Mech 464: Project Report
Bitcraze Drone Platform Tracking

Group 10:

John Matheson – 97462337

Ethan Alexander – 84207034

Kristoffer Klingenberg – 23443260

Willem Van Dam – 33500646

1 Abstract:

The problem that we are attempting to solve in our project is to program a drone to match the unknown movements of a landing platform. The drone would use a combination of onboard sensors along with the lighthouse data to track the platform. A second drone was used as the moving "platform" with known position. A useful application for this would be the landing of a helicopter on a boat in rough water. The Bitcraze development suite was used in the project with their Crazyflie drones and IR locator lighthouses. In the project, we implemented our own control algorithm for the follower drone. We installed the lighthouses. We designed our own drone and boundary collision avoidance functions. To improve the performance of the follower drone, we also implemented a position prediction algorithm that used the acceleration and velocity of the tracked drone to estimate its future position. To measure the performance of the follower drone, we compared the desired and actual position of the follower drone. The smaller the difference between these two values the better.

The follower drone was tested in 4 different scenarios, step, step + prediction, ramp, ramp + prediction. In the step response test, the tracked drone is flown to a fixed point in space and the follower drone was commanded to move to the desired position. This test was used to see what the "steady state" deviation is to use as a reference, and if our position-prediction function causes any changes in the steady state stability. In the ramp response test the tracked drone follows a circular path while rotating. What we found was that for a step response, the follower drone was able to settle within 1 second and for the ramp response, the follower accuracy oscillated between 0.1m and 0.2m. We found that our position prediction algorithm reduced the error in the ramp response but increased the steady state error in the step response. There are several ways that the performance could be improved that we lacked the time to implement, including PID optimization and implementing a stiffness controller instead of the position prediction algorithm.

2 Contents:

3 Introduction:

The Bitcraze Crazyflie drone is an easily programmable quadcopter. The problem that we are attempting to solve in our project is to program a drone to match the unknown movements of a landing platform. The drone would use a combination of onboard sensors along with the lighthouse data to track the platform. A second drone was used as the moving "platform" with known position, so a platform did not need to be constructed. A useful application for this would be the landing of a helicopter on a boat in rough water. We will be measuring the performance of the tracking by monitoring the error between the desired position of the tracker and the actual position of the tracker over the course of several tests.

4 Methodology:

4.1 Apparatus

The physical apparatus we used consisted of a shroud (mosquito net) and carpet to protect ourselves and the drone from damage; 2 Crazyflie drones equipped with lighthouse sensor decks; 2 Bitcraze lighthouses; and a computer to run the python program that controlled the drones.

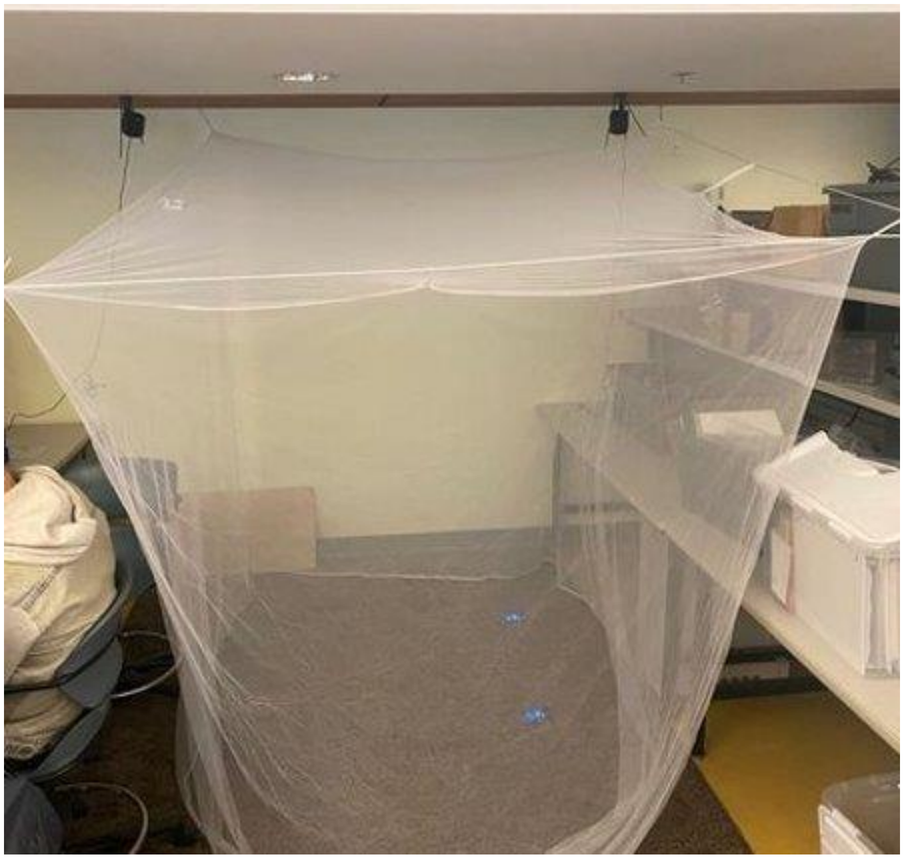


Figure 1: Physical Apparatus

4.2 Robot Layout

To relate the tracking drone to the following drone we treated the tracking drone as a parallel manipulator that can move in X, Y, Z and θ independently. We then attached the following drone through a serial arm that is offset in +k and -i from the tracking drone. From this robot layout we were able to calculate the forward kinematics, The Jacobian and J_dot. Using this allowed us to calculate the target velocity and acceleration of the following drone which improved our control. The figure below shows the location of the following drone relative to the tracking drone.

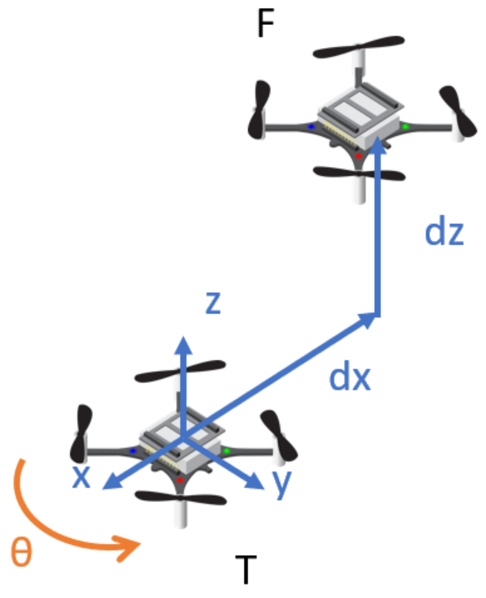


Figure 2: Robot Manipulator Layout

4.3 Control Algorithm

After initializing the Following and Tracking drone the code loops through a set of instructions in chronological steps.

Step 1: The Current position, velocity and acceleration of the tracking drone and the position of the following drone are recorded by the lighthouse and sent to the computer and the respective drones.

Step 2: Using the current position and movement data from the Following and Tracked drone, we calculate the desired position, velocities and accelerations and predict where the desired position is going to be at in 0.3 seconds time. In our tests, the desired position of the following drone was a normal distance of 0.4m from the tracked drone, and 30° above the tracked drone.

Step 3: For the following drone, the vector (desired position - current position) and rotation (yaw) required to achieve the desired position is calculated.

Step 3.5: The movement-vector is checked against the boundaries, both cage-edges and tracking-drone safety-cylinder. (A cylinder was chosen to avoid downdraft-problems). If there is interaction, either shorten the vector to end at the edge, or create a flightpath around the Tracked drone to get to the desired position.

Step 4: The desired position for each drone (end of the vector with origin in the center for the following drone) gets sent back to the drones and they move to those positions (or as close to it as they can before a new iteration begins).

4.4 Implementation

To successfully track the drone, we wanted to avoid collisions between drones, avoid collisions between the tracking drone and the walls, and to increase the accuracy of the tracking by using a prediction algorithm. This section will describe how we implemented each of these solutions.

4.4.1 Drone Avoidance

To avoid mid-air collisions between the tracked drone and the follower drone, a cylinder of exclusion was created around the tracked drone. The algorithm is listed below and references figure X which visually represents the geometry of the solution.

1. If $r_1 > ||\underline{a}||$, F moves along $\underline{a}$ away from T
2. Else If $\phi < \theta$ and $\underline{w} < \underline{d}$, move to point A
3. Else F moves along $\underline{d}$ to F'

Where,

- r_1 is the radius of the exclusion zone. In our case 0.2m.
- r_2 is the targeted radius of the follower drone. In our case 0.4m.
- $\underline{d}$ is the vector between the current position of the follower drone and the desired position of the follower drone.
- $\underline{a}$ is the vector between the current position of the follower drone and the tracked drone.
- A is a vector from the tracked drone to the targeted radius that is orthogonal to vector d.

The cylinder of exclusion has an infinite height. During initial testing, we used an exclusion sphere instead of cylinder. Occasionally, the follower drone would pass above or below the tracked drone. The effects of the downwash caused the lower drone to destabilize and often crash.

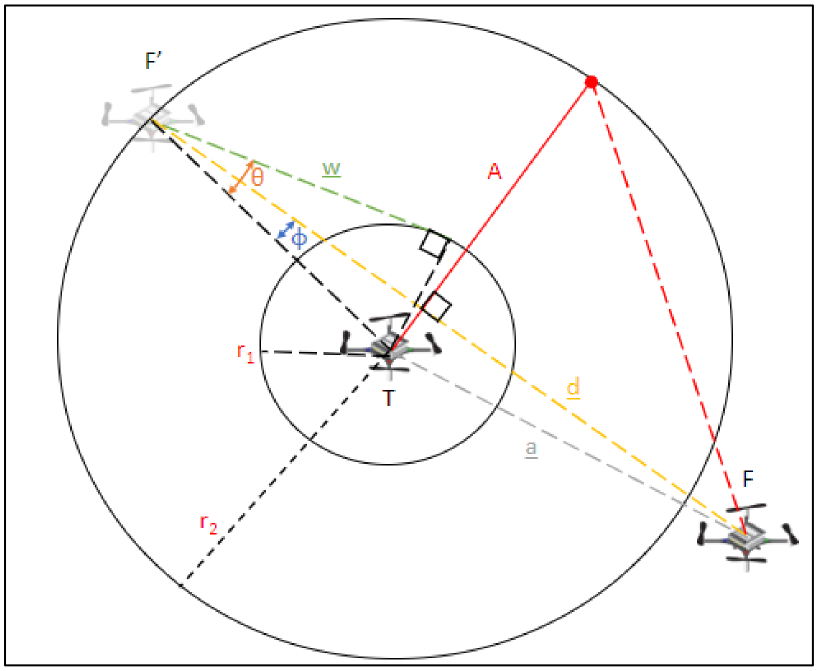


Figure 3: Drone Avoidance Diagram

4.4.2 Position Prediction

To increase the accuracy of the follower drone, we implemented a position prediction algorithm. The algorithm uses the first and second derivates of position to predict the final position of the follower drone (F'). The prediction is used to compensate for the lag of program processing and delay in the communication between the drone and the computer. Another benefit of this improvement method is that it avoids the use of low-level firmware modification. We did some research into changing the PID values of the drone. We also investigated optimizing the communication protocol of the drone. Both avenues seemed very time consuming, so we avoided them. The amount of time to predict ahead was based on experimentation. We found that predicting the movement 0.4 seconds into the future was optimal. Going further into the future would destabilize the drone, less time would cause the follower to be less accurate. The figure below shows the block diagram for how we implemented this control method K_v=.4 and K_a=.16, which is the position .4 seconds in the future.

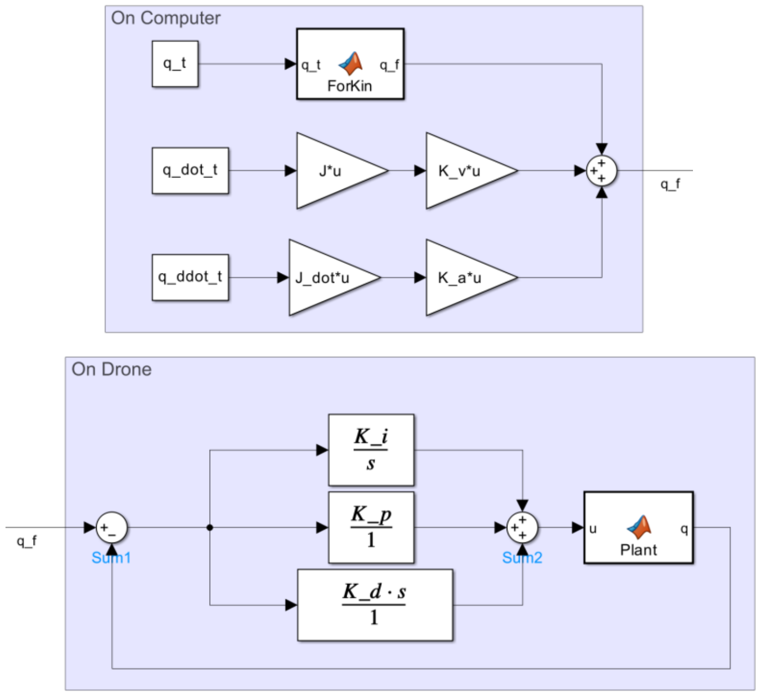


Figure 4: Flow Diagram of Control Algorithm, drone algorithm created by Bitcraze, computer algorithm created by team

The figure below shows the geometry of the position prediction. The blue and yellow arrows represent the movement prediction based on the acceleration, velocity, and rotational velocity of the tracked drone. The follower drone's current position is F and desired position is F'. The tracked drone (T) defines the coordinate system of the drones.

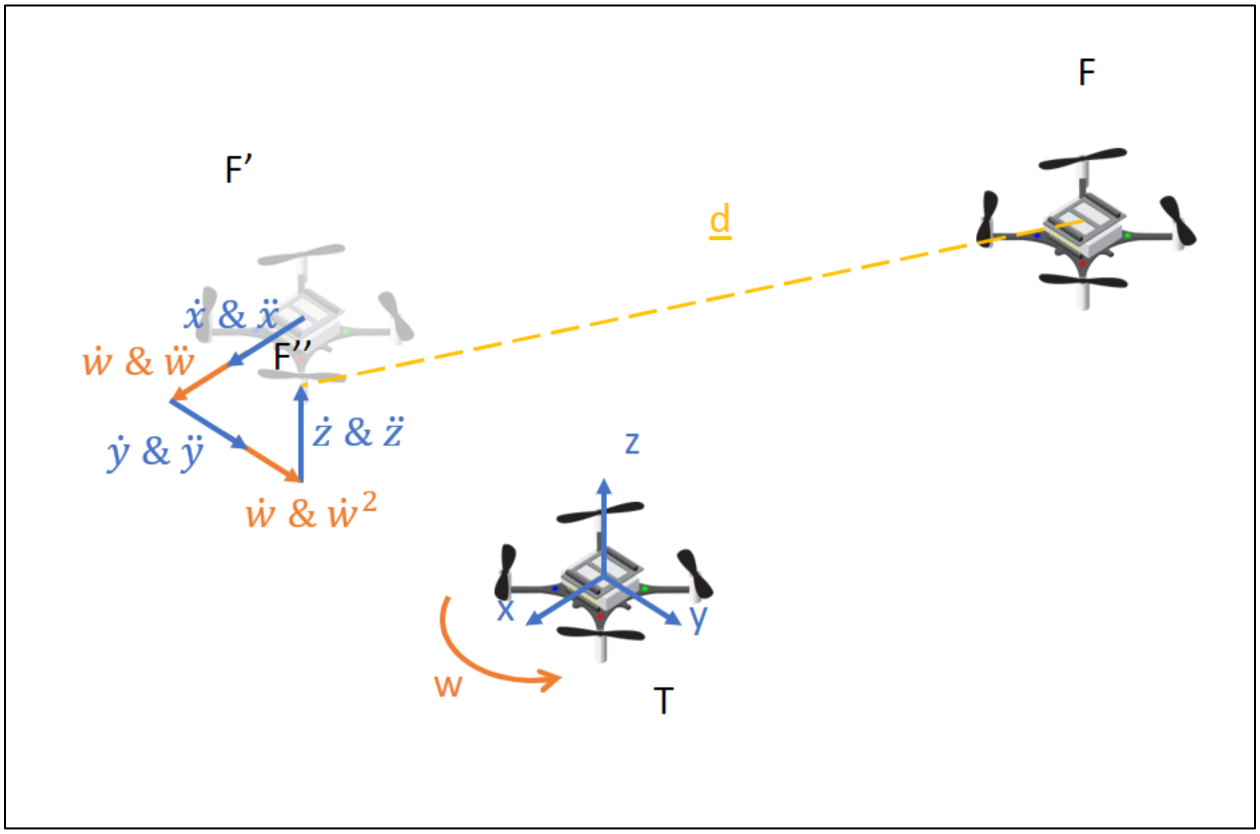


Figure 5: Follower Drone Position Prediction

5 Results:

We have stored the demonstration videos of the tracking drone functioning in different test cases on a members google drive. The videos are viewable by anyone with this link, and will be referenced in this section.

To measure the performance of our tracking software, we recorded the distance between desired position and actual position of the tracking drone during a step response test and a ramp response test. The difference between desired and actual X, Y, Z, θ, and norm(X, Y, Z) were all recorded. The test was first done without the position prediction software, then with it activated so we could measure its performance. In the step response test, the tracked drone is flown to a fixed point in space and the follower drone was commanded to move to the desired position. This test was used to see what the "steady state" deviation is to use as a reference, and if our position-prediction function causes any changes in the steady state stability. In the ramp response test the tracked drone follows a circular path while rotating (multiple times per circle), causing the follower drone to have significant angular acceleration to stay on its designated point. The motion of the drones during the ramp response test can be seen in the *Sinusoidal Motion* video.

Figures 5-10 show the results of our four test cases. The raw data for our tests can be found in CSV files in the project GitHub.

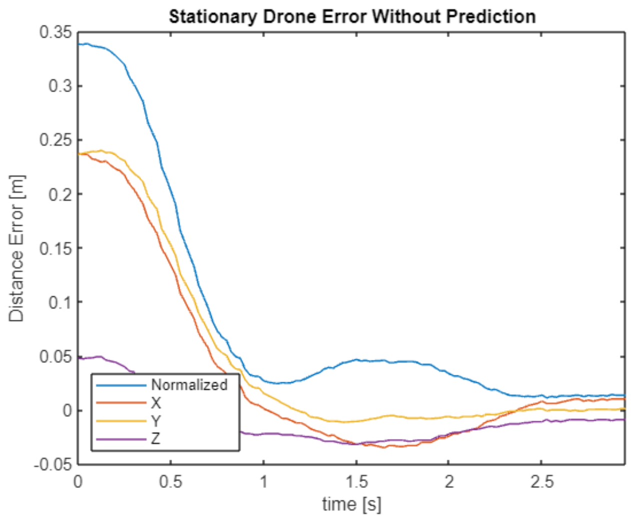


Figure 6: Performance of Tracking Program during Step Response without Prediction

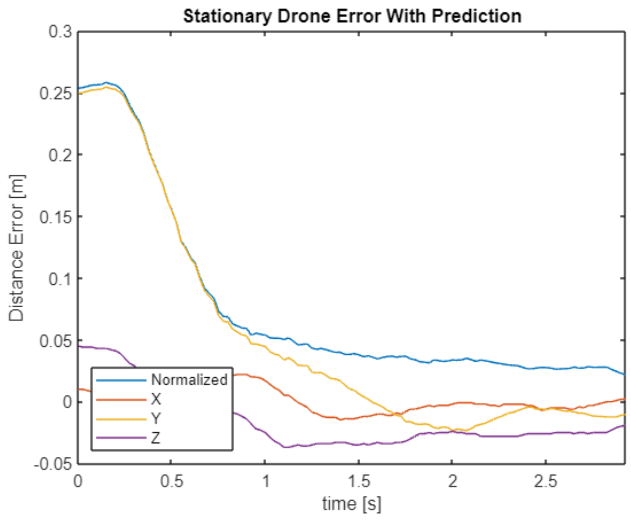
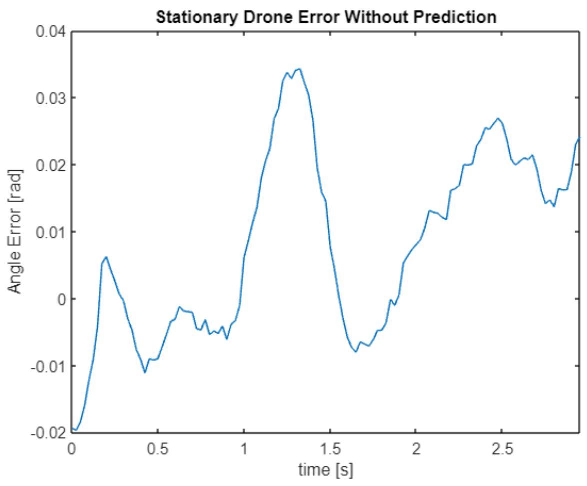


Figure 7: Performance of Tracking Program during Step Response with Prediction



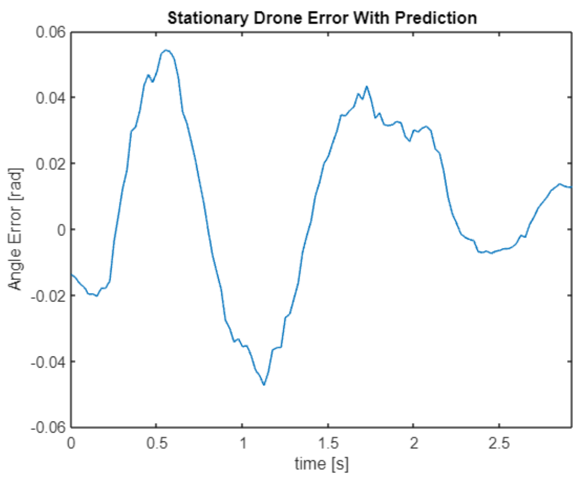


Figure 8: Performance (ϑ) of Tracking Drone during Step Response. Without Prediction (Left), with Prediction (Right)

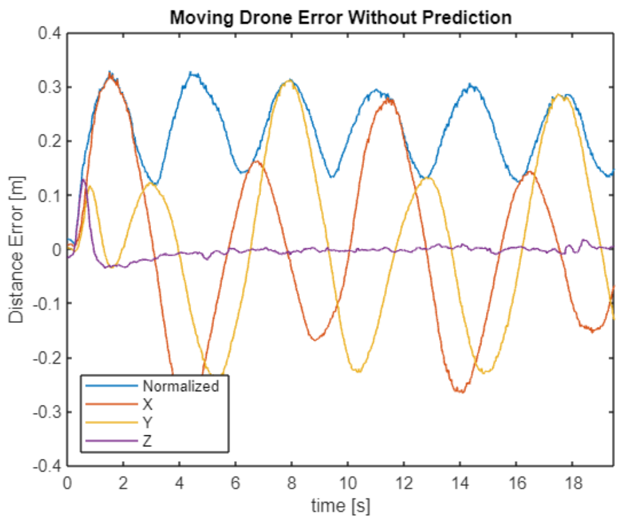


Figure 9: Performance of Tracking Program during Ramp Response without Prediction

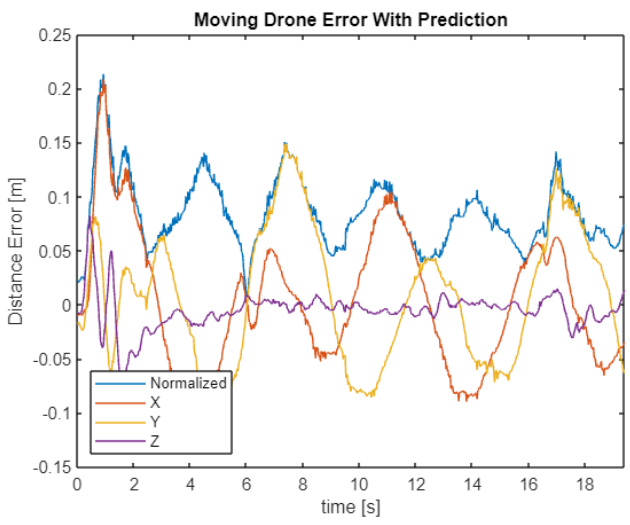
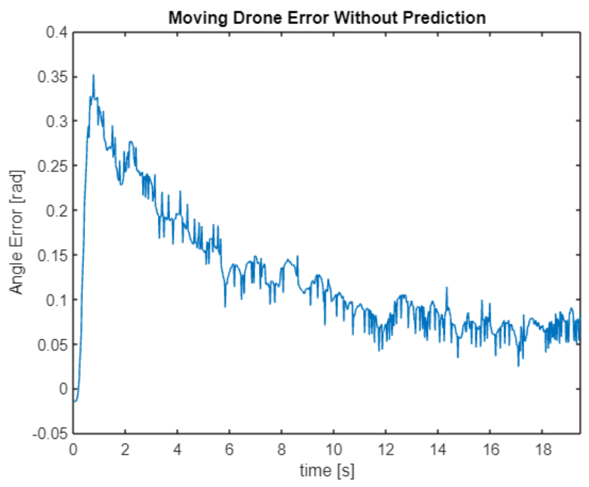


Figure 10: Performance of Tracking Drone (X,Y,Z) during Ramp Response with Prediction



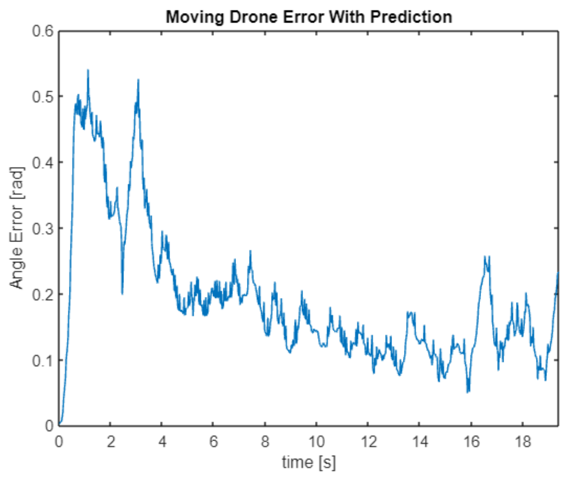


Figure 11: Performance (ϑ) of Tracking Drone during Ramp Response. Without Prediction (Left), with Prediction (Right)

A summary of the performance of the tracking drone during each test can be found in table 1. We will discuss our results in the discussion section.

Table 1: Summary of Test Results

Test Case	Prediction Time (s)	Linear Results (Norm) (m)	Angular Results (rad)
Moving	0.4	Oscillating between ~0.05 and 0.13 giving it an average error of ~0.09	Decreasing down from 0.5 and “stabilizes” at 0.15
Moving	0	Sinusoidal oscillation between 0.1 and 0.3 giving it an average of ~0.2	Decreasing from 0.35 to “stabilizing” between 0.1 and 0.05

Stationary	0.4	After initial movement (1 second), linearly decreases from 0.05 down to 0.025	Damped oscillation between 0.058 and -0.05
Stationary	0	After initial movement (1 second), goes from ~0.03 to ~0.02 (after an increase /deviation up to ~0.05)	(Ignoring starting position) variation between 0.035 and −0.01

6 Discussion and conclusions:

One achievement that we made was to track another moving drone instead of a moving platform as originally proposed. Tracking a moving drone means moving in three dimensions to track (including yaw angle) and being able to avoid the tracked drone effectively. One challenge that came from this is the air disturbance below each drone that disrupted the pathing of the drones. We prevented this by making a keep-out zone in a cylinder around the drone that was effective enough. This does mean that the following drone cannot position itself directly above the tracked drone. The only way this could be improved is to test the minimum height above a drone that does not disturb the drone below and change the keep-out cylinder height to this value.

Our second order position prediction method proved to be effective in improving the tracking accuracy. At all points it resulted in a reduction in the position error of the tracking and flying drone which proves our method is successful. The velocity and acceleration data from the drone is fairly noisy so a filter or averaging function could be used to reduce the instability of the following drone which we observed when we set the prediction value to a higher value

A potential improvement would be adding a stiffness controller on top of the prediction method to close the gap between the current and target positions. This would allow us to still compensate for the lag in the data transfer and reduce some of the errors in the drone's velocity, which is not part of the PID controller on the drone. The next step for this controller would be to modify the drone’s firmware so that a position and velocity setpoint can be implemented, allowing for higher tracking accuracy.

Another minor improvement would be to revise the code structure to be more pythonic (such as with classes) to make the code easier to build from for more complicated projects.

7 Statement of contribution:

John Matheson – 97462337	Most of the writing in the report, coded some of the cases for testing the follower drone, debugged the test cases.
Ethan Alexander – 84207034	Designed and programmed the feedforward controller designed to work with pure position programming. Setup the lighthouse

	and positioning system. Programmed drone communication backend.
Kristoffer Klingenberg – 23443260	Preliminary sources, coded some of the test cases, debugging, report writing
Willem Van Dam – 33500646	Coded some of the algorithms, debugged all algorithms and test cases, ran final tests for recording, report writing

8 References:

[1] Lecture notes ch. 3 to 7: https://canvas.ubc.ca/courses/109153/modules

[2] Bitcraze build and VM/firmware installation (includes Code examples): https://www.bitcraze.io/documentation/tutorials/getting-started-with-crazyflie-2-x/

[3] Bitcraze firmware installation: https://www.bitcraze.io/documentation/repository/crazyflie-clients-python/master/installation/install/

[4] General Bitcraze GitHub: https://github.com/bitcraze

[5] Project GitHub: https://github.com/walnka/bitcrazeDrone

[6] Project PowerPoint containing test-videos: Group 10 presentation.pptx

9 Appendices:

9.1 Appendix A: Bitcraze Start-up Guide

9.1.1 Overview

To control a Crazyflie drone autonomously, the position of the drone must be tracked with either the lighthouses or the flow deck. This document will only cover the use of lighthouses.

This guide includes a step-by-step guide to set up the Crazyflie drone, the lighthouses, and the programming of the drone. It will also include the issues that were solved by previous groups who used the Bitcraze drone.

9.1.2 Setting Up Lighthouses

Follow this tutorial:

https://www.bitcraze.io/documentation/tutorials/getting-started-with-lighthouse/

9.1.3 Safety Considerations

This section discusses the apparatus that was used for past teams. The drone should be operated within a shroud and above a soft-landing area. See figure A1 for the apparatus that was created in KAIS3080. Contact between moving propellers will most likely not damage the skin but will damage the eyes, if there is not a shroud between a user and the drone, eye protection should be worn.

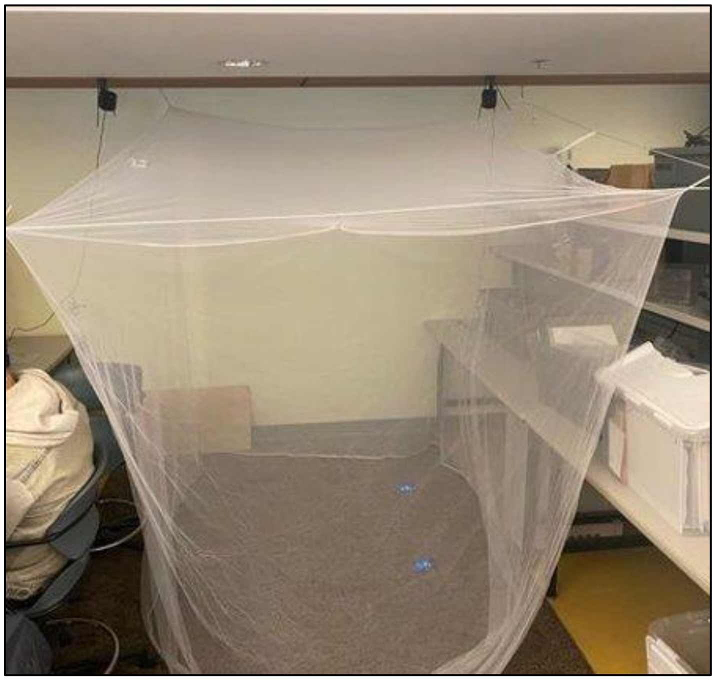


Figure A1: Drone Shroud and Carpet

9.1.4 About the Drone

This section consists of a list of compiled information about the drone.

- The battery of the drone will last for 10 minutes of continuous flight.
- The state of charge of the battery is shown on the drone as the duty cycle of the blinking blue light on the drone while it is charging. During charging, if both blue LEDs on the frame are solid, the battery is fully charged, if one led is off for the same time it is on, the battery is 50% charged.
- Any time the drone is turned upside down the motors will be disabled until a power cycle. This can also be used to allow for the drone to be tracked with the lighthouse without flying

9.1.5 Installing the VM & Connecting to the Drone using the Client

To begin, the VM should be installed onto your computer. Follow the link below for the instructions to set up the VM. After installing the VM it is best to reinstall the cfclient using the following commands “Pip3 uninstall cfclient”, “Pip3 uninstall cflib”, “Pip3 install cfclient”.
https://www.bitcraze.io/documentation/tutorials/getting-started-with-crazyflie-2-x/

9.1.6 Coding Guide

The link below leads to the GitHub project of a previous team that used the Bitcraze drone. Their project involved tracking one drone using another drone and used many of the available Bitcraze functions. The pathfinding&logging.py file is the one with all the useful information. Some good modifications to make to the got to command in the high-level commander is to remove the sleep command, this will make it a non blocking function so you can update the target point while its moving to another point. You can also modify this function to allow you to implement programing of yaw into the HLC by adding another input to it and passing it into the yaw. Included in our GitHub repo is a high-level commander function with these modifications. Make sure to not continuously send commands as the radio will not be able to keep up resulting in a backlog of commands and the drones not being responsive.

https://github.com/walnka/bitcrazeDrone