25 m, we perform direction detection in the ColorDetector node. The algorithm to perform direction detection can be summarized as:

1. The image should still be in the HSV color space, with the equalized Saturation channel.
2. Use the yellow mask from the previous step and all the detected contours. Fill the detected contours using a flood fill algorithm to create a new yellow mask that has all contours filled. The minimum area rectangle from the previous step should also be available.
3. Use the minimum area rectangle from the previous step to crop the image. This is done leaving the height of the image alone, but cropping the image so it was between the smallest width and largest width of the minimum area rectangle of the largest contour. We also add some padding (of 10 pixels) to ensure that underestimations don't affect the crop.
4. Use the filled yellow mask from step 2 to mask out anything detected in the background that is not the detected obstacle. This will form a new HSV image that is cropped and filtered to contain only pixels of the obstacle
5. Within the filtered HSV image, detect green pixels by detecting any pixels that are in-range between particular H, S and V pixels
6. Perform a morphological operation of opening to remove any noisy pixel detections.
7. Sum all the pixels within the green mask and ensure it exceeds a defined threshold. If it exceeds the threshold, then consider the direction as "RIGHT", otherwise define it as "LEFT"

The algorithm this time defines the direction that the drone should aim to traverse. The algorithm logic is largely the same as yellow direction, however we chose to filter and crop the HSV image to ensure that all we have within the image is the obstacle of interest. Our cropping method was selected because we noticed edge cases with an homography approach, so we chose to keep things simple and instead crop a bounding box around the obstacle of interest (with some padding), which we found to work more reliably.

For color detection, we tried a few different approaches, including band detection, contour based detection of green vs red and green-detection. We found that both of the first two methods had failure cases. For band detection, we found it unreliable to detect the bands since across all viewpoints, the bands may not be straight horizontal lines. Additionally, sometimes, we'd detect aspects of the background as an obstacle. During such cases, using band detection would tell us to go "RIGHT", rather than give us a way to detect false positives. As a result of this, we decided to scrap band detection and aim to purely use color detection.

As for the alternative, contour detection based on green vs red, we found that we would often have parts of the orange Myhal chairs in the background that would accidentally be detected as being red letters.

During our testing, we found that green detection offered much more reliable results, likely due to how distinct green is in the environment. Introducing morphological-based filtering made the algorithm even more robust from false-positives. Given that we aim to detect letters and obstacles further away from the obstacle itself, the letters detected were small in the image. As such, we decided to threshold based on the amount of pixels in the green mask rather than using contour-based detection.

Once our color detection outputs a direction, our obstacle avoidance algorithm, which is subscribed to the direction topic, receives the course of action to avoid the obstacle. We use a rolling window of actions as a consensus mechanism to decide on our course of action. If 10 consecutive messages from the direction topic tell us to avoid the obstacle by moving in one direction, we take that action.

After coming to a consensus about the presence of the obstacle we then create 2 additional waypoints to avoid the obstacle while complying to the direction constraints. These waypoints are then added to the start of our waypoint list. After passing these new waypoints, we then try to reach the nearest waypoint from our original trajectory.

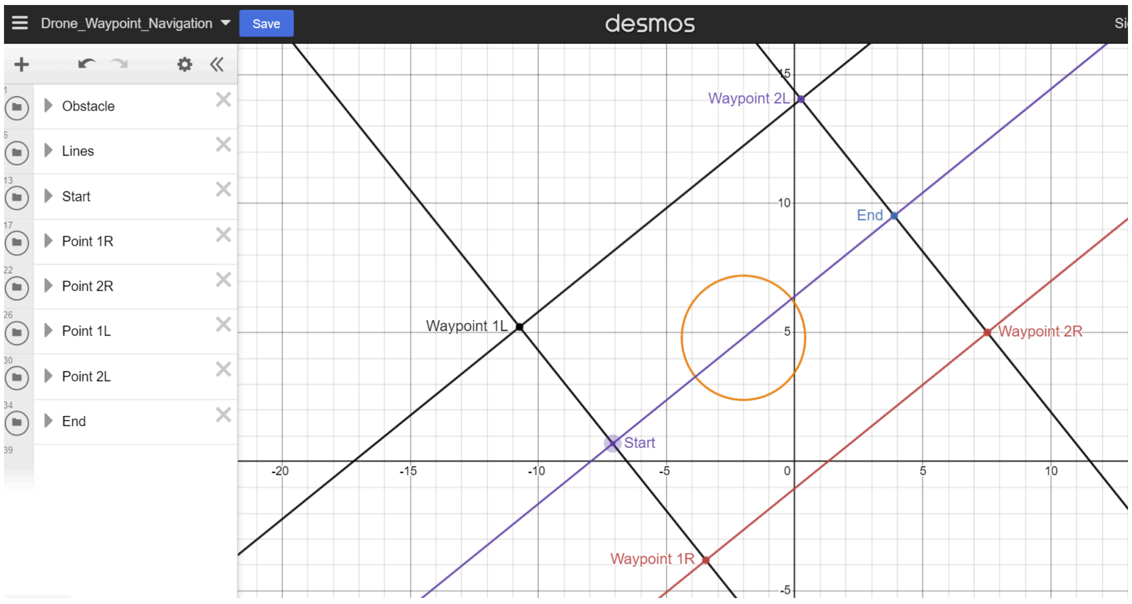

Figure 9: *Obstacle Avoidance algorithm simulated in Desmos*

4.3.4. Challenge Task 4 Results

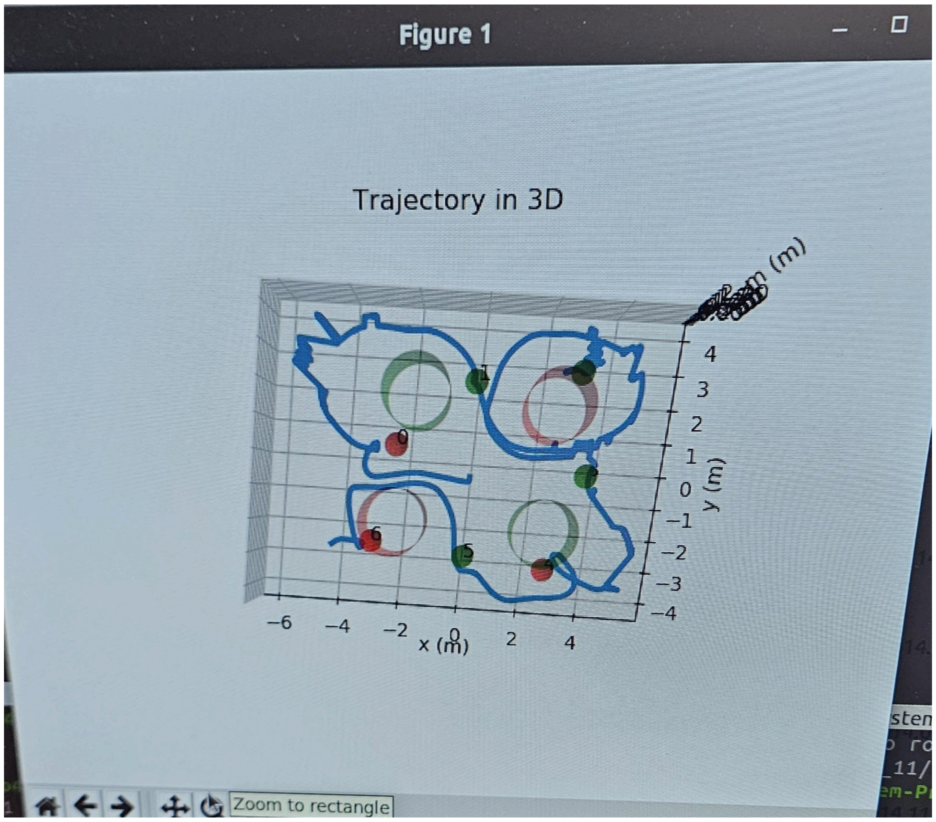

Figure 10: *Top: Picture of our drone accomplishing CT4. Bottom: Trajectory of our drone when testing CT4.*

To test our solution's effectiveness, we first created a simulated environment, including the obstacles. We created a virtual 2D lidar sensor that can scan the environment and detect the presence and location of obstacles. With this setup, we were able to test if our algorithmically-generated waypoints to avoid the obstacles gave the drone enough clearance to safely avoid the obstacle.

To test our obstacle detection algorithms, we first calibrated our camera using openCV's camera calibration functions. Using it, we were able to get the camera's intrinsics matrix and distortion coefficients, enough to enable us to calibrate our cameras.

We then manually flew our drone in the flight arena near the obstacles and recorded footage of our drone near obstacles. Using these videos, we tested our code offboard, to evaluate the drone's behavior. See Figure 11 for a sample of our processing pipeline. After tuning the HSV thresholds for yellow and green, we were ready to test.

When we performed our test, we managed to make it to 4 of the 7 waypoints without colliding. 2 of the waypoints were almost reached, but we believe they were missed due to the drift of the visual odometry. Given how close the last waypoint was to the obstacle, our drone ended up colliding after completing the task, but was thankfully not damaged. This means that this design has failed objective **O3.1**. See Figure 10 for the results. Given the time constraints we were operating under, we did not have the time to test again.

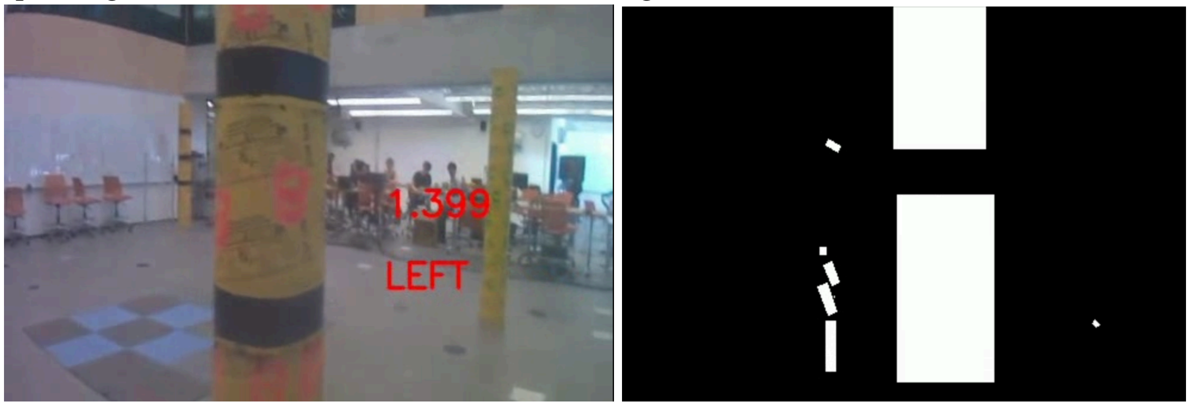

Figure 11: *Object recognition and processing of a camera frame. Left picture is the frame annotated with the prescribed direction command, along with the estimated distance to the obstacle in meters. Right is the obstacle mask.*

4.4 Challenge Task 5: Bonus

4.4.1 Challenge Description

This bonus Challenge Task's aim is to decode a sign in the Yautja language. There are 4 numbers written on the board that reveal a location. We are provided with a conversion table to convert from these Yautja numbers to arabic numerals. This task must be completed using the drone's jetson and camera. Upon completing the decoding, the message must be published to a ROS topic, a message of type std_msgs/Int16MultiArray.

4.4.2. Objectives and Task Functional Analysis Metrics

O4.1: The algorithm should be able to detect the Yautja numbers and parse them

- **M4.4.1**: Algorithm can identify the numbers from the background
    - **Criteria:** Should be able to identify numbers from the environment

- **M4.4.2**: Algorithm can parse numbers correctly
    - **Criteria:** Algorithm should not mix numbers up

4.4.3. Challenge Task 5 Design Overview

Our simple algorithm to achieving letter detection is as follows:

1. Grayscale image and undistort it using camera matrix.
2. Threshold the background to only include out the "paper region", which defines the area of the four pieces of paper the letters are printed on.
3. Determine a bounding box to identify the four points that define the paper region.
4. Use a homography matrix to transform and crop the rectangular area of the four points so that the resulting image is horizontal.
5. Divide the homography image into 4 even regions.
6. Determine the number of horizontal and vertical lines there are in each segment, which is enough to uniquely identify each of the letters.

4.4.4. Challenge Task 5 Results

We were successful in completing the challenge task. One issue that we had faced was that our thresholding method was not working well when the letters were against a yellow board. The reason is because during thresholding, it was grouping the white and the yellow together and it wasn't able to differentiate the paper from the board. More tuning with the thresholding was necessary to fix this problem. See Figures 12 and 13 for the images taken during the tests.

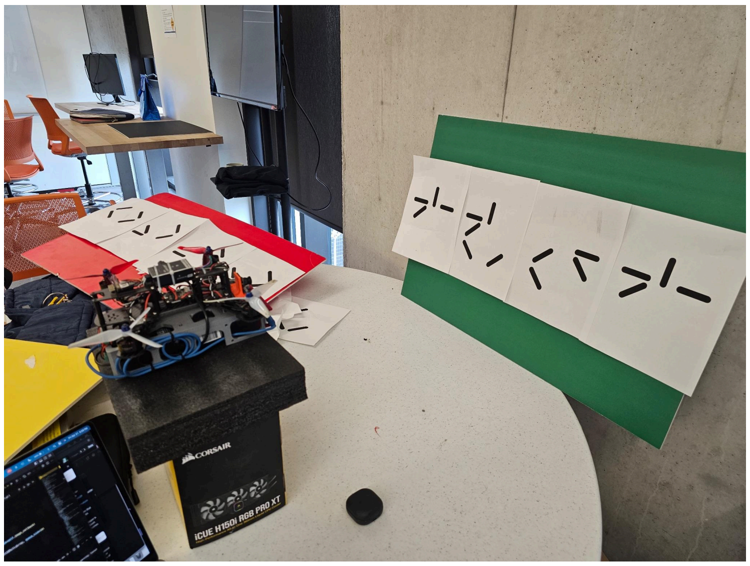

Figure 12: *Picture of our drone in front of the board it is trying to decipher.*

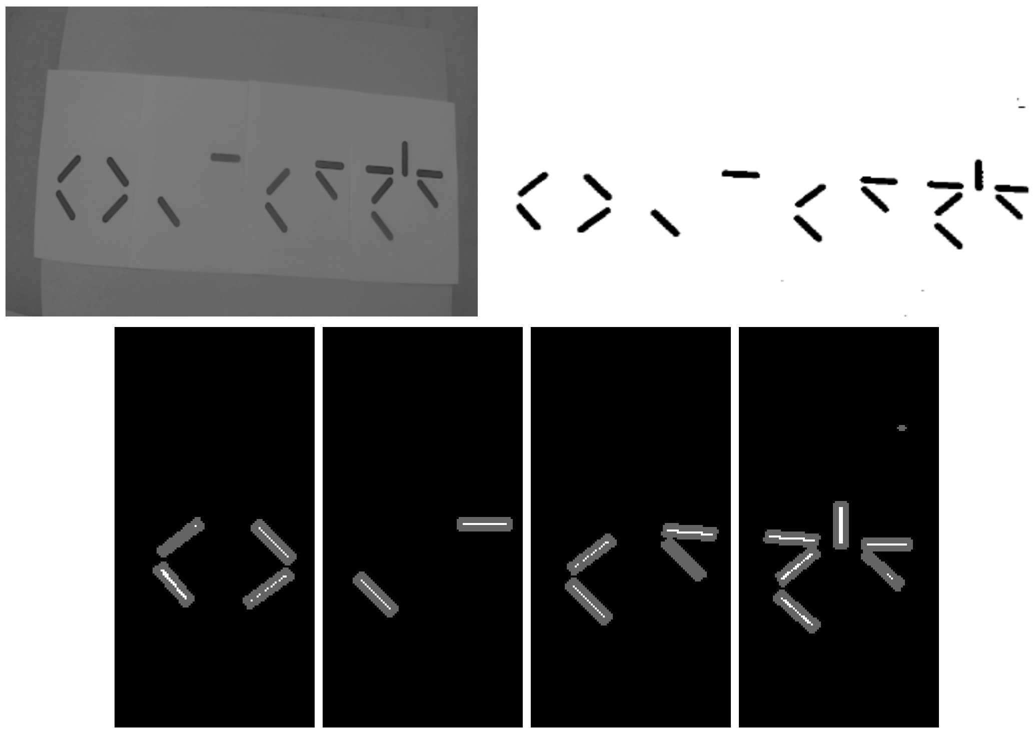

Figure 13: *From top left, right and bottom: Processing of an image frame, from the initial picture of the board, the numbers isolated from the picture and the individually parsed Yautja numbers.*

5 Lessons We Learned

Throughout this course, we gained invaluable insights into engineering design, autonomous drone navigation, and obstacle avoidance. Despite the knowledge acquired, we also faced numerous challenges. This section discusses the challenges we encountered, to serve as feedback for the course. We also reflect on the things we wish we had done differently and provide advice for future students to learn from our experiences.

5.1 Technical Related Challenges

During this course, our team faced several technical challenges. We were able to resolve most of them, thanks to the support of the teaching team. The table below summarizes the problems we encountered and the solutions we suggest to address them.

Issue	Description	Suggested Solutions

Testing Times	Our team faced significant wait times for testing our drone. This issue was common among all teams and likely impeded controller tuning. At times, we had to wait up to 3-4 hours for just 20 minutes of flight time.	1. Set up a simulation for the entire course to enable testing in a simulated environment. 2. If possible, establish an additional testing room without Vicon for teams to collect data or debug drone issues.
Jetson Overheating	Our Jetson would often overheat, even during operation, causing system lag and poor performance. In some cases, this led to the drone behaving unexpectedly, such as flying uncontrollably towards the ceiling.	Provide teams with fans to ensure the Jetson remains cool during operation.
Difficulty setting up Vicon	We had trouble getting the Vicon to work with our drone, and due to time constraints, we couldn't spend enough time debugging the issue. Consequently, all our flights did not utilize Vicon data.	Offer a tutorial for teams on how to use Vicon data through qgroundcontrol/MAVROS instead of the Realsense T265.
Transformation between Vicon and drone's local pose	As mentioned in our Challenge Task 3 solution evaluation, we faced difficulties in transforming between the Vicon pose and the drone's local pose. Testing these transformations without actually flying was challenging.	Set up the ROS tf tree to automate this transformation for teams or provide a library for automating this transformation. This would allow teams to focus on controller tuning rather than dealing with transformation errors.

Not enough test trials	During a test run, hardware issues could arise, and there shouldn't be a hard limit on the number of trials a team has. Limiting trials may penalize teams for random hardware failures.	Avoid imposing a limit on testing trials, which would promote rapid testing/prototyping and reduce stress associated with testing.
Strict marking for missing waypoints	We found the penalty for missing a waypoint to be a bit unfair, especially when chance played a significant role in the drone's success in reaching waypoints.	Implement a decaying function-based scoring system, allowing teams to receive partial points for almost reaching waypoints.
No ability to 'unit test' functionality	The course marks were largely based on the integration of multiple complex components, making it difficult to test functionalities separately.	Break down testing into smaller, more achievable milestones, allowing teams to receive marks for successful individual components.

5.2 Collaboration Related Challenges

During the course, our team also ran into some collaboration issues. Once again, we were able to resolve most of them. The table below summarizes the collaboration issues we ran into and our recommendations for solutions.

Issue	**Description**	**Suggested Solutions**
Difficult to Parallelize Tasks	Due to the nature of how the challenge tasks were released, it made it very difficult to parallelize tasks.	We recommend releasing all the challenge tasks at the start of the course, rather than in a rolling fashion. This way teams can better plan what to do ahead of time.

Disagreements on approach	Different team members sometimes had conflicting ideas about how to approach a problem due to not knowing what would work the best, which led to delays in decision-making.	Reach a consensus on the best approach, and be willing to adapt if a chosen approach proves ineffective. Have TAs review team's approaches and provide relevant feedback which might help them be more confident towards their approach.
Uneven workload distribution	Some team members had to take on more tasks than others, due to difficulty in accurately assessing how hard the tasks were (both underestimation and overestimation)	Regularly assess the workload distribution within the team and make adjustments as needed to ensure a more equitable distribution of responsibilities. Seek feedback from TAs to get a better sense of how much time should be allocated to a given task
Inefficient meetings	Some meetings were unproductive, with discussions veering off-topic or team members unprepared to contribute meaningfully.	Establish clear agendas for meetings, and ensure all team members come prepared to discuss their assigned tasks and any issues they have encountered.
Commute	Our team consisted largely of commuting, who lived fairly far away from campus. This made working with hardware (e.g. testing, etc).	We were able to partially mitigate issues by leveraging simulation, but we believe that having more spaces open for testing, or having a strict flight schedule defined the day before a team goes in would be useful so that commuters can better plan whether they should come to the flight arena or not.

By addressing these collaboration-related challenges, future teams can work more effectively together and achieve better outcomes in the course.

5.3 Things We Wish We Did Differently

- **Better time management:** Allocating sufficient time for debugging and testing, especially for complex tasks that require the integration of multiple components.
- **Improved communication:** Ensuring regular team meetings and updates to keep everyone informed about the progress and potential issues.
- **Early focus on controller tuning:** We would have benefited from spending more time on controller tuning early in the course, as this could have significantly improved our drone's overall performance.
- **More emphasis on simulations:** We should have used simulations more extensively to test and refine our algorithms, reducing the need for physical testing time.
- **Encourage cross-functional understanding:** Ensuring all team members had a basic understanding of each other's work, which would have fostered better collaboration and reduced bottlenecks.

5.4 Advice for Future Generations

- **Start early:** Begin working on the course projects as soon as possible to ensure adequate time for testing, debugging, and refining your solutions.
- **Utilize simulations:** Make the most of simulations to test and iterate on your designs, reducing the need for physical testing time.
- **Prioritize controller tuning:** Spend sufficient time on controller tuning early in the course, as this can have a significant impact on your drone's performance.
- **Collaborate effectively:** Foster an environment of open communication and collaboration within your team. This will help you address issues more effectively and learn from each other.
- **Learn from previous teams:** Review the experiences and lessons learned from previous teams, as this can provide valuable insights and help you avoid similar pitfalls.

6 References

[1] Norman, D. A. (2013). The Design of Everyday Things: Revised and Expanded Edition. Basic Books

[2] Quengineering. “Jetson-Nano-Ubuntu-20-image”. https://github.com/Qengineering/Jetson-Nano-Ubuntu-20-image. GitHub. Accessed April 22, 2023

[3] Clockmaker LLC. “HSV Color Range Thresholding”. https://learncodebygaming.com/blog/hsv-color-range-thresholding. Modified. July 14, 2020. Accessed April 22, 2023