Internship Report

Ishir Roongta

Supervised by:
Soulaimane Berkane, UQO

Contents

Preface

This internship report is made for the project done in the summer of 2022. The internship was part of the MITACS Globalink Research Internship Program, and the project supervisor was Dr. Soulaimane Berkane of Université du Québec en Outaouais, Ottawa, Canada. Throughout the 12 weeks period of the internship, the intern was required to review literature, set up the testing hardware testing environment, develop new theories and work on MATLAB implementation in addition to regular interaction with the supervisor. Following is the compilation of the above-said work. On a personal level, this internship presented me with multiple challenging tasks, each of them being a great learning experience. This opportunity has been immensely helpful in my academic journey toward a career in Robotics.

I would genuinely like to thank my project supervisor for his constant support and guidance throughout the internship.

1 Introduction

With the rise of practical applications of multi-agent systems in areas like surveillance, disaster management, delivery systems, and warehouse management, one of the very interesting challenges to overcome is the dependency on the full range of information to localize the agents in an autonomous system. A full range of information is not always guaranteed in such scenarios during which the multi-agent system should adapt using cooperative localization techniques. This involves sharing information between the agents, much like a traditional autonomous full swarm.

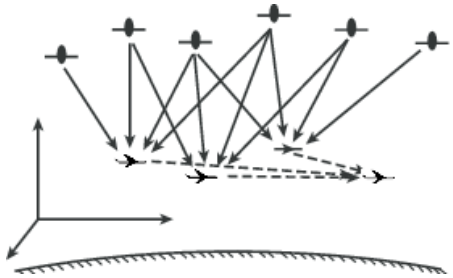

Figure 1: Cooperative Localization

This is precisely what we aim to explore in this project. We want to comment on the functionality of a multi-agent system (agents being crazyflie drones here) that has less than conventional information to localize each of the agents and must rely on sharing information to some level to make up for that missing measurement. We then want to determine the *local region* if any, in which this shared system would be viable.

With this background, the following internship objective was proposed:

Approaching the problem of co-operative localization in a multi-UAV system using less than the traditional amount of UWB modules.

---

2 Internship Timeline

2.1 Week 1-4

The tasks for the first four weeks of the internship were to familiarize myself with the crazyflie suite and go through the tutorials and documentation on the BitCraze website diligently. The problem of multi-UAV localization was discussed, and in order to have a precise testing environment for the multi-crazyflie setup, we experimented with the Lighthouse Positioning Deck and the Loco Positioning Deck of the Crazyflie system.

Parallelly literature review was done on topics such as *State Estimation* using shared information, *Luenberger Observer* for such cases, *Lyapunov Functions*, Extended Kalman Filters, Basics of Linear Algebra and Matrix Properties

2.2 Week 5-8

On the theory side, we worked on the observer design to check the viability of the system and to answer the question would the limited information present be sufficient to localize both agents. This was ensured through the Lyapunov Stability Criterion. More on this will be discussed in the relevant section.

On the hardware side of things, flight tests were conducted to log parameters like gyroscope data and UWB ranges distances from the anchors positioned at *known* locations. Python scripts were tested as well to fly a single and double crazyflie agent setup.

2.3 Week 9-12

The focus for the remaining period of the internship was to continue the design of our observer and simultaneously setup a MALTAB pipeline to test the Luenberger observer against the reference data from the Lighthouse Positioning system.

After completing the hardware setup for the testing environment, efforts were made to understand how information is shared inside the Crazyflie firmware as sharing information between the agents was required. Sample trajectories were taken to generate input logs for the Matlab implementation of the designed Luenber Observer.

Realizing the absence of a shared messages capability in the Crayflie firmware, the last week was spent gathering data and attempting to self-implement this feature.

3 Theoretical Setup

In the subsequent sections, we talk about the theory involved and the problem formulation for the project. Please consider the topics discussed below are in brief.

3.1 Background

Before moving on to the problem formulation, we discuss the requirement of *four UWB modules* or, in essence, *four measurements* to localize an agent in a *3D* space. The proof for this is trivial, and it can be seen geometrically that three spheres intersect in almost two distinct points in *3D* space, and a fourth sphere or a fourth measurement is required to attain a **unique point**.

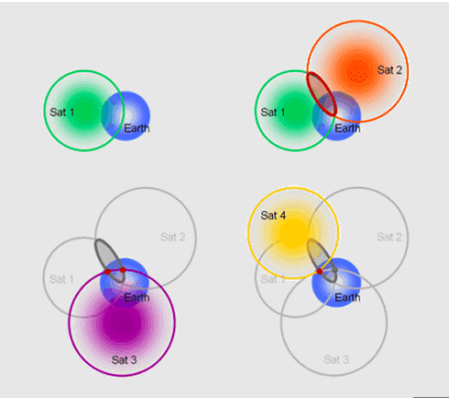

Figure 2: Working of a traditional GPS

This can also be understood in terms of position and time. In a traditional *GPS*, measurements are taken from four different satellites to calculate the four variables which are: *x, y, z, time*.

3.2 Problem Formulation

We start with the simplest form of a multi-agent system. Consider a two-crazyflie setup. Now each of these would require four UWB anchors to localize themselves, but in our setup, we will consider *only three anchors* in totality. Further, among these three anchors, one would be *unique* to each of the crazyflie and the other

would be *shared*. **Note**: Here, *unique* would mean that the range measurement from that anchor to a crazyflie would be available to that specific crazyflie **only**. While *shared* means range measurement for both the crazyflies is present for this anchor.

Below is a figure for reference of the entire setup:

- - a_i is the anchor with i being the id number.
- - p_i is the pose of i^{th} crazyflie.
- - d_{ij} is the range measurement between the i^{th} drone and the j^{th} anchor.

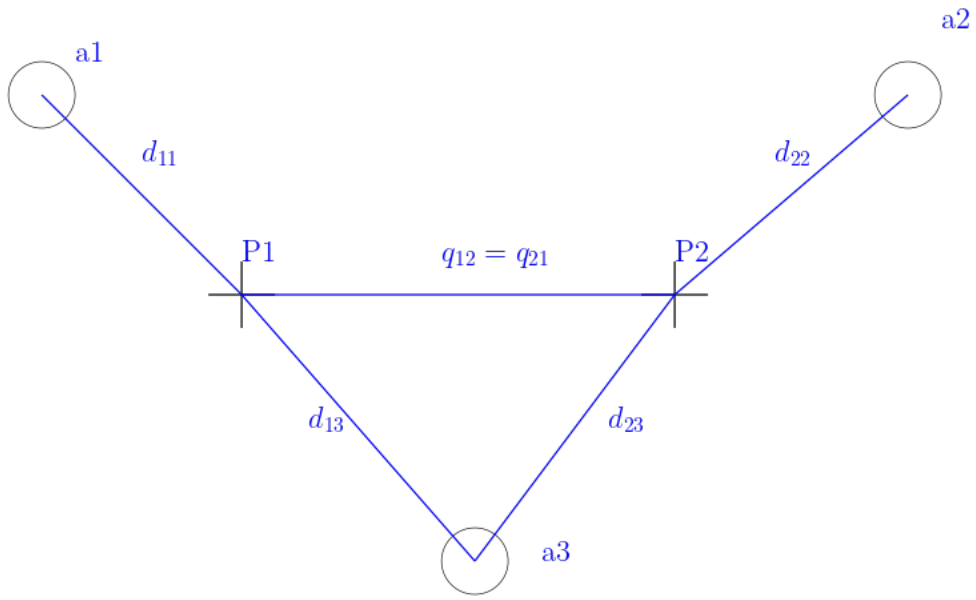

Figure 3: Problem Setup

The dynamics equations followed by the agents are:

$$\begin{aligned} \dot{p_1} &= v_1 \\ \dot{p_2} &= v_2 \end{aligned} \tag{1}$$

Now we define the range measurements and their availability with respect to the different crazyflie agents:

- Range information that we have for the first target crazyflie with a_3 being our shared anchor:

$$\begin{aligned} d_{11} &= ||p_1 - a_1|| \\ d_{13} &= ||p_1 - a_3|| \\ q_{12} &= ||p_1 - p_2|| \end{aligned} \tag{2}$$

- Similarly, for the second crazyflie we have:

$$\begin{aligned} d_{22} &= ||p_2 - a_2|| \\ d_{23} &= ||p_2 - a_3|| \\ q_{21} &= ||p_2 - p_1|| \end{aligned} \tag{3}$$

In our setup, it is **important** to note that the two crazyflies share their pose estimate and the range from the common anchor with each other.

3.3 State Estimation and Lyapunov Approach

We start with the following states to estimate from the dynamics equations: (where χ is the correction term for the estimated state)

$$\begin{aligned} \hat{p_1} &= v_1 + \chi \\ \hat{p_1} &= v_1 + \chi \end{aligned} \tag{4}$$

Our error estimates accordingly become:

$$\begin{aligned} \tilde{p_1} &= p_1 - \hat{p_1} \\ \tilde{p_2} &= p_2 - \hat{p_2} \end{aligned} \tag{5}$$

Now we extract the following information from the above problem setup:

$$\begin{aligned} d_{23}^2 &= ||p_2||^2 + ||a_3||^2 - 2p_2^T a_3 \\ d_{13}^2 &= ||p_1||^2 + ||a_3||^2 - 2p_1^T a_3 \\ q_{21}^2 &= ||p_1||^2 + ||p_2||^2 - 2p_1^T p_2 \end{aligned} \tag{6}$$

From there, we get our common measured value which is: (here y_{ij} means i^{th} measurement for the j^{th} drone)

$$y_{22} = d_{23}^2 + d_{13}^2 - q_{21}^2 - 2||a_3||^2 = -2a_3^T(p_1 + p_2) + 2p_1^T p_2 = y_{21} \tag{7}$$

For the individual drones, we have the following measured values:

$$\begin{aligned} y_{12} = d_{22}^2 - d_{23}^2 + ||a_3||^2 - ||a_2||^2 &= 2(a_3 - a_2)^T p_2 \\ (a_3 - a_2)^T &= C_{23}^T \\ y_{11} = d_{11}^2 - d_{13}^2 + ||a_3||^2 - ||a_1||^2 &= 2(a_3 - a_1)^T p_1 \\ (a_3 - a_1)^T &= C_{13}^T \end{aligned} \tag{8}$$

Now we can write the correction terms in equation **(4)** as: (here k_{ij} are controller gains where j represents the drone id)

$$\begin{aligned} \dot{\hat{p_1}} &= v_1 + k_{11}[y_{11} - 2c_{13}^T \hat{p_1}] + k_{21}[y_{21} + 2a_3^T(\hat{p_1} + \hat{p_2}) - 2\hat{p_1}^T \hat{p_2}] \\ \dot{\hat{p_2}} &= v_2 + k_{12}[y_{12} - 2c_{23}^T \hat{p_2}] + k_{22}[y_{22} + 2a_3^T(\hat{p_1} + \hat{p_2}) - 2\hat{p_1}^T \hat{p_2}] \end{aligned} \tag{9}$$

We take the controller gains of the following form in an attempt to attain some kind of symmetry in our state equations:

$$\begin{aligned} k_{11} &= \gamma_1 * C_{13} \\ k_{21} &= \gamma_1 * (\hat{p_2} - a_3) \\ k_{12} &= \gamma_2 * C_{23} \\ k_{22} &= \gamma_2 * (\hat{p_1} - a_3) \end{aligned} \tag{10}$$

The error estimates in equation **(5)** now can be written as:

$$\begin{aligned}\dot{\tilde{p}_1} &= -2k_{11}[c_{13}^T\tilde{p_1}] - 2k_{21}[-a_3^T(\tilde{p_1}+\tilde{p_2}) + \tilde{p_1}^T p_2 + \hat{p_1}^T\tilde{p_2}] \\ \dot{\tilde{p}_2} &= -2k_{12}[c_{23}^T\tilde{p_2}] - 2k_{22}[-a_3^T(\tilde{p_1}+\tilde{p_2}) + \tilde{p_2}^T p_1 + \hat{p_2}^T\tilde{p_1}]\end{aligned} \tag{11}$$

We perform the following rearrangements in our equations. We begin with the terms:

$$-a_3^T([p_1-\hat{p_1}]+[p_2-\hat{p_2}]) + p_1^T p_2 - \hat{p_1}^T\hat{p_2} \tag{12}$$

For the first ($\mathbf{p_1}$) drone:

$$\begin{aligned}\Rightarrow -a_3^T(\tilde{p_1}+\tilde{p_2}) + p_1^T p_2 - \hat{p_1}^T(p_2-\tilde{p_2}) \\ \Rightarrow -a_3^T(\tilde{p_1}+\tilde{p_2}) + (p_1^T-\hat{p_1}^T)p_2 + \hat{p_1}^T\tilde{p_2}) \\ \Rightarrow -a_3^T\tilde{p_1} - a_3^T\tilde{p_2} + \tilde{p_1}^T p_2 + \hat{p_1}^T\tilde{p_2}) \\ \Rightarrow -a_3^T\tilde{p_1} - a_3^T\tilde{p_2} + p_2^T\tilde{p_1} + \hat{p_1}^T\tilde{p_2}) \\ \Rightarrow \mathbf{(p_2-a_3)^T\tilde{p_1} + (\hat{p_1}-a_3)^T\tilde{p_2}}\end{aligned} \tag{13}$$

For the second ($\mathbf{p_2}$) drone we rewrite the original terms as:

$$-a_3^T(\tilde{p_1}+\tilde{p_2}) + \mathbf{p_2^T p_1} - \mathbf{\hat{p_2}^T\hat{p_1}} \tag{14}$$

Then we rearrange as follows:

$$\begin{aligned}\Rightarrow -a_3^T(\tilde{p_1}+\tilde{p_2}) + p_2^T p_1 - \hat{p_2}^T(p_1-\tilde{p_1}) \\ \Rightarrow -a_3^T(\tilde{p_1}+\tilde{p_2}) + (p_2^T-\hat{p_2}^T)p_1 + \hat{p_2}^T\tilde{p_1}) \\ \Rightarrow -a_3^T\tilde{p_1} - a_3^T\tilde{p_2} + \tilde{p_2}^T p_1 + \hat{p_2}^T\tilde{p_1}) \\ \Rightarrow -a_3^T\tilde{p_1} - a_3^T\tilde{p_2} + p_1^T\tilde{p_2} + \hat{p_2}^T\tilde{p_1}) \\ \Rightarrow \mathbf{(\hat{p_2}-a_3)^T\tilde{p_1} + (p_1-a_3)^T\tilde{p_2}}\end{aligned} \tag{15}$$

Using this simplified form, we put in the controller gains in equation **(11)**, which gives us:

$$\begin{aligned}\dot{\tilde{p}_1} &= -2\gamma_1[c_{13}c_{13}^T]\tilde{p_1} - 2\gamma_1(\hat{p_2}-a_3)[(p_2-a_3)^T\tilde{p_1} + (\hat{p_1}-a_3)^T\tilde{p_2}] \\ \dot{\tilde{p}_2} &= -2\gamma_2[c_{23}c_{23}^T]\tilde{p_2} - 2\gamma_2(\hat{p_1}-a_3)[(\hat{p_2}-a_3)^T\tilde{p_1} + (p_1-a_3)^T\tilde{p_2}]\end{aligned} \tag{16}$$

Now before further simplification, we observe that we have terms that either contain just $\tilde{p_1}$ or just $\tilde{p_2}$. These terms with isolated error estimates are easier to handle while designing the *Lyapunov function* while the terms with both of them clubbed are difficult to manage and for further reading are termed as *" extra terms "*.

Now we introduce M_1 and M_2 as follows: (*extra terms* are written in **bold**)

$$\begin{aligned}M_1 &= c_{13}c_{13}^T + (p_2-a_3)(p_2-a_3)^T \\ M_2 &= c_{23}c_{23}^T + (p_1-a_3)(p_1-a_3)^T \\ \dot{\tilde{p}_1} &= -2\gamma_1 M_1\tilde{p_1} - 2\gamma_1(\hat{p_2}-a_3)(\hat{p_1}-a_3)^T\tilde{p_2} + \mathbf{2\gamma_1\tilde{p_2}(p_2-a_3)^T\tilde{p_1}} \\ \dot{\tilde{p}_2} &= -2\gamma_2 M_2\tilde{p_2} - 2\gamma_2(\hat{p_1}-a_3)(\hat{p_2}-a_3)^T\tilde{p_1} + \mathbf{2\gamma_2\tilde{p_1}(p_1-a_3)^T\tilde{p_2}}\end{aligned}$$

We begin by taking our Lyapunov function of the following form and differentiating it:

$$\begin{aligned} V &= \frac{1}{2}||\tilde{p_1}||^2 + \frac{\mu}{2}||\tilde{p_2}||^2 \\ \dot{V} &= \tilde{p_1}^T \dot{\tilde{p}_1} + \mu \tilde{p_2}^T \dot{\tilde{p}_2} \end{aligned} \tag{17}$$

Putting the values of $\tilde{p_1}$ and $\tilde{p_2}$ in the Lyapunov we get:

$$\begin{aligned} \dot{V} = \tilde{p_1}^T[-2\gamma_1 M_1]\tilde{p_1} + \tilde{p_2}^T[-2\mu\gamma_2 M_2]\tilde{p_2} + \tilde{p_1}^T[-2\gamma_1(\hat{p_2} - a_3)(p_1 - a_3)^T]\tilde{p_2} \\ + \tilde{p_2}^T[-2\mu\gamma_2(\hat{p_1} - a_3)(p_2 - a_3)^T]\tilde{p_1} + extraterms \end{aligned} \tag{18}$$

Comparing this expansion with the form:

$$\begin{bmatrix} \tilde{p_1}^T & \tilde{p_2}^T \end{bmatrix} \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix} \begin{bmatrix} \tilde{p_1} \\ \tilde{p_2} \end{bmatrix} = \tilde{p_1}^T P_{11} \tilde{p_1} + \tilde{p_2}^T P_{21} \tilde{p_1} + \tilde{p_1}^T P_{12} \tilde{p_2} + \tilde{p_2}^T P_{22} \tilde{p_2} \tag{19}$$

we get the following value for our **P matrix**:

$$\begin{aligned} \mathbf{P_{11}} &= -2\gamma_1 M_1 \\ \mathbf{P_{12}} &= -2\gamma_1(\hat{p_2} - a_3)(\hat{p_1} - a_3)^T \\ \mathbf{P_{21}} &= -2\mu\gamma_2(\hat{p_1} - a_3)(\hat{p_2} - a_3)^T \\ \mathbf{P_{22}} &= -2\mu\gamma_2 M_2 \end{aligned}$$

3.4 Challenges with the Remaining terms

Terms remaining to deal with:

$$\dot{V} = \tilde{p_1}^T 2\gamma_1 \tilde{p_2}(p_2 - a_3)^T \tilde{p_1} + \mu \tilde{p_2}^T 2\gamma_2 \tilde{p_1}(p_1 - a_3)^T \tilde{p_2} \tag{20}$$

The challenge with these is we need to manipulate our state equations in such a way by either introducing a new state variable or redesigning the Lyapunov, that these terms are eliminated, and we get an isolated **P matrix** which can then follow the results of *positive definiteness* in the particular form expressed above.

4 Hardware Setup

4.1 Background

The choice of drone that we worked with was the Crazyflie 2.1 by *Bitcraze*. The advantages are it's lightweight, easily customizable firmware and open-source flying development platform. The customizable options of various decks, which can be attached to the base crazyflie make it extremely useful as we could test out different positioning systems for their accuracy. The crazyflie client provides an easy-to-learn user experience as well. For the final testing environment, we need to fix the UWB modules (*anchors*) at appropriate locations which do not have any obstruction of view and are about *5-10*cm away from the ground, walls, and the roof of the room. More about the positioning systems are discussed in the appropriate subsection below.

4.2 Crazyflie Suite

We used the crazyflie system along with the Crayflie PA radio USB module which one can connect to their computer system. We installed the suggested Crazyflie VM; otherwise, the firmware can be compiled in Ubuntu 20 and newer versions.

Figure 4: Assembled Crazyflie

4.3 Positioning Systems

We used two positioning systems available in the Crazyflie suite, namely *Lighthouse* positioning system and *Loco* positioning system. The plan was to use the data from the lighthouse positioning system as our ground truth and reference trajectory. This option was taken because of the unavailability of a more accurate motion capture setup in the lab due to renovations at the time.

4.3.1 Lighthouse Positioning System

This system was based on *optical beam*, which allowed the Crazyflie agent to calculate its position with decimeter accuracy and millimeter precision. It had two base stations that contained periodically spanning optical beams, which are further received by the photo-diode (*light receiver*) on the Crazyflie equipped with the Lighthouse deck.

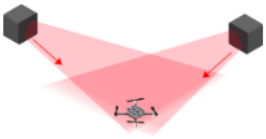

Figure 5: Source: Bitcraze Documentation

It is important to note that in order to measure the pose and orientation just from the range measurements provided by these lighthouse decks, the Crazyflie needs to know the position and orientation of these base stations.

A successful setup was created using this positioning system (video of the setup here), and flight tests were conducted to log different data like the pose, distance measurements, gyro measurements, and information from the IMU. After this, we moved to the Loco system to set up the testing part of the environment.

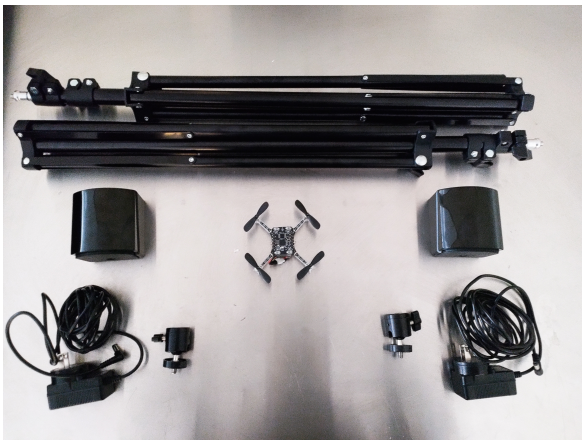

Figure 6: Lighthouse Setup

4.3.2 Loco Positioning System

In it's essence, it is a *local* positioning system that is based on the Ultra Wide Band Radio. This can be thought of as a mini GPS, and each of the UWB modules act as a satellite with known pose and orientation which allows the crazyflie to calculate it's position and orientation from the range measurements from each of these *anchors*.

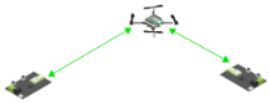

Figure 7: Source: Bitcraze Documentation

Along with proper positioning of the anchors, a Loco deck is installed to the Crazyflie agent to allow estimation of absolute position, which can be further used for autonomous flight. Flight tests were conducted to make the agent move in a square of a defined length.

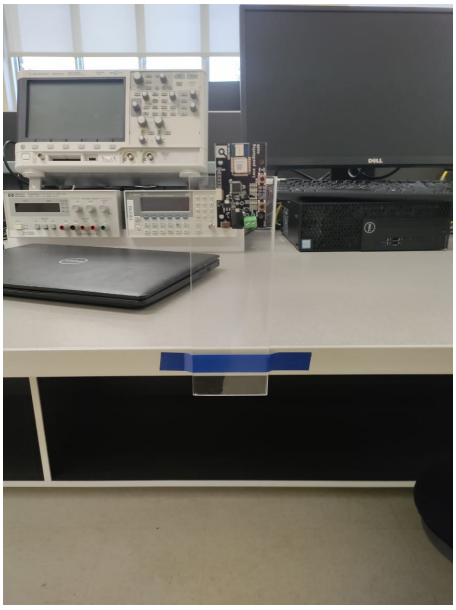

Figure 8: UWB Modules Setup

Calibrations for the flight tests: Before any flight test, we keep the crazyflie in the marked center (the assumed origin from where we have measured all the positions of the anchors). We turn on the crazyflie

client which communicates to the agent via the PA radio module, and we check if the crazyflie is receiving data from the anchors. This is indicated by the green boxes under the Loco positioning tab in the CF client. We feed in the positions of the anchors through the client, which is measured from the assumed origin. To check everything, one should manually pick the Crazyflie drone up and try to move it near any anchor, and check the status in the client for the same. Also, it is important to place the Crazyflie in the correct orientation in the x-y plane.

After these steps, we can run the *python* script for autonomous flight.

```python
def take_off(cf, position):
    take_off_time = 1.0
    sleep_time = 0.1
    steps = int(take_off_time / sleep_time)
    vz = position[2] / take_off_time

    print(vz)

    for i in range(steps):
        cf.commander.send_velocity_world_setpoint(0, 0, vz, 0)
        time.sleep(sleep_time)


def land(cf, position):
    landing_time = 1.0
    sleep_time = 0.1
    steps = int(landing_time / sleep_time)
    vz = -position[2] / landing_time

    print(vz)

    for _ in range(steps):
        cf.commander.send_velocity_world_setpoint(0, 0, vz, 0)
        time.sleep(sleep_time)

    cf.commander.send_stop_setpoint()
    # Make sure that the last packet leaves before the link is closed
    # since the message queue is not flushed before closing
    time.sleep(0.1)
```

```python
def run_sequence(scf, sequence):
    try:
        cf = scf.cf

        take_off(cf, sequence[0])
        for position in sequence:
            print('Setting position {}'.format(position))
            end_time = time.time() + position[3]
            while time.time() < end_time:
                cf.commander.send_position_setpoint(position[0],
                                                    position[1],
                                                    position[2], 0)
                time.sleep(0.1)
        land(cf, sequence[-1])
    except Exception as e:
        print(e)


if __name__ == '__main__':
    # logging.basicConfig(level=logging.DEBUG)
    cflib.crtp.init_drivers()

    factory = CachedCfFactory(rw_cache='./cache')
```

Figure 9: Python script to run a Swarm Sequence

5 MATLAB Implementation

A Luenberger Observer-based model was implemented in Simulink to create a pipeline to generate trajectories from the data logs that we collect from flight tests. The entire codebase can be found at the GitHub repository linked here.

```
filename = 'test.xlsx';
% T = readtable(filename, 'ReadRowNames', true);
% T(1:3, 1:4);
% T(2,3)

% use raw as it has all the columns
[numbers, txt, raw] = xlsread(filename);

%plotting ranges fro different anchors
range0 = str2double(raw(:, 3));
% range1 = str2double(raw(:, 4));
range2 = str2double(raw(:, 4));
range3 = str2double(raw(:, 5));
% range4 = str2double(raw(:, 3));
% range5 = str2double(raw(:, 3));
% range6 = str2double(raw(:, 3));
```

test

	VarName1	timestamp	rangingdis...	rangingdis...	rangingdis...
	Number	Text	Text	Text	Text
1		timestamp	ranging.dist...	ranging.dist...	ranging.dist...
2	0	2022-06-15...	2.3918	2.4938	3.1052
3	1	2022-06-15...	2.3918	2.4409	3.1052
4	2	2022-06-15...	2.3913	2.4468	2.8925
5	3	2022-06-15...	2.2290	2.4468	2.8925
6	4	2022-06-15...	2.2290	2.3271	3.1287
7	5	2022-06-15...	2.5524	2.3179	3.1287
8	6	2022-06-15...	2.5524	2.3179	2.9971
9	7	2022-06-15...	2.5491	2.3179	3.1532
10	8	2022-06-15...	2.4081	2.3179	3.1532
11	9	2022-06-15...	2.4081	2.4350	3.1169
12	10	2022-06-15...	2.3987	2.4187	3.0509

Figure 10: Loading the logging data

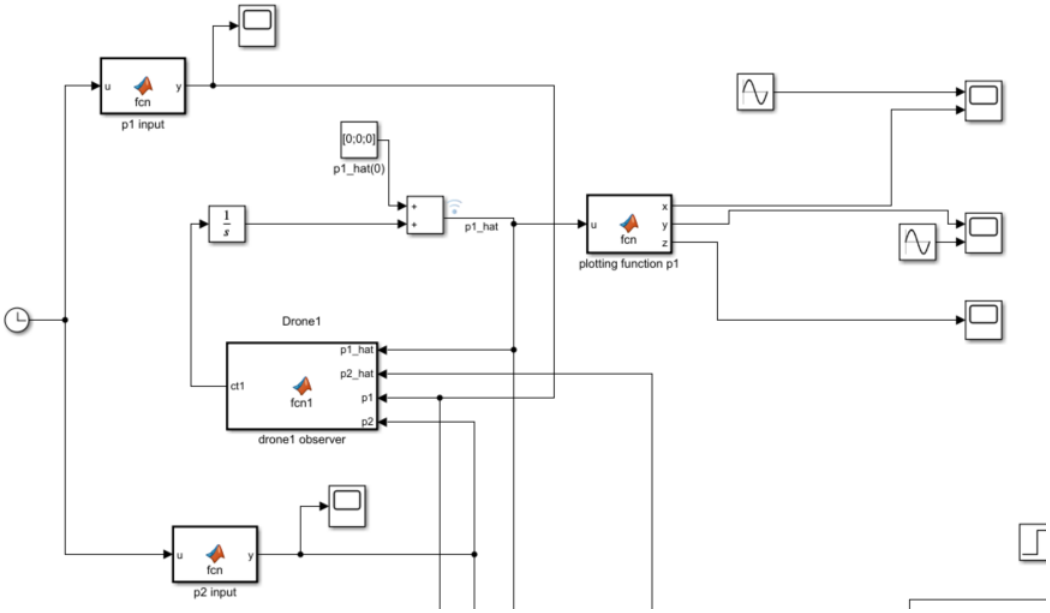

Figure 11: Part of the Simulink Model for Drone 1

6 Future Work

Future work would incorporate the following tasks:

- Handling the *extra terms* in the theoretical proof by either choosing a different Lyapunov Function or adding another state which somehow manipulates the equations to eliminate such terms.
- Finding a means of communication between the crazyflies so that they can share their position with each other.
- After the above issues have been resolved, we can work towards implementing it on a multi-agent system of two crazyflies and extend it to multiple crazyflie agents as well.
- It is very exciting to think about this multi-agent system in other similar scenarios with some arbitrary number of UWB modules being *unique* and some being *shared* and their impact on the cooperative localization pipeline that we have.