Development of a Multirotor Aerial Robotic Platform and Impulse Control with Mass Suspended by a Link

A Thesis
Submitted in partial fulfillment of
the requirements for the degree of
Master of Technology
by

Modi Harsh Jashvantbhai
(203100049)

Department of Mechanical Engineering
Indian Institute of Technology Bombay
Mumbai 400076 (India)

17 June 2022

Acceptance Certificate

Department of Mechanical Engineering

Indian Institute of Technology, Bombay

The M.Tech. thesis entitled "Development of a Multirotor Aerial Robotic Platform and Impulse Control with Mass Suspended by a Link" submitted by Modi Harsh Jashvantbhai (203100049) is approved for the partial fulfillment of the degree of Master's of technology.

Examiners:

Prof. Leena Vachhani

Prof. Amber Shrivastava

Supervisor:

Prof. Vivek Sangwan

Date: 24 June, 2022

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I declare that I have properly and accurately acknowledged all sources used in the production of this report. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Modi Harsh Jashvantbhai

Date: 17 June 2022

(203100049)

Abstract

Multirotor Aerial Robotic Platforms can be thought of as flying robots. These platforms can have various capabilities, such as impacting a cable suspended mass on the object, doing object manipulations, etc. In order to perform these tasks, the control of the flying platform (i.e UAV) needs to be very precise (in mm order). In this project, the hardware design, controller design, and manufacturing of this precisely controlled UAV are explored. In order to run the control algorithms, two different flight controllers are considered, one is ready to use a commercial flight controller and another is a fully customized flight controller. Various flight experiments for a commercial flight controller have been performed. The offboard outerloop controller to be used with Pixhawk Flight Controller has been designed. Various experiments such as hover control, trajectory control, and the experiments with a 1 DOF pendulum setup have been performed.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Multirotor Aerial Robotic Platform

The aim of this project is to design and make a small-scale ($\sim$ 0.1 to 1 m) multirotor Unmanned Aerial Vehicle (UAV) and do various dynamic experiments on it. UAV is an aerial vehicle, which does not have any pilot onboard. The first question is: why to make the UAV on our own when we have so many readily available commercial products in the market. The answer is: they are mostly limited to being used in cinematography etc. We can control their trajectories very easily. However, they are designed to be used by regular users, so they are very difficult to control at the inner loop level (Fig 1.10). In order to do dynamic experiments (e.g. mount an arm on it and check the interaction forces with the wall), we need to control the UAV at the inner loop level, which can be easily done only on self-made UAV.

In order to design a UAV capable of following the trajectories precisely and doing the dynamic experiments, background research was done about various components used in UAVs. (3) has performed research about power and weight considerations of the small UAVs. (2) has described various components used in the UAVs and the assembly of the hardware. It also shows the performance of the flight in hover experiments, but trajectory experiments and further dynamic experiments have not been performed. (8) has done the theoretical and simulation work about impacting the object via a cable suspended load, however, the hardware implementation was pending. In order to control the swing of the pendulum, the concepts from (9) and (10) have been used in this project. In the next section, we will explore aerial manipulators.

1.2 Aerial Manipulators

The ultimate aim of the project is to mount various manipulators on the UAV to carry out the required tasks. Reference (11) shows various kinds of aerial manipulator projects under research at various universities. In this project, one of the experiments which we are planning is impacting a cable suspended mass to the quadrotor (8). We are planning to achieve various complex tasks such as putting a basketball into a basket using the impact of the cable suspended mass. Another task that is to be done is mounting a 2 link manipulator on the quadrotor and analyzing the forces it can apply on the wall. After achieving precise control over the movement of the UAV, these experiments need to be performed.

1.3 Overall Schematic

The Fig.1.1 explains the simplified overall schematic of the quadrotor indoor UAV. The sensors (i.e. IMU - Inertial Measurement Unit, Visual sensors etc.) provide data to MCU (Microcontroller Unit) and based on the control scheme, the MCU generates the actuation signal in the form of PWM (Pulse Width Modulated) signal to be sent to ESCs. The ESCs based on signal received from MCU and using the 12V battery power, generates 3 phase signal for BLDC (Brushless Direct Current) motor. The hall effect sensor or back EMF (for sensorless BLDC motor) of the motor provides the feedback to ESC to control the phase signal correctly. The detailed information of this is available in the seminar report (12).

1.4 Quadrotor Configurations

Multirotor UAVs can have various numbers of rotors as shown in Fig.1.2. The most common multirotor UAV is quadrotor UAV. So, to begin with in this project, we will be focusing upon quadrotor UAVs as they are the most commonly used multirotor platforms and have a good component availibility. The Fig.1.3 shows the various configurations of quadrotor UAV. They are X, H and + configurations. The pros and cons of each of them is as below: (2).

- The X configuration is easy to construct and it is ideal for forward facing camera.
- The + configuration is simple in terms of mathematical model and control aspect. But for forward facing camera, it is not good since the motor in the front can obstruct the view.

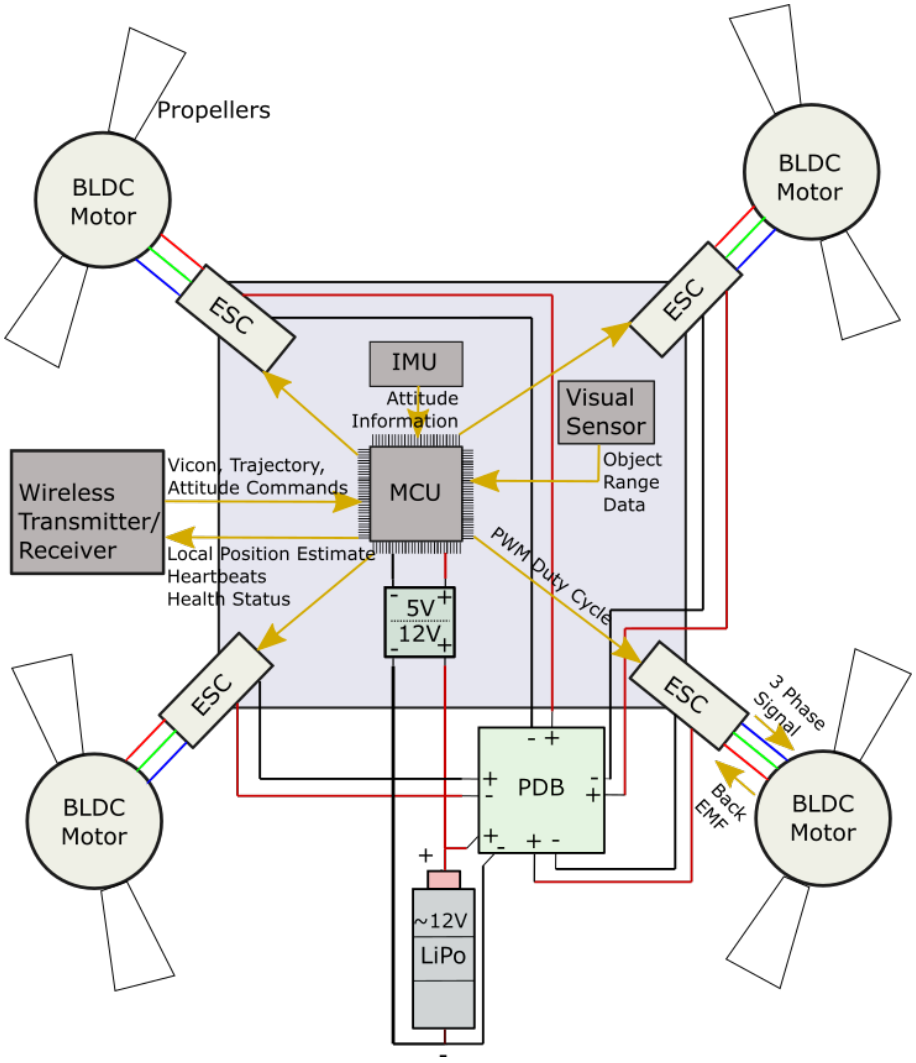


Figure 1.1: Overall Schematic

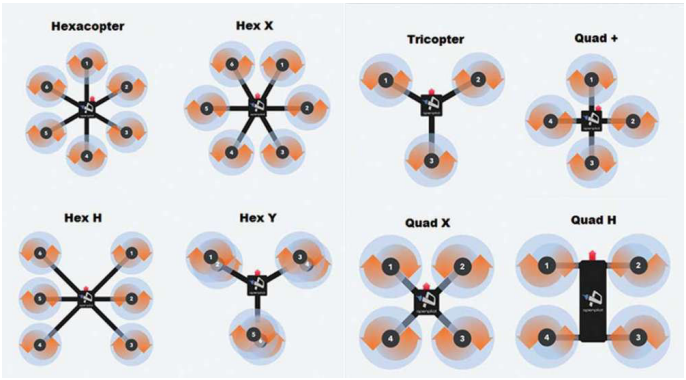


Figure 1.2: Various Multirotor UAVs (Source: (1))

- The H configuration is also good for forward facing camera, but is not symmetric about center.

In this project, as we need to mount robotic arm etc. in the front, so we will be using X configuration.

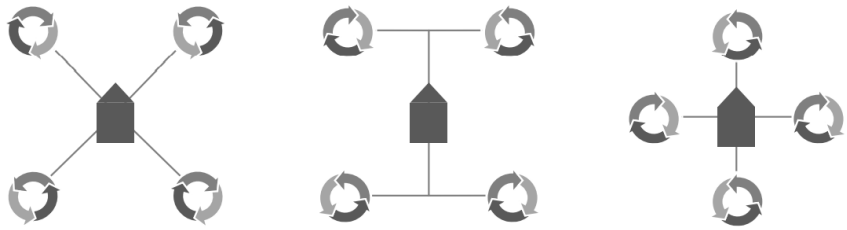
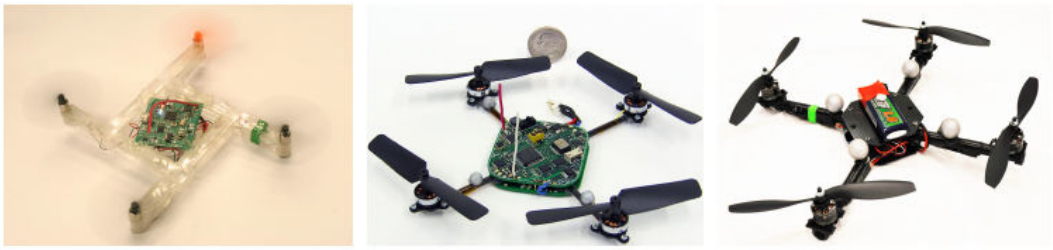


Figure 1.3: X, H and + configurations of Quadrotors (Source: (2))

Now, we will explore the importance of size, mass and power considerations in the further subsections. The subsections Size considerations, Mass considerations and Power considerations are referenced from (3).

Size considerations

(a) AscTec Hummingbird (b) AscTec Pelican (unloaded) (c) AscTec Pelican (loaded)



(d) Folded Quadrotor (e) KMel Nano (f) KMel kQuad 500

Figure 1.4: UAVs used for size, mass and power considerations (Source: (3))

This size considerations analysis was done by the reference (2) and it is based on the order of magnitude analysis of various factors of the UAV. The summary of the analysis is as below:

Characteristic Length (distance between motor and center of mass): d
So, Mass (m) $\propto d^3$

Thrust $T = C_T\omega^2$ and Drag $Q = C_Q\omega^2$
where, ω is rotor speed; C_T and C_Q are thrust and drag coefficients respectively, which depend upon rotor disc area and square of the rotor radius. Also, the rotor radius scales linearly with d

Hence, C_T or $C_Q \propto d^4 \implies T$ or $Q \propto d^4 \cdot \omega^2$

Linear Acceleration $a = \frac{T}{m} \implies a \propto \frac{d^4 \cdot \omega^2}{d^3} \implies a \propto d \cdot \omega^2$

Angular Acceleration $\alpha = \frac{\tau}{I} \propto \frac{T \cdot d}{m \cdot d^2} \implies \alpha \propto \frac{d^4 \cdot \omega^2 \cdot d}{d^3 \cdot d^2} \implies \alpha \propto 1 \cdot \omega^2$

Based on various fluid mechanics assumptions, $\omega \propto \frac{1}{d}$ or $\omega^2 \propto \frac{1}{d}$

Which gives us, $a \propto \frac{1}{d}$ and $\alpha \propto \frac{1}{d^2}$ or $a \propto 1$ and $\alpha \propto \frac{1}{d}$

Mass considerations

As per the analysis done by the reference (3), we can see in Fig.1.5 that battery has the highest contribution to the mass of the UAV, so high-density lightweight batteries are the requirements of the batteries in such applications. Also, we can see in Fig.1.6 and Fig.1.7 that the motors have a high impact on inertia as they are away from the center of mass.

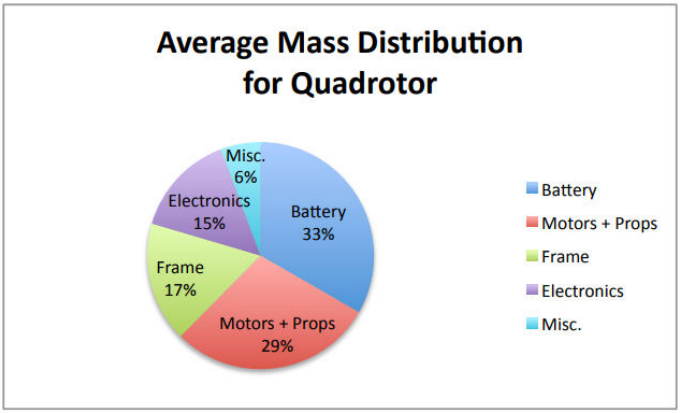


Figure 1.5: Mass Distribution of Different Components (Source: (3))

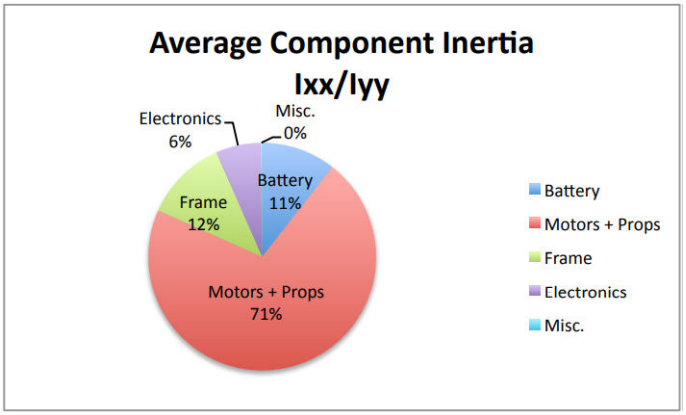


Figure 1.6: Inertia Distribution of Different Components (Source: (3))

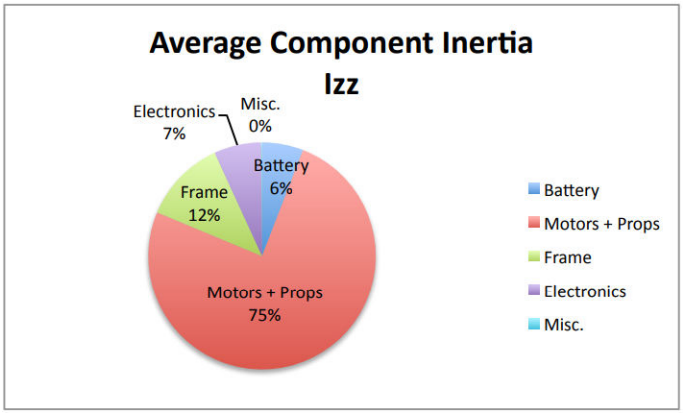


Figure 1.7: Inertia Distribution of Different Components (Source: (3))

Power considerations

As per Fig.1.8, we can observe that most of the power is consumed by the motors. So, making motors more efficient can lead to increased endurance of UAV and hence increse the range.

1.5 Flight Controller

There are various open-source flight controllers available such as Pixhawk, Sparky 2, Paparazzi, etc. In these kinds of ready-to-use hardware, it is difficult to modify the inner control loop (Fig 1.10) of the vehicle, but it is easy to do the implementation for the flight. On the other hand, we can design our own flight controller based on general-purpose programming boards and various sensors. In this type of flight controller, we will have a control over all the parameters of flight easily, but it will be time and effort consuming to program. So in this project, we are first initializing the development with the open source

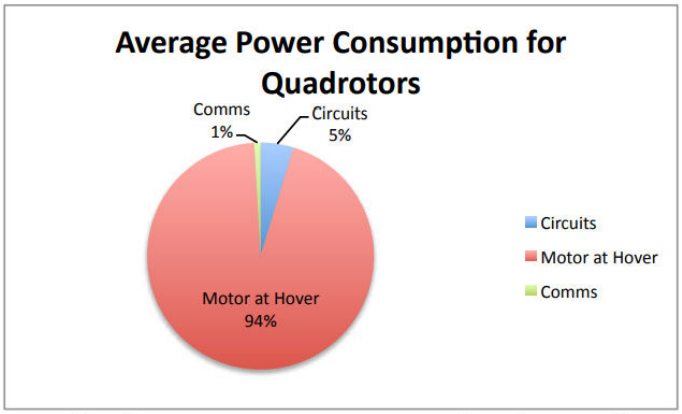


Figure 1.8: Power Distribution of Different Components (Source: (3))

hardware Pixhawk and will be slowly building the flight controller based on Teensy board. Chapter 3 and chapter 2 explain the work done for both of these flight controllers. Before exploring the hardware implementation, we will first explore the simulation performed in section 1.6.

1.6 Simulation

In order to understand the working of the controller algorithm and quadrotor dynamics, 3D quadrotor simulation in MATLAB was performed. This learning experience will be useful while designing a Teensy-based flight controller.

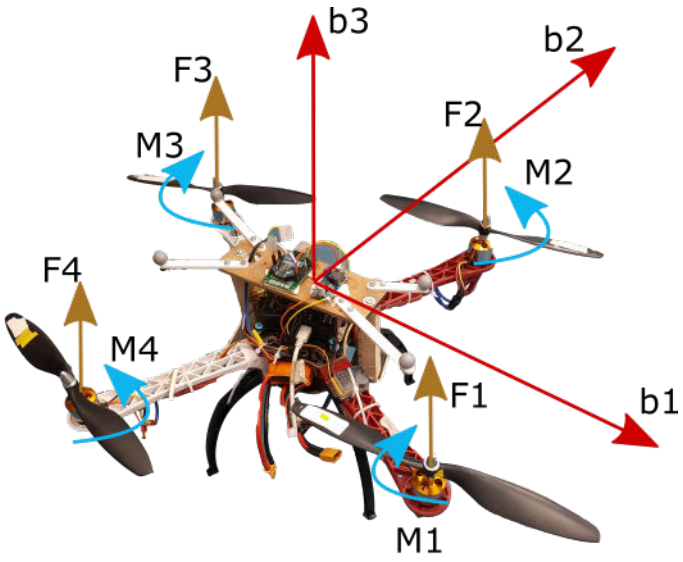


Figure 1.9: X-Quadrotor - Body Frame Coordinate System and Quadrotor Fources

System Dynamics

The dynamics and controls of the quadrotor considered is as per below (Reference (13)):

The linear equations of motions are (described in world frame):

$$m\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix} \tag{1.1}$$

where R is a rotation matrix describing the relation between body frame of the quadrotor and the world frame. It is derived using 3 consecutive Euler angle rotations ψ about Z axis, ϕ about X axis and θ about Y axis.

$$R = \begin{bmatrix} c\psi & -s\psi & 0 \\ s\psi & c\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\phi & -s\phi \\ 0 & s\phi & c\phi \end{bmatrix} \cdot \begin{bmatrix} c\theta & 0 & s\theta \\ 0 & 1 & 0 \\ -s\theta & 0 & c\theta \end{bmatrix} = \begin{bmatrix} c\psi c\theta - s\phi s\psi s\theta & -c\phi s\psi & c\psi s\theta + c\theta s\phi s\psi \\ c\theta s\psi + c\psi s\phi s\theta & c\phi c\psi & s\psi s\theta - c\psi c\theta s\phi \\ -c\phi s\theta & s\phi & c\phi c\theta \end{bmatrix} \tag{1.2}$$

The angular equations of motion are:

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{L}{\sqrt{2}}(F_2 + F_3 - F_1 - F_4) \\ \frac{L}{\sqrt{2}}(F_3 + F_4 - F_1 - F_2) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{1.3}$$

where p, q and r (anglular velocities in body frame) are described as below:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} c\theta & 0 & -c\phi s\theta \\ 0 & 1 & s\phi \\ s\theta & 0 & c\phi c\theta \end{bmatrix} \cdot \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{1.4}$$

Total 12 states can describe the quadrotor's position and orientation:

S(1): x: Position in X direction

S(2): y: Position in Y direction

S(3): z: Position in Z direction

S(4): $\dot{x}$: Velocity in X direction

S(5): $\dot{y}$: Velocity in Y direction

S(6): $\dot{z}$: Velocity in Z direction

S(7): ϕ Rotation about current X axis (Roll)

S(8): θ: Rotation about current Y axis (Pitch)

S(9): ψ: Rotation about world Z axis (Yaw)

S(10): p: Anglular velocity in b1 direction

S(11): q: Anglular velocity in b2 direction

S(12): r: Anglular velocity in b3 direction

The evolution of these states will be driven as following:

$$\begin{bmatrix} \dot{S(1)} \\ \dot{S(2)} \\ \dot{S(3)} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} S(4) \\ S(5) \\ S(6) \end{bmatrix} \tag{1.5}$$

Using Equation 1.1:

$$\begin{bmatrix} \dot{S(4)} \\ \dot{S(5)} \\ \dot{S(6)} \end{bmatrix} = \ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \frac{R}{m} \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix} \tag{1.6}$$

Using Equation 1.4:

$$\begin{bmatrix} \dot{S(7)} \\ \dot{S(8)} \\ \dot{S(9)} \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} c\theta & 0 & -c\phi s\theta \\ 0 & 1 & s\phi \\ s\theta & 0 & c\phi c\theta \end{bmatrix}^{-1} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{1.7}$$

Using Equation 1.3:

$$\begin{bmatrix} \dot{S(10)} \\ \dot{S(11)} \\ \dot{S(12)} \end{bmatrix} = \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = I^{-1} \left(\begin{bmatrix} \frac{L}{\sqrt{2}}(F_2 + F_3 - F_1 - F_4) \\ \frac{L}{\sqrt{2}}(F_3 + F_4 - F_1 - F_2) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} p \\ q \\ r \end{bmatrix} \right) \tag{1.8}$$

The dynamics of the states described in the Equations 1.5, 1.6, 1.7 and 1.8 were coded in MATLAB in order to simulate the dynamics. Using the MATLAB function ODE45, the solution of the system can be derived based on $F_1, F_2, F_3, F_4, M_1, M_2, M_3$, and M_4 as inputs.

Controller

The controller for this simulation is designed using the linearization of the system dynamics equation. Hence, the controller is expected to behave well near the equilibrium point. The controller can be divided into 3 different parts: 1) Inner loop controller, which is running at around 1 kHz frequency 2) Outer loop controller, which is running at 200 Hz frequency and 3) Motor controller, which is also running at 1 kHz frequency (Fig 1.10). The details of these controllers are described in the following subsections.

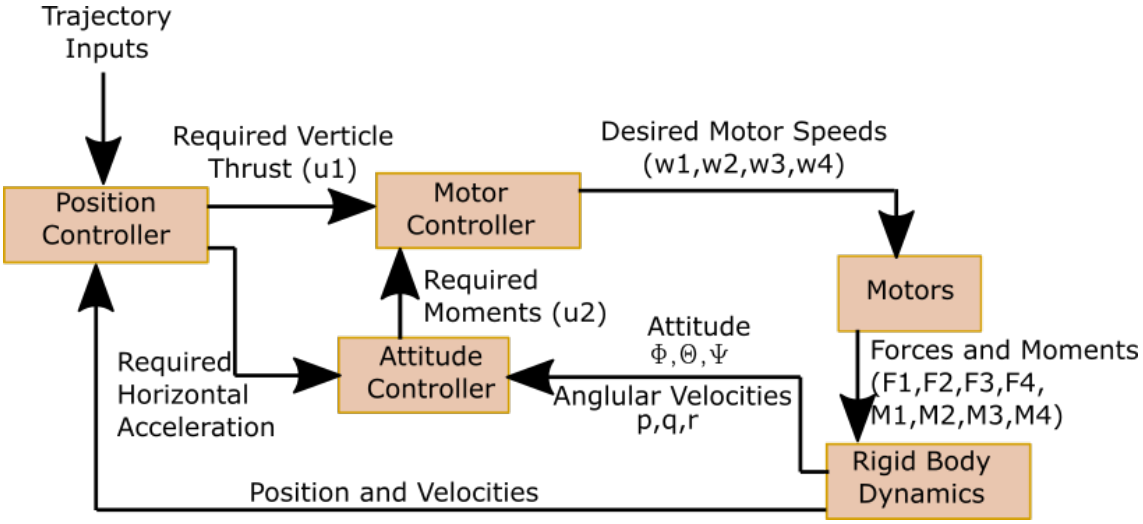


Figure 1.10: Typical Multirotr Control Loop

Inner Loop Controller

The linearization is done about the equilibrium point $\phi = 0$, $\theta = 0$, $\psi = \psi_0$, $\dot{\phi} = 0$, $\dot{\theta} = 0$, $\dot{\psi} = 0$, $\mathbf{r} = r_0, \dot{r} = 0$. So for this equilibrium point, $cos(\theta) \approx 1$, $cos(\phi) \approx 1$, $sin(\phi) \approx \Delta\phi$ and $sin(\theta) \approx \Delta\theta$. Also, $sin(\psi) = sin(\psi_0)$ and $cos(\psi) = cos(\psi_0)$.

Putting in Equation 1.2:

$$R = \begin{bmatrix} c\psi_0 - \Delta\phi s\psi_0 \Delta\theta & -s\psi_0 & c\psi_0 \Delta\theta + \Delta\phi s\psi_0 \\ s\psi_0 + c\psi_0 \Delta\phi \Delta\theta & c\psi_0 & s\psi_0 \Delta\theta - c\psi_0 \Delta\phi \\ -\Delta\theta & \Delta\phi & 1 \end{bmatrix} \tag{1.9}$$

Putting this result in Equation 1.1:

$$m\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + (F_1 + F_2 + F_3 + F_4) \begin{bmatrix} c\psi_0 \Delta\theta + \Delta\phi s\psi_0 \\ s\psi_0 \Delta\theta - c\psi_0 \Delta\phi \\ 1 \end{bmatrix} \tag{1.10}$$

Also, at the equilibrium point $u_1 = F_1 + F_2 + F_3 + F_4 = mg$

So, we get

$$\ddot{\mathbf{r}} = \begin{bmatrix} g(\Delta\theta c\psi_0 + \Delta\phi s\psi_0) \\ g(\Delta\theta s\psi_0 - \Delta\phi c\psi_0) \\ \frac{u_1}{m} - g \end{bmatrix} \tag{1.11}$$

In this equation, the current θ and ϕ need to be operating very close to equilibrium and we also want to maneuver the UAV in this close range only. So we can replace $\Delta\theta$ with θ_{des}, $\Delta\phi$ with ϕ_{des}, ψ_0 with ψ_T and $\ddot{\mathbf{r}}$ with $\ddot{\mathbf{r}}_{\mathbf{des}}$. Here T in ψ_T stands for trajectory. Putting in the above equation, we get,

$$\ddot{\mathbf{r}}_{\mathbf{des}} = \begin{bmatrix} g(\theta_{des}c\psi_T + \phi_{des}s\psi_T) \\ g(\theta_{des}s\psi_T - \phi_{des}c\psi_T) \\ \frac{u_1}{m} - g \end{bmatrix} \tag{1.12}$$

Solving these equations for ϕ_{des} and θ_{des}, we get:

$$\phi_{des} = \frac{1}{g}(\ddot{r}_{1,des}s\psi_T - \ddot{r}_{2,des}c\psi_T) \tag{1.13}$$

$$\theta_{des} = \frac{1}{g}(\ddot{r}_{1,des}c\psi_T + \ddot{r}_{2,des}s\psi_T) \tag{1.14}$$

Here, $\ddot{r}_{1,des}$ and $\ddot{r}_{2,des}$ will be calculated in the outer control loop. Also, at near equilibrium, we can take $p_{des} = 0$ and $q_{des} = 0$. The ψ_T and $\dot{\psi_T} = r_{des}$ will be received from trajectory generator. Based on these values, the control input $\mathbf{u_2}$ will be derived by:

$$\mathbf{u_2} = \begin{bmatrix} k_{p,\phi}(\phi_{des} - \phi) + k_{d,\phi}(p_{des} - p) \\ k_{q,\theta}(\theta_{des} - \theta) + k_{d,\theta}(q_{des} - q) \\ k_{r,\psi}(\psi_{des} - \psi) + k_{d,\psi}(r_{des} - r) \end{bmatrix} = \begin{bmatrix} k_{p,\phi}(\phi_{des} - \phi) - k_{d,\phi} \cdot p \\ k_{q,\theta}(\theta_{des} - \theta) - k_{d,\theta} \cdot q \\ k_{r,\psi}(\psi_T - \psi) + k_{d,\psi}(\dot{\psi}_T - r) \end{bmatrix} \tag{1.15}$$

Here, $k_{p,\phi}$,$k_{q,\theta}$,$k_{r,\psi}$,$k_{d,\phi}$,$k_{d,\theta}$ and $k_{d,\psi}$ are the gain constants which need to be tuned.

Outer Loop Controller

In order to make sure that the errors in position go to 0 exponentially, we can right the feedforward control equation as:

$$(\ddot{\mathbf{r}}_{\mathbf{T}} - \ddot{\mathbf{r}}_{\mathbf{des}}) + \mathbf{k_d}\mathbf{e_v} + \mathbf{k_p}\mathbf{e_p} = 0 \tag{1.16}$$

hence,

$$\ddot{\mathbf{r}}_{\mathbf{des}} = \ddot{\mathbf{r}}_{\mathbf{T}} + \mathbf{k_d}\mathbf{e_v} + \mathbf{k_p}\mathbf{e_p} \tag{1.17}$$

where $\mathbf{e_p}$ and $\mathbf{e_v}$ are positional and velocity errors respectively in 3D space and specified in world frame. $\mathbf{k_p}$ and $\mathbf{k_d}$ are the gain constants (3×3 matrices) which need to be tuned. The errors are calculated from the specified trajectory $\mathbf{r_T}$ as per following:

$$\mathbf{e_p} = ((\mathbf{r_T} - \mathbf{r}) \cdot \hat{\mathbf{n}})\hat{\mathbf{n}} + ((\mathbf{r_T} - \mathbf{r}) \cdot \hat{\mathbf{b}})\hat{\mathbf{b}} \tag{1.18}$$

and

$$\mathbf{e_v} = \dot{\mathbf{r}}_{\mathbf{T}} - \dot{\mathbf{r}} \tag{1.19}$$

Using $\ddot{\mathbf{r}}_{\mathbf{des}}$, the control input u_1 is given by:

$$u_1 = mg + m\ddot{r_{3,des}} \tag{1.20}$$

In the above equations, $\mathbf{r_T}$ and $\dot{\mathbf{r}}_\mathbf{T}$ will be specified by the trajectory generator. $\mathbf{r}$ and $\dot{\mathbf{r}}$ are the current states of the system. $\hat{\mathbf{n}}$ is an unit vector describing the change in the desired velocity direction. $\hat{\mathbf{b}}$ is given by $\hat{\mathbf{t}} \times \hat{\mathbf{n}}$, where $\hat{\mathbf{t}}$ is unit vector in the direction of the desired velocity.

Motor Controller

Based on the control inputs u_1 and $\mathbf{u_2}$, we need to derive the thrust inputs required for the motors. Based on thrust requirements and the empirical relationship of PWM to thrust value (Section 2.3), the PWM signal to be supplied to ESCs can be derived. For the purpose of this simulation, only thrust values are generated and are given as input to the system dynamics.

The average thrust requirement for all for motors will be given by:

$$F_{average} = u_1/4 \tag{1.21}$$

Now, based on inner loop controller output, we will have non-zero $\mathbf{u_2}$ values. Using the $\mathbf{u_2}$ values, we can derive the required increament/decreament from the $F_{average}$ for each motors:

$$F_{b1} = \frac{u_2(1)}{2 \cdot L/\sqrt{2}} \tag{1.22}$$

$$F_{b2} = \frac{u_2(2)}{2 \cdot L/\sqrt{2}} \tag{1.23}$$

$$F_1 = F_{average} - F_{b1}/2 - F_{b2}/2 \tag{1.24}$$

$$F_2 = F_{average} + F_{b1}/2 - F_{b2}/2 \tag{1.25}$$

$$F_3 = F_{average} + F_{b1}/2 + F_{b2}/2 \tag{1.26}$$

$$F_4 = F_{average} - F_{b1}/2 + F_{b2}/2 \tag{1.27}$$

These values are input for the system dynamics described earlier and based on the ODE45 of MATLAB, the state of the system changes and iterations continue.

Simulation Results

As per the system dynamics and controller design, the MATLAB simulation was run for 15 seconds to follow a circular trajectory of radius 5. Zero mean random disturbances were added to the state of the system for each iterations to reflect real life

small perturbations. The system parameters were chosen to be as following:

$$m = 1.2kg, L = 0.15m, I = \begin{bmatrix} 0.00025 & 0 & 0.0000026 \\ 0 & 0.0002320 & 0 \\ 0.0000026 & 0 & 0.0003738 \end{bmatrix} kgm^2$$

Simulation Run 1:

For this simulation run, the gains used were: $\mathbf{k_v} = diag[25, 35, 30], \mathbf{k_p} = diag[195, 195, 100]$ $k_{p,\phi} = 20, k_{d,\phi} = 0.1, k_{q,\theta} = 20, k_{d,\theta} = 0.1, k_{r,\psi} = 1, k_{d,\psi} = 0.1$;

The path followed by quadrotor in X-Y and Z direction can be seen in Fig 1.11 and 1.12. The average position error in X-Y direction was 0.0251 m in this case.

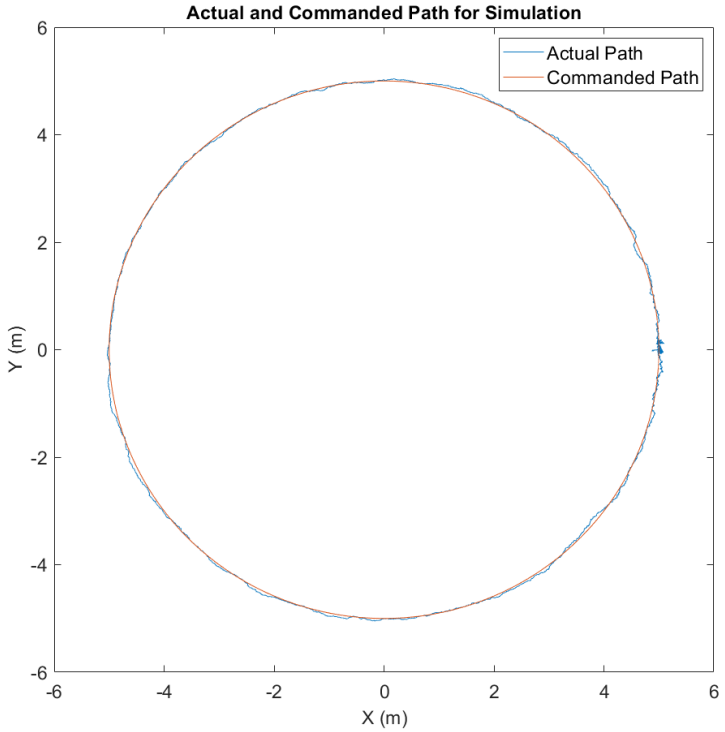


Figure 1.11: Simulation Run 1 - X-Y

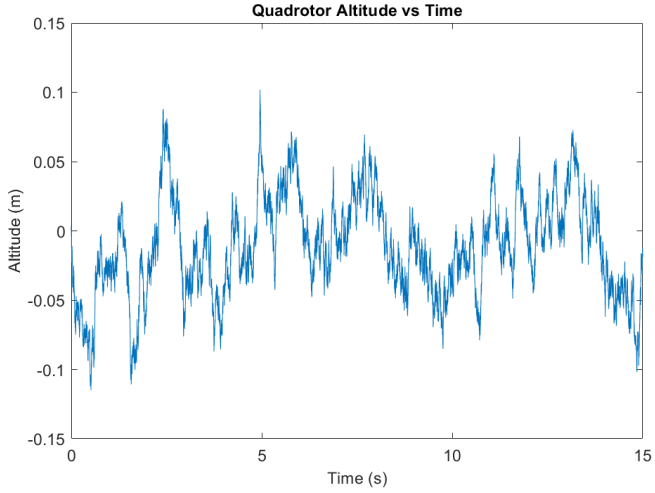


Figure 1.12: Simulation Run 1 - Altitude

Simulation Run 2:

For this simulation run, the gains used were: $\mathbf{k_v} = diag[25, 35, 30], \mathbf{k_p} = diag[195, 195, 100]$ $k_{p,\phi} = 10, k_{d,\phi} = 0.1, k_{q,\theta} = 10, k_{d,\theta} = 0.1, k_{r,\psi} = 1, k_{d,\psi} = 0.1$;

The path followed by quadrotor in X-Y and Z direction can be seen in Fig 1.13 and 1.14. The average position error in X-Y direction was 0.0386 m in this case.

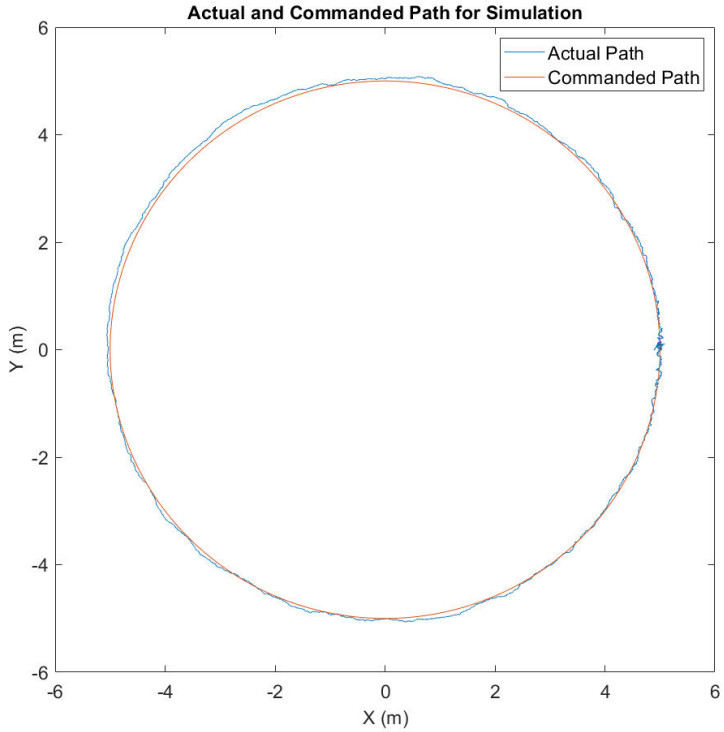


Figure 1.13: Simulation Run 2 - X-Y

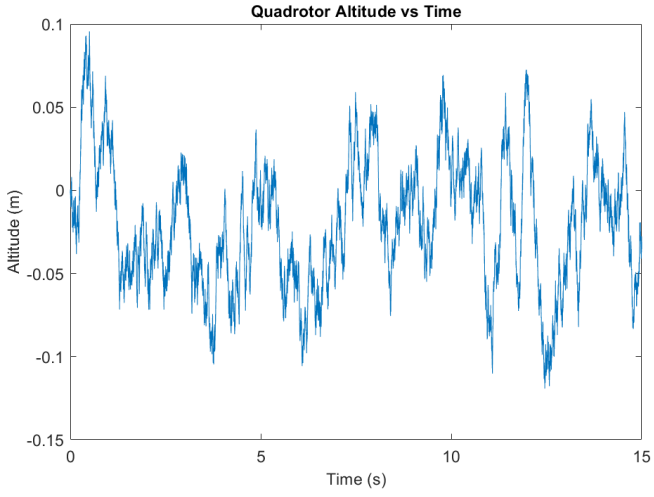


Figure 1.14: Simulation Run 2 - Altitude

Simulation Run 3:

For this simulation run, the gains used were: $\mathbf{k_v} = diag[25, 35, 30], \mathbf{k_p} = diag[195, 195, 100]$ $k_{p,\phi} = 10, k_{d,\phi} = 0.3, k_{q,\theta} = 10, k_{d,\theta} = 0.3, k_{r,\psi} = 1, k_{d,\psi} = 0.3$;

The path followed by quadrotor in X-Y and Z direction can be seen in Fig 1.15 and 1.16. The average position error in X-Y direction was 0.0283 m in this case.

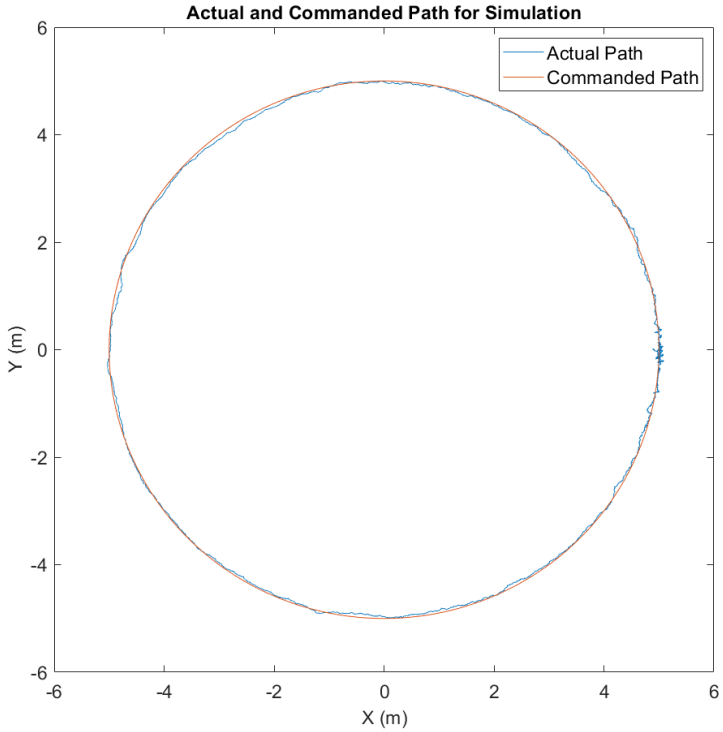


Figure 1.15: Simulation Run 3 - X-Y

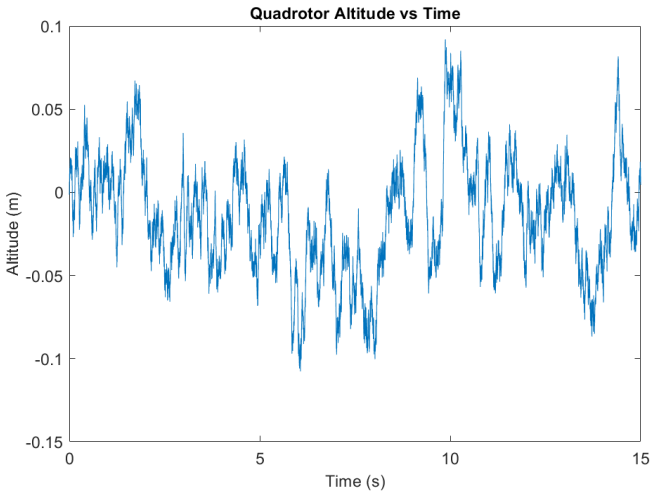


Figure 1.16: Simulation Run 3 - Altitude

Simulation Run 4:

For this simulation run, the gains used were: $\mathbf{k_v} = diag[25, 35, 30], \mathbf{k_p} = diag[195, 195, 100]$ $k_{p,\phi} = 40, k_{d,\phi} = 0.3, k_{q,\theta} = 40, k_{d,\theta} = 0.3, k_{r,\psi} = 1, k_{d,\psi} = 0.3$;

The path followed by quadrotor in X-Y and Z direction can be seen in Fig 1.17 and 1.18. The average position error in X-Y direction was 0.0304 m in this case.

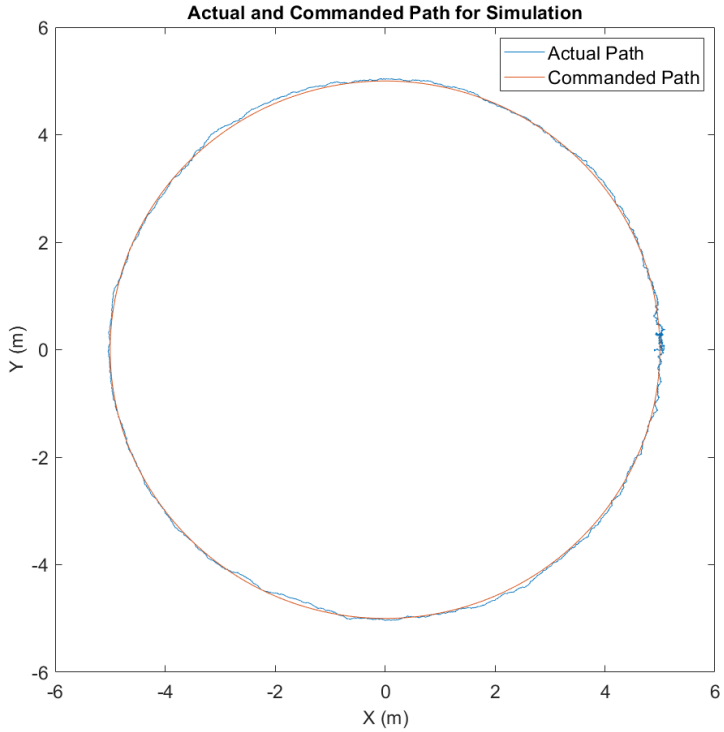


Figure 1.17: Simulation Run 4 - X-Y

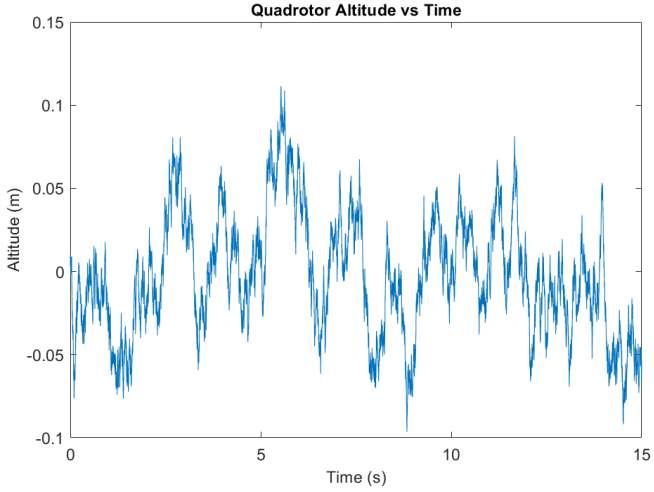


Figure 1.18: Simulation Run 4 - Altitude

Chapter 2

Fully Customized Quadrotor Platform

Teensy 4.0 Development Board (Fig: 2.1) is a 32 bit Arduino-compatible microcontroller development board. It has ARM-Cortex-M7 microprocessor, 1984K flash memory, 1024K RAM, 40 digital I/O pins and 31 PWM output pins. We will be using this microcontroller board as the custom flight controller. We will be taking the inputs from various IMUs and will be using sensor fusion algorithms to estimate the states. Then we will be sending the PWM signals to the ESCs for the speed control of the motors. In order to have robust control over the quadrotor, we need to characterize a few things of the system. The sections below explain various system characterizations performed.

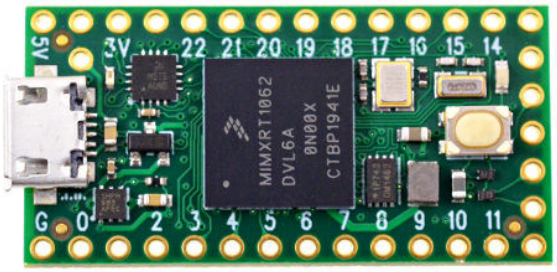


Figure 2.1: Teensy® 4.0 Development Board (Image Source: (4))

2.1 Characterization of Delay in Wireless Communication

In this project, we are focusing on indoor flights and hence we do not have access to the GPS. Also, even if we had access to the GPS, the accuracy of the GPS is in the order of meters, whereas we require mm order precision control of the quadrotor for our dynamic experiments. Hence, the Vicon motion capture camera setup is being used to locate the quadrotor in 3D space. As the position detection is offboard, the wireless communication

between the Vicon camera setup and the quadrotor is a necessity. Because of this, there will be some inherent delay for the position data received from the Vicon camera setup. In order to reduce the effect of this delay, it has to be ensured that the delay is as less as possible.

There are various hardware setups available at our disposal in order to achieve this wireless communication. So in order to determine which hardware setup will provide the best performance for our application, the following hardware setups were analyzed for their different characteristics and are explained in the upcoming subsections:

1. ESP8266 Wifi Module
2. ESP32 Microcontroller
3. XBee S2C

Communication with Desktop PC for benchmark

All the communication devices have their own processors inbuilt to process the data, which might be one of the causes of the delay in the communication. Assuming these processors are not as powerful as the desktop PC processor, the first experiment was done to communicate with the desktop PC in order to benchmark the minimum possible delay over the WiFi router. The communication route was as shown in Figure 2.2.

Figure 2.2: Communication route with Desktop PC

The timestamp data is sent from the Laptop as a UDP packet using a Python script. The UDP is faster than the TCP/IP, which is commonly used in day-to-day internet usage. It does not require any feedback from the receiver client and hence does not need to wait for the communication surety. Due to this, there might be packet losses, but it does not affect our application as the next incoming packet can be used by the UAV. The timestamp data received on the PC is echoed back to the laptop using UDP. The echoed timestamp data is read by the laptop and the current time is noted when the data arrives. This way, the difference between the current time and the sent time is considered the 2

way latency of this communication. The average 2-way latency for this setup was **4.20 ms** with sometimes reaching up to **56.85 ms**.

ESP8266 WiFi Module

The ESP8266 module can transmit and receive the data using WiFi and can also do serial communication with other micro-controllers. This module is used to provide WiFi capability to the microcontrollers which do not have in-built WiFi. Similar to the previous setup, The latency analysis was done for the ESP8266 WiFi module. The module is programmed using Arduino IDE to echo back the received data to the sender IP address immediately. So in this setup also, the timestamp data was echoed back to the laptop and the travel time was calculated using the difference between sent time and receipt time. The average 2-way latency for this setup was **4.93** ms with sometimes reaching up to **213.33 ms**.

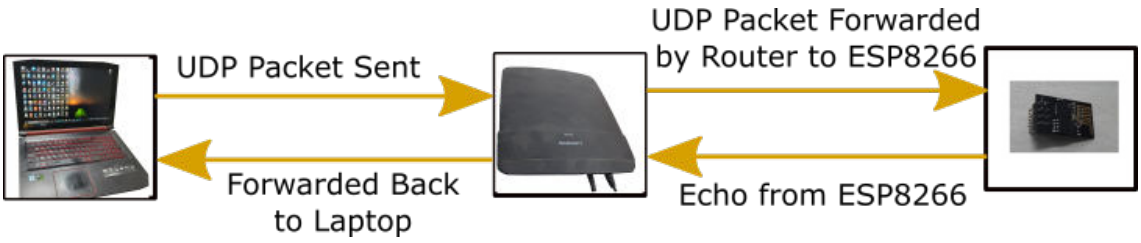


Figure 2.3: Communication route with ESP8266 Module

ESP32 Microcontroller with WiFi

In this analysis, the ESP32 microcontroller's WiFi communication was tested. The ESP32 microcontroller has an inbuilt WiFi feature and can perform computations much more complex than ESP8266. The ESP32 board is programmable using micro-python language. Similar to previous setups, in this setup also the timestamp data was sent to ESP32 over WiFi and it was echoed back by ESP32 to the laptop. The time it took to travel in 2 way was calculated and the average 2-way latency for this setup was found to be **14.38 ms** sometimes reaching up to **152.51 ms**.

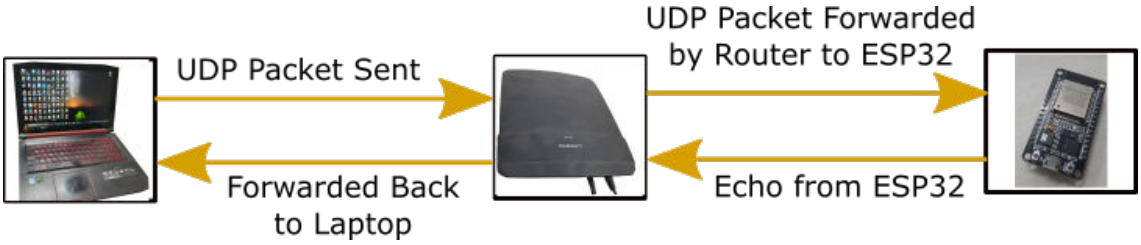


Figure 2.4: Communication route with ESP32 Microcontroller

XBee S2C

The XBee S2C chips works on IEEE 802.15.4 (14) protocol. In this setup, 2 XBee S2C chips are paired with each other and communicate with each other using their own RF signals and do not rely over WiFi router. In the analysis setup (Fig 2.5), the communication happens as per following:

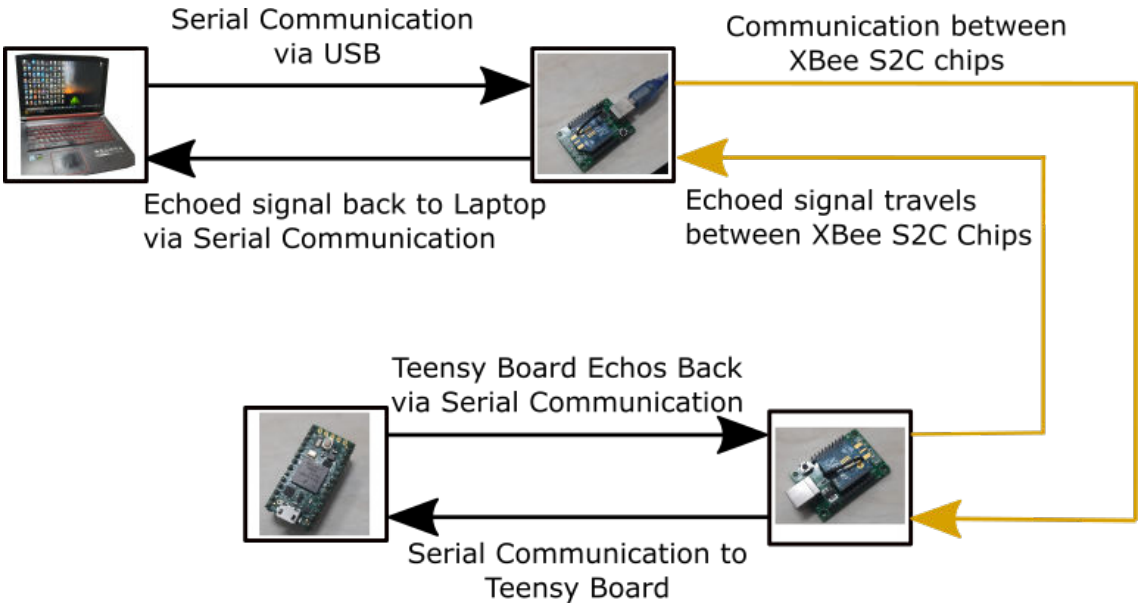


Figure 2.5: Communication route using XBee S2C Chips

1. Laptop sends the timestamp data to one of the XBee S2C chips via USB serial communication.
2. First XBee S2C chip forwards the data to another XBee S2C chip wirelessly.
3. Second XBee S2C chip forwards the data via serial communication to the Teensy board as XBee S2C does not have capability to reply back the received signal.
4. The Teensy board echoes the received timestamp data back to the second XBee S2C via serial communication which then forwards it to the first XBee S2C wirelessly.

5. The echoed signal is received back at the laptop via USB serial communication from the first XBee S2C.

6. Laptop counts the time it took to travel back and calculates the 2-way latency.

With this setup, the average 2-way latency for this setup was **15.13 ms** but was very reliable and did not have any high intermittent latency. Another observation was also made for this setup that the latency increases significantly when the packet size is more than 39 bytes. In our application, we will not be requiring more than 39 bytes of data at a given point of time and hence we can take the advantage of this reliable communication medium.

Final Recommendation for Wireless Communication

After the analysis done above, it was considered that the XBee S2C-chip based setup is most reliable since it does not depend upon the WiFi router and does not have high intermittent latency. Even though it has higher average latency than other setups, it is guaranteed that there will not be any intermittent high delay (which might have very catastrophic consequences). So the complete loop latency analysis with the Vicon camera setup was performed with XBee S2C as wireless communication hardware.

2.2 Complete Loop Latency with Vicon Cameras

In our final setup, the Vicon cameras will be providing the location data to the UAV. After analyzing the standalone wireless communication, it was a time to analyse the complete loop communication of Rigid Body - Vicon - Wireless Communication path. In order to do this, the rigid body with 1 degrees of freedom (Fig 2.6) was attached to a rotary encoder. The communication in this setup (Fig: 2.7) was as per following:

1. The rigid body with markers is rotated at a slow speed and the markers on the rigid body are detected in Vicon cameras

2. The server PC receives the data-frames from the vicon cameras and the tracker software calculates the orientation of the rigid body.

3. The orientation data is forwarded to XBee S2C using USB serial communication.

4. The second XBee S2C receives the data wirelessly and sends it to the Teensy board.

5. As soon as the Vicon data arrives at the Teensy, it calculates the orientation as per the rotary encoders also (considered as no delay-ground truth).

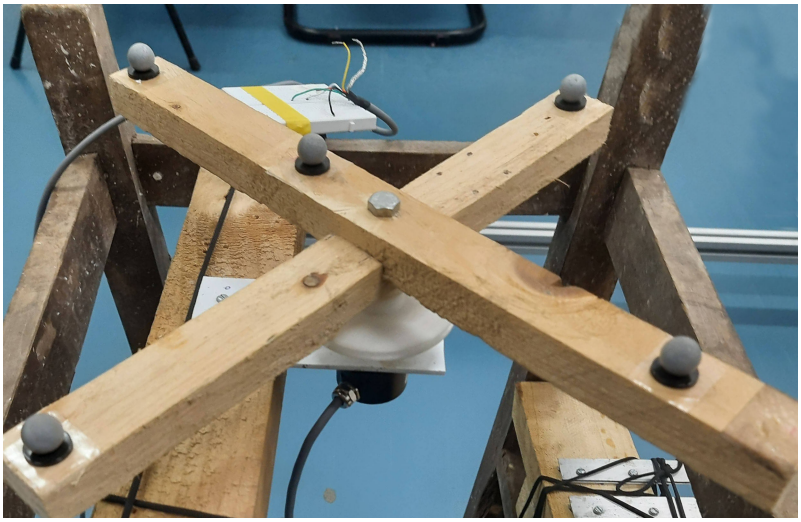


Figure 2.6: 1 DOF Rigid Body with Markers

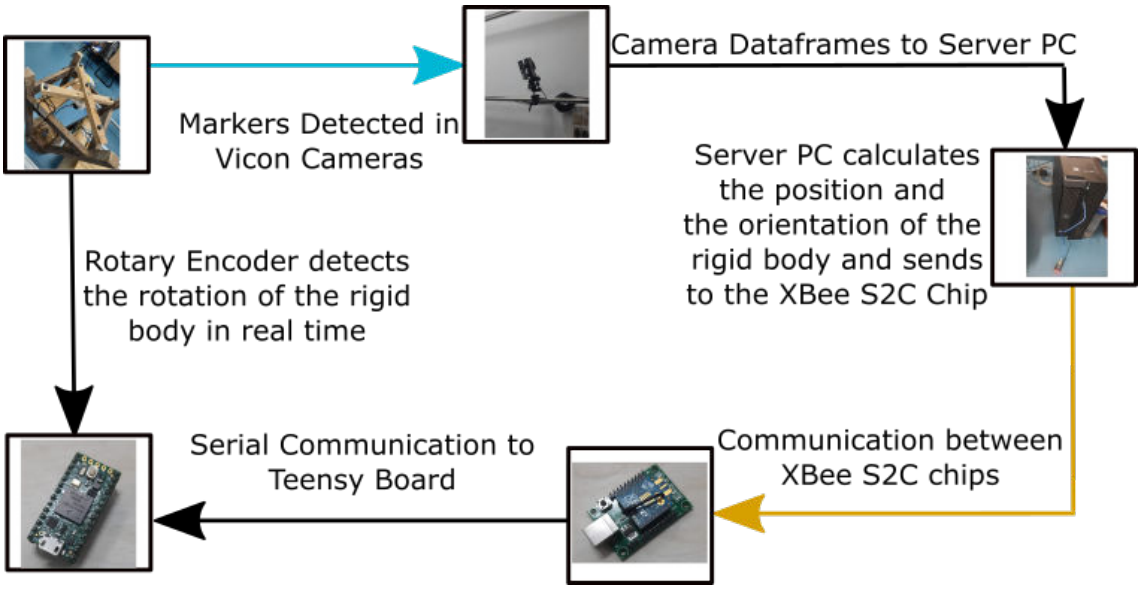


Figure 2.7: complete Loop Latency Estimation

6. The orientation data from both the measurements are stored with the time-stamp

Using the orientation data which is stored using the above method, we can determine the time delay of the Vicon data by analysing the difference between the angle v/s time plots of both the measurement sources (Fig 2.8). Using the interpolation, the linear curve was fitted on the data points to derive the linear relationship of angle v/s time for both the measurement sources. Using this equation, the horizontal distance between 2 lines was calculated and was considered the delay of the complete loop.

The complete loop latency of this setup was found to be **22 ms** for the data length less than 39 bytes (when only orientation data was sent). If more than 39 bytes of data was sent on the wireless communication (when translation data was also sent with higher

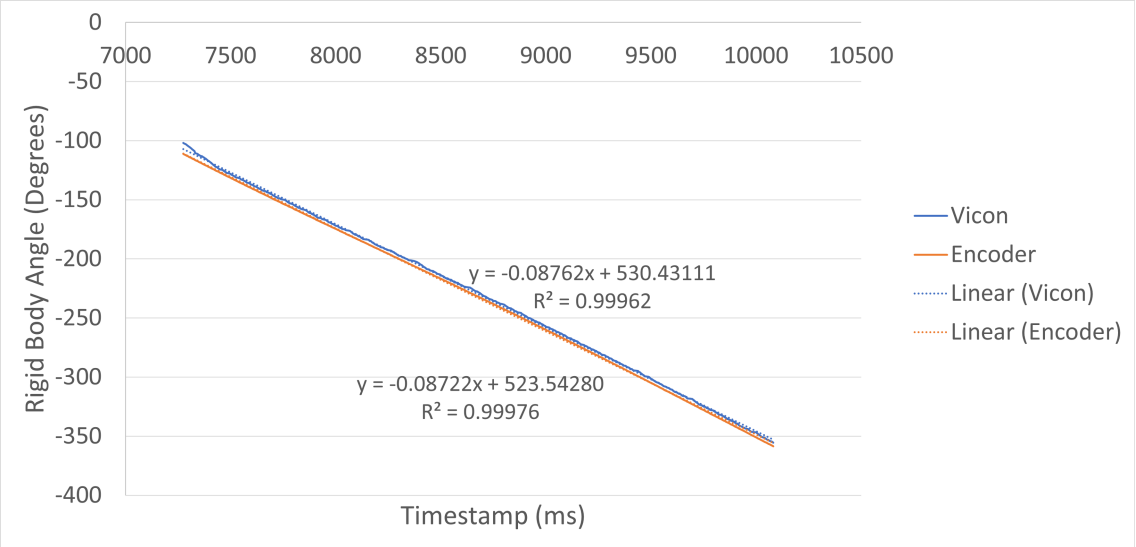


Figure 2.8: Rigid Body Angle v/s Time for Vicon and Encoder measurement

decimal point precision), the latency was **39 ms**. In our application, we will need to communicate the datapacket of less than 39 bytes only and hence we will consider 22 ms as the complete loop latency. Out of this 22 ms delay, 14 ms delay is caused by the Vicon cameras itself (as reported by Vicon Tracker software). So our communication delay is 8 ms only, which seems to be the minimum possible delay with the reliability assurance. So it was finalized to use XBee S2C for further usage.

The raw data used for these analysis can be found at **link** and step-by-step guide on how to do the analysis can be found at **link**.

2.3 PWM Input v/s Thrust Measurement Setup

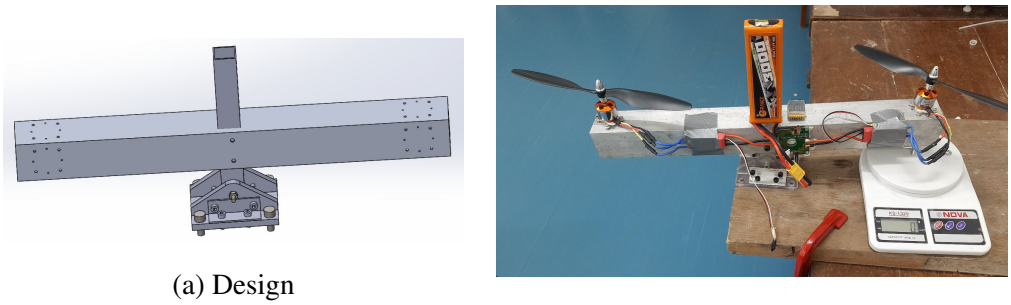
subsection*Design and Working Principle In the multirotor UAV platform, the BLDC motors are used. These motors are driven from the input of the Electronic Speed Controllers (ESCs). ESCs receive the PWM input signal from the flight controller, which is the reference signal for the ESC to rotate the motors at a certain speed. So it is very important for us to know the relation between the input PWM signal and the thrust output of the motors. As per the reference (15), the relationship between the speed of the motors and the thrust generated is:

$$F_i = K_f \omega_i^2 \tag{2.1}$$

where K_f is an empirical constant.

In this relationship also, we require the speed of the motors to derive the thrust value. So still the relationship between the PWM input and the speed of the motors remains unknown. In order to directly know the relationship between the PWM input and the thrust output, the experimental setup was designed and manufactured.

As shown in Fig 2.9a, the thrust measurement setup was designed. It has one horizontal channel, which is supported by the pivot in the center. The motor-propeller set, for which we want to derive the relationship, is attached to the one end of this horizontal channel. A similar set is attached to the other side of the channel also for balancing the mass about the pivot. The free-body diagram of this setup is shown in Fig 2.10. The thrust of the motor causes the moment on the horizontal channel about the pivot. The weighing scale is attached to the other end of the channel at the same distance. Due to this, the force same as the thrust of the motor will be applied to the weighing scale. By recording the weighing scale reading, we can determine the thrust produced by the motor. Fig. 2.9b shows the manufactured thrust measurement setup.



(a) Design

(b) Manufactured Setup

Figure 2.9: Thrust Measurement Tool

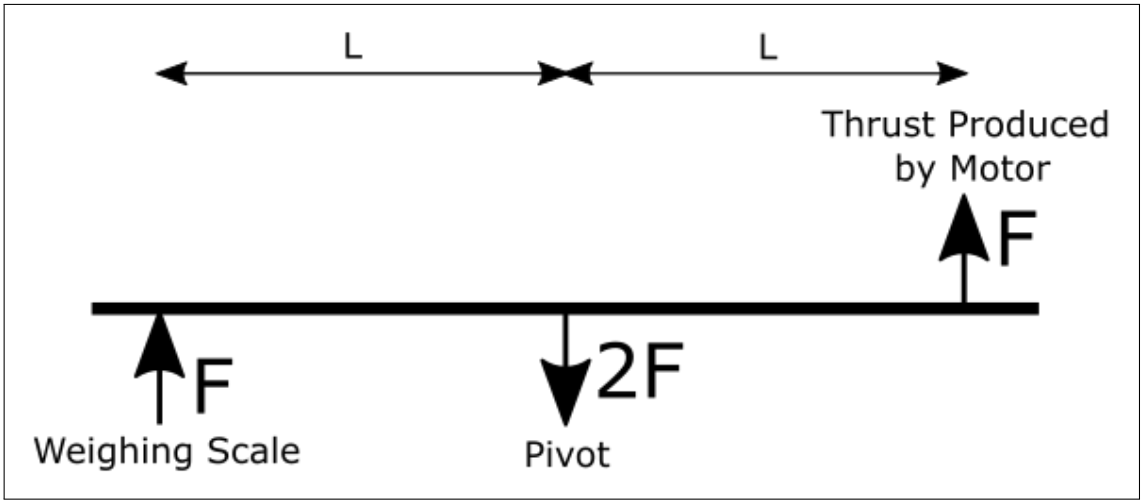


Figure 2.10: Thrust Measurement Tool Free Body Diagram

The same design will be used for reaction moment measurement also, which will be useful in the yaw control of the quadrotor. In order to achieve this, the motors are to be attached on the side of the channel, then the reaction moment generated will be transferred to the weighing scale. With the distance from the pivot to the weighing scale known, the reaction moment generated by motors can be calculated.

subsection*Thrust Measurement Experiment

On the manufactured setup explained in the previous section, the relationship was derived for 2212 1000 KV 13T BLDC motor and 1045 propellers (10 inch length, 4.5 inch pitch). This motor/propeller set was initially used in our UAV flying experiments (explained in the chapter 3). During the experiments, the 12 bit input PWM signal was varied with the interval of 100 each (12 bit signal has a range from 0 to 4095). For each input PWM value, the thrust measurements were taken from the weighing scale. The video of one of the sample reading can be seen at **thrust measurement experiment video link**.

A total of 3 experiments were done on different days to check the repeatability. The result of these experiments is shown in Fig 2.11. We can see that the experiment setup produces repeatable results with a maximum difference of 5.12%. These small variations are due to variations in the battery voltage, which varies in the range of 9.9V to 12.6V based on the charge status. We can also observe that the ESCs start the motors only after 43% of the duty cycle. The maximum thrust produced in this setup is about 700 grams.

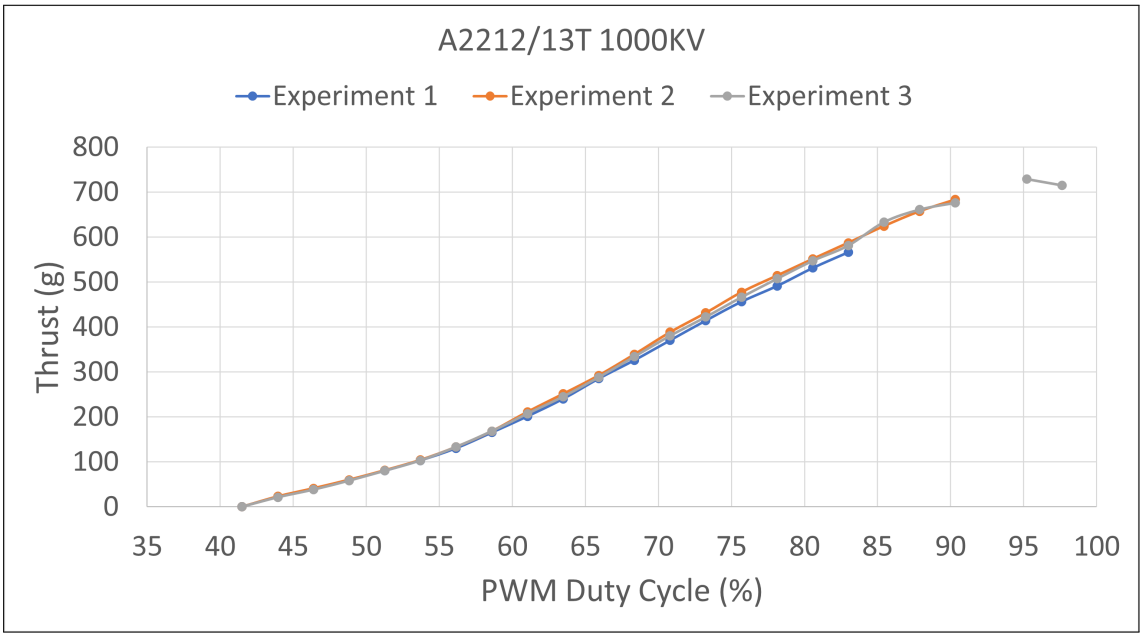


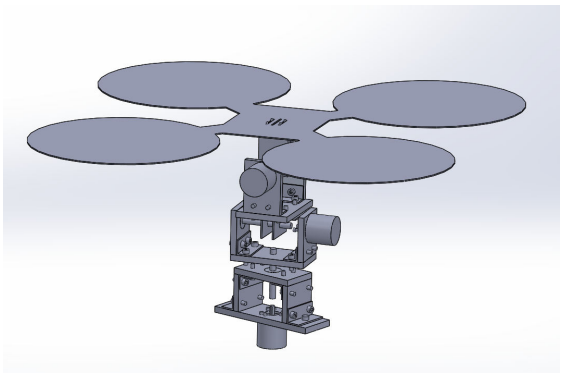
Figure 2.11: Thrust Measurement Readings

With the repeatability ensured from the setup, it can be used to derive the same relationship for different motors. The current experiment was done on the motor-propeller set which is currently being used. We will be using this setup to determine which setup will be best for our application.

2.4 Safe Attitude Testing Platform

The multirotor UAV is 6 degrees of freedom system and typically has sharp propellers rotating at very high speed. In order to test the new control algorithms, it is very dangerous to directly fly the UAV in the free space. This might cause damage to both the environment and the UAV itself. So the testing setup is necessary to safely test the new control algorithms and new controllers.

The multi-rotor aerial vehicles have 2 control loops (Fig 1.10): 1. Inner control loop - which controls the attitude and 2. Outer control loop - which controls the position. The outer loop of the multirotor is operated at a lower rate than the inner loop. The inner loop is responsible for the stability of the vehicle. So in the first phase, we can focus solely on the attitude control without worrying about the position control in order to test the stability of the controller. For this purpose, the safe attitude testing platform with 3 rotational degrees of freedom was designed as shown in Fig. 2.12a with the angle measurement capability.



(a) Design

(b) Manufactured Setup

Figure 2.12: Safe Attitude Testing Platform

This setup has 3 rotational degrees of freedom. So even if the control of the UAV goes haywire, it won't fly to a random place and damage itself. In the setup, the rotational degrees of freedom have the rotary encoders installed. Using these rotary encoders, we can measure the ground truth of the orientation in real-time (reiterating the fact that if we use Vicon data, there will be a delay of 22 ms). These measurements can be used to

validate the IMU-based onboard attitude estimation. The manufactured setup is shown in Fig 2.12b.

The rotary encoder on this setup are relative rotary encoders and they give the angle readings with respect to the angle when they were booted. So we need some calibration method to relate the rotary encoders' readings with the actual readings. In order to do so, the Vicon camera setup was utilized. The Vicon camera setup can provide the rotational angles in the Euler angle form. Similarly, 3 rotary encoders angles can also be considered the Euler angles. So in the calibration setup, the Euler angles readings from both the Vicon cameras and encoders were taken. During the experiment, the setup was rotated manually to different configurations to cover all the possible scenarios of the flight. As we know that the Vicon camera setup has a delay of about 22 ms, the rotated setup was held in a place for some time and the transitional data was removed. This way, the comparison between Vicon cameras and the encoders is be done.

From the readings of the Euler angles, the rotation matrices for both the readings were calculated. Due to small inaccuracies of the manufacturing and inaccuracies while booting the encoders from 0 degree orientations, there will be a small error between both the rotation matrices. Let us denote the rotation matrix derived using Vicon cameras as R_V and the rotation matrix derived using encoder readings as R_C, so

$$R_V = R_E \cdot R_C \tag{2.2}$$

where R_E is the error matrix, which should ideally be an identity matrix. The idea is that after each calibration, we can derive R_E and can pre-multiply it to R_C in order to derive the ground truth of the rotation. In order to quantify how good this experimental setup is, the R_0 matrix was calculated.

$$E = R_V \cdot R_C^T - I \tag{2.3}$$

where E is another error matrix which ideally should be a 0 matrix. From R_0 matrix, we can determine how close the values are to the 0 and hence quantify how good our calibration is. This error matrix is plotted as shown in Figure 2.13. We can observe that all the matrix element values are staying within 0.1 limit. Each step that is seen in the plot corresponds to a different orientation of the rigid body. The values are changing with each new orientation, but still remaining less than 0.1. The primary setup was built as proof of concepts and was not manufactured accurately. So these small errors can be considered manufacturing errors, which should be vanished in future manufacturing iterations.

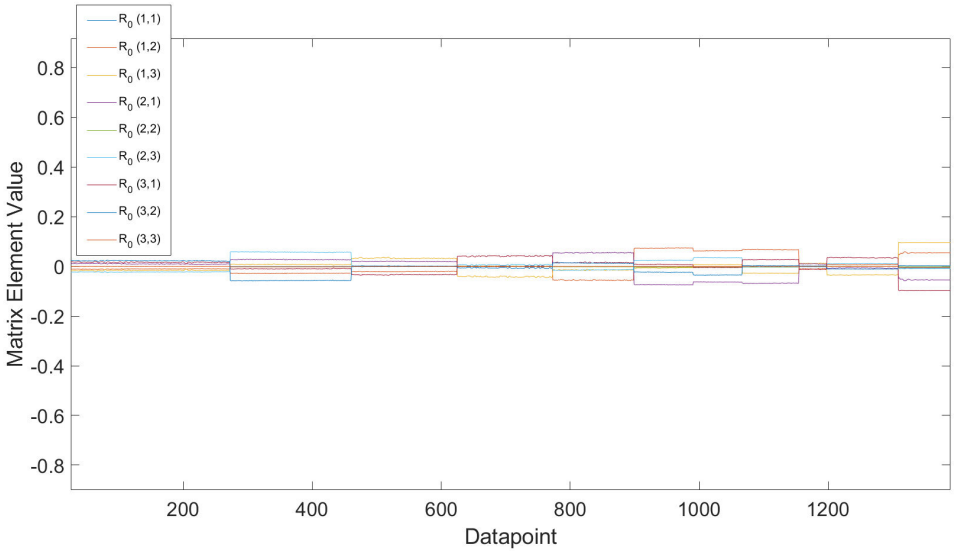


Figure 2.13: R_0 Matrix Elements

Chapter 3

Commercially Available Flight Controller based Quadrotor

The Pixhawk flight controllers were developed by the Computer Vision and Geometry Lab of ETH Zurich (16) and then became an independent open source project. These controllers are based on the PX4-Flight Management Unit. In this project, We are initially using this readily available flight controller. It has control algorithms and sensor fusion algorithms built-in. We are trying to achieve the best possible control possible with the Pixhawk flight controller and will analyze its limitations. In the second step, we will be using our custom-built Teensy-based flight controller to overcome the limitations of the Pixhawk.

Figure 3.1: Pixhawk Flight Controller (Image Source: (5))

Fig 3.2 shows the photograph of the Pixhawk based flight controller assembled by us. Various flight experiments were performed with the Pixhawk based quadrotor to analyze various characteristics. The details of these experiments are explained in the upcoming sections.

Figure 3.2: Pixhawk Based Quadrotor assembled at INDUS Lab, IIT Bombay

3.1 Pixhawk Flight Modes

Manual Flight

The first experiment was done with the Manual/Stabilised flight mode of the PX4 (17). In this flight mode, the controller tries to stabilize the multi-rotor by controlling the inner loop. The roll/pitch sticks (Fig 3.3) of the radio controller give the commands to change the angle of the multi-rotor, through which we can control the X-Y directional movement of the vehicle. The throttle stick of the RC transmitter controls the speed of the motors.

Figure 3.3: RC Transmitter

In this flight mode, there is no autonomous feedback to the controller regarding its position and hence it has to be controlled manually. In the ideal conditions, removing the command from the pitch/roll stick should make the multi-rotor hover in the same place. But due to the imbalance of masses on the quadrotor, variations in the speeds of the motors for a given input command, aerodynamic forces, etc lead to drift of the quadrotor in X-Y direction as well as the altitude direction.

The video of this experiment can be seen at **Manual Flight Video Link**. As explained previously, due to real-life non-ideal conditions, there was difficulty in holding the flight at a constant hover location, however, it was controlling the attitude of the quadrotor perfectly. Also, while trying to control the x-y position using the roll/pitch stick of the RC transmitter, we need to tilt the quadrotor. This leads to a decrease in the effective upward thrust and we need to compensate for this by increasing the throttle a little. This compensation is also very difficult to perform in the manual flight mode.

This experiment gave us confidence about the stability of the quadrotor to try out other more automated flight modes.

Altitude Hold Flight

In the altitude mode of the PX4 (18), the vehicle tries to hold the constant altitude when the throttle stick is centered. In order to control the altitude precisely, the controller needs feedback about its altitude. This is achieved in our case using a single point range finder LiDAR sensor (Fig 3.4). This sensor has the accuracy of 1% in the 0.3 to 6 m range and 2% in the 6 to 12 m range (19).

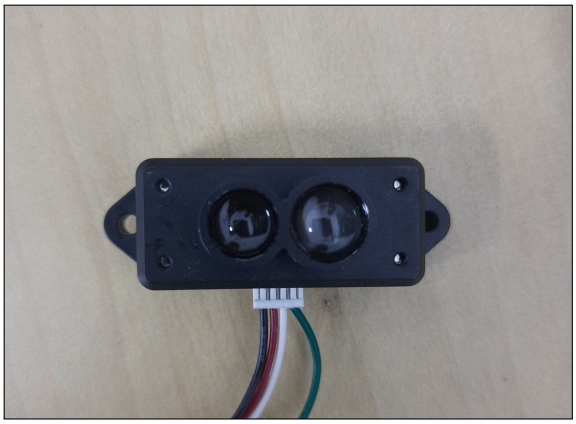


Figure 3.4: LiDAR Sensor

In order to verify that the sensor is able to provide reliable measurements, a simple experiment was performed. The sensor was kept at a certain measured distance from the object and sensor readings were logged over time. This gave us an estimate about how

much does the sensor measurement vary for the object at the constant distance and how accurate the average reading is to the actual distance. The plots of this analysis can be seen in Fig 3.5 and Fig 3.6. We found that the variation was within 2% for most readings with rarely reaching up to 7%. Also, the distance measured by the sensor was not equal to the actual distance, however, it had a linear relationship. The relationship was found to be $R_{LiDAR} = 0.894 \cdot R_{actual} + 11.393$. So we can get predictable actual distance reading with LiDAR sensor using this relationship.

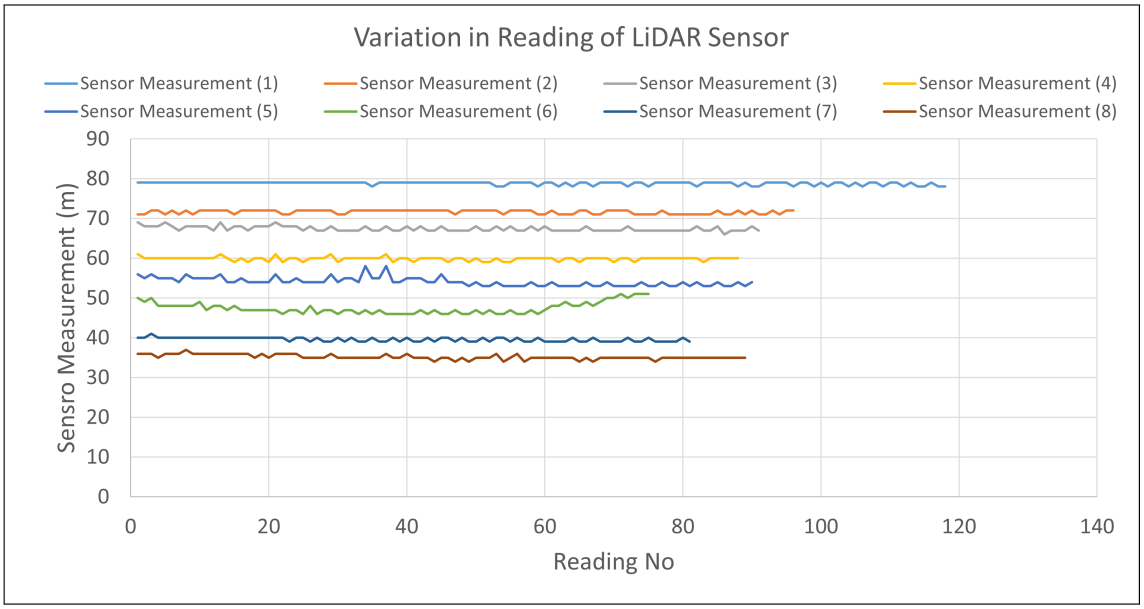


Figure 3.5: LiDAR Measurement Variations for an Object at Constant Distance

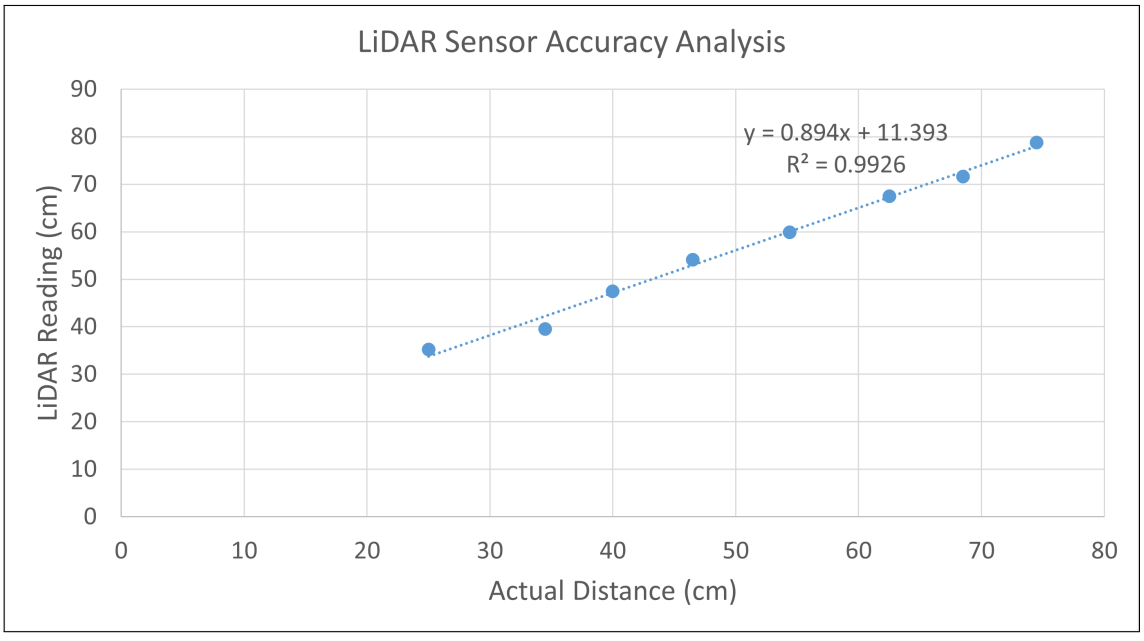


Figure 3.6: LiDAR Measurement relation with the Actual Dimension

After getting the confidence about the sensor measurement, the flight experiment with the altitude hold mode was performed. After the takeoff with the manual flight mode, the throttle of the RC transmitter is centered and then the flight mode is switched to the altitude hold mode. The pitch/roll sticks still control the X-Y direction position by controlling the attitude of the quadrotor. With the throttle stick, the altitude set-point is changed to increase or decrease the altitude. This flight was much more stable than the manual flight mode. In this flight also, the quadrotor was drifting in the X-Y direction as there is no feedback regarding its position, however, it was controllable easily with the RC transmitter as the altitude was being controlled automatically. The video of the flight with this mode can be seen at **video link**.

Position Hold Flight

As the altitude flight was successful in controlling the position in verticle direction, we could now try the position hold flight mode with the Vicon camera setup in the more constrained space of the lab. The position hold flight mode of the PX4 (20) holds the current position when the RC transmitter sticks are centered. The roll and pitch sticks control the accelerations in the horizontal directions similar to a car's accelerator pedal. The throttle stick controls the ascend or descend of the quadrotor.

Analogous to the altitude flight mode, we will need the accurate sensing of the position in all 3 directions in order to successfully achieve the controlled position hold. So for this purpose, Vicon motion capture camera setup was utilized. As finalized in the section 2.1, the Vicon data was communicated to Pixhawk using XBee S2C hardware. The communication with the PX4 has to be done using MAVLINK protocol and hence it is not possible to send the position data directly from the server PC to the Pixhawk. Also, the PX4 utilizes NED (North East Down) body reference frame, which is different than the reference frame of the Vicon cameras. So for communicating Vicon position data to the Pixhawk, another computer running Robotic Operating System (ROS) was required to forward the position data to the PX4. The communication happens in the following manner (Fig 3.7):

1. The markers attached on Pixhawk based quadrotor are detected by Vicon Cameras.
2. The Vicon data is received at the server PC and is processed using Vicon Tracker software.
3. The processed Vicon data is streamed using Vicon Data Stream SDK on the localhost of the server PC

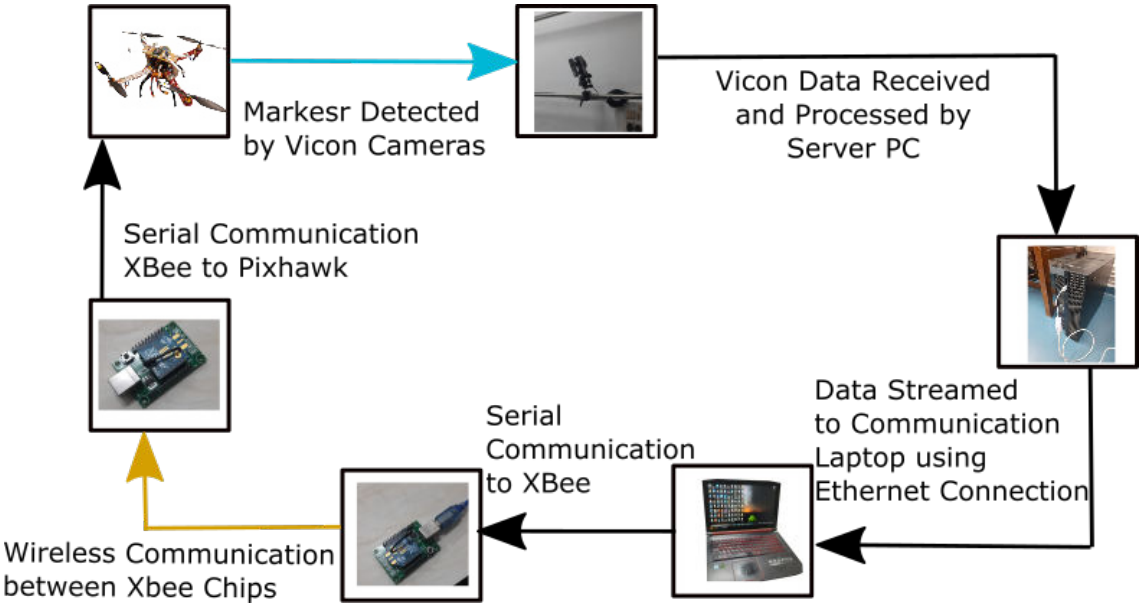


Figure 3.7: Position Feedback Communication for Pixhawk

4. The data available on the localhost of the server PC is then fetched by the /vicon node running through vicon_bridge package and is published to /tf topic (Fig. 3.8).

5. The /mavros node running on the same communication laptop subscribes to the /tf topic and receives the position data of the vehicle.

6. /mavros node transforms the coordinates into NED reference frame and creates necessary message as per the MAVLINK communication protcol and sends it to XBee S2C using USB serial communication.

7. The processed position data is forwarded to another XBee S2C chip.

8. The Pixhawk receives the data from the second XBee S2c using serial communication and uses it in its EKF2 datafusion algorithm to estimate its position.

Using the communication medium described above, the flight experiment was performed. In the experiment, the constant position was held for about 5-7 seconds before the oscillations were starting in the quadrotor. The video of the experiment can be seen at the **Link**. The reasons for oscillations were thought to be:

1. As the protective nets are yet to be installed, the propeller guards were put on the quadrotor which made it very heavy. Because of the heavy weight and the output PWM signal being limited to a certain value, the input to the ESCs was getting saturated when the controller tried to increase the speed of one of the motors.

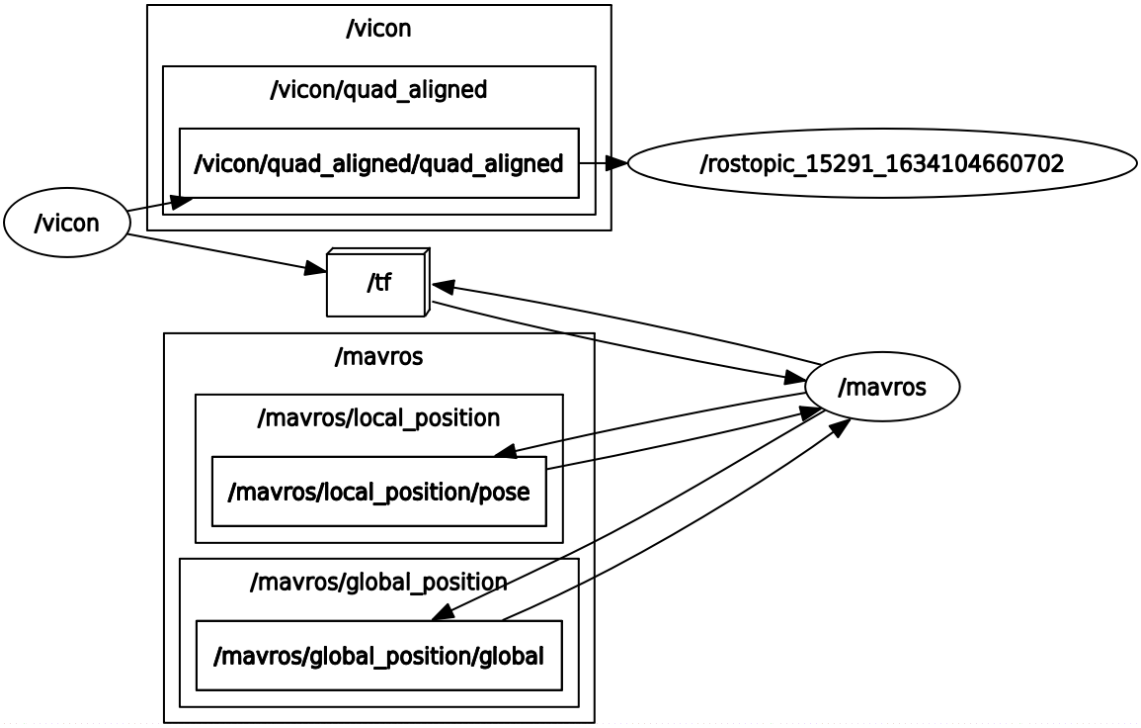


Figure 3.8: ROS Graph of Communication Laptop for Position Hold

2. One of the motors had a little worn-out bearing, which made control in one of the directions difficult.

3. The setpoint was fixed manually, which made it difficult to understand what setpoint is available on Pixhawk at any given point of time.

4. As it was the first experiment inside the lab, the altitude was increased very conservatively, which made the ground effect also very prominent.

Offboard Control

In order to overcome the shortcomings of the position hold mode, the offboard mode of the PX4 was utilised (21). In this mode, the setpoint is streamed to the Pixhawk from the communication laptop using the same MAVLINK protocol. In the offboard mode, the setpoint of the position can be changed very precisely. In this mode also, the position data of the Vicon camera setup was streamed to Pixhawk using the communication path explained in the section 3.1. On top of the position data, the setpoints were also streamed to Pixhawk from an additional node /setpoint_node_... (Fig 3.9) via /mavros node.

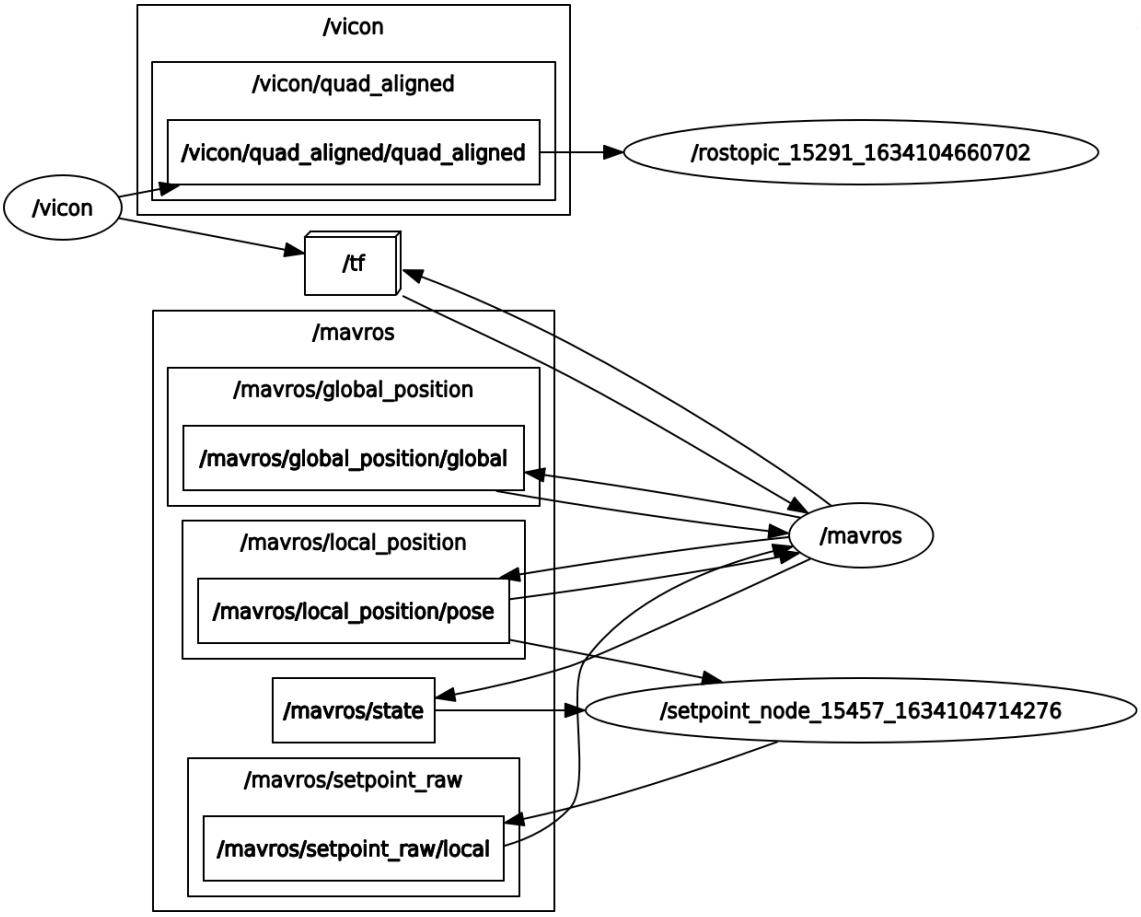


Figure 3.9: ROS Graph of Communication Laptop for Offboard Mode

Experiment 1

The experiment using this method was performed. The video of the experiment can be seen at **Link**. The path followed by the UAV can be seen in Fig 3.10. The altitude followed by the UAV can be seen in Fig 3.11. The observations from this experiment are as below:

- The control of the quadrotor was much more precise than any of the methods described previously
- The variation in X-Y direction was within a 5 cm envelope. This might be due to an imbalance of masses on the quadrotor.
- Sometimes, the quadrotor was dropping suddenly in the verticle direction. After the analysis, it was found that this was due to intermittent loss of communication (for a fraction of seconds) from the ground station laptop. Because of this, the Pixhawk was trying to send the quadrotor to the home location (0,0,0) while the

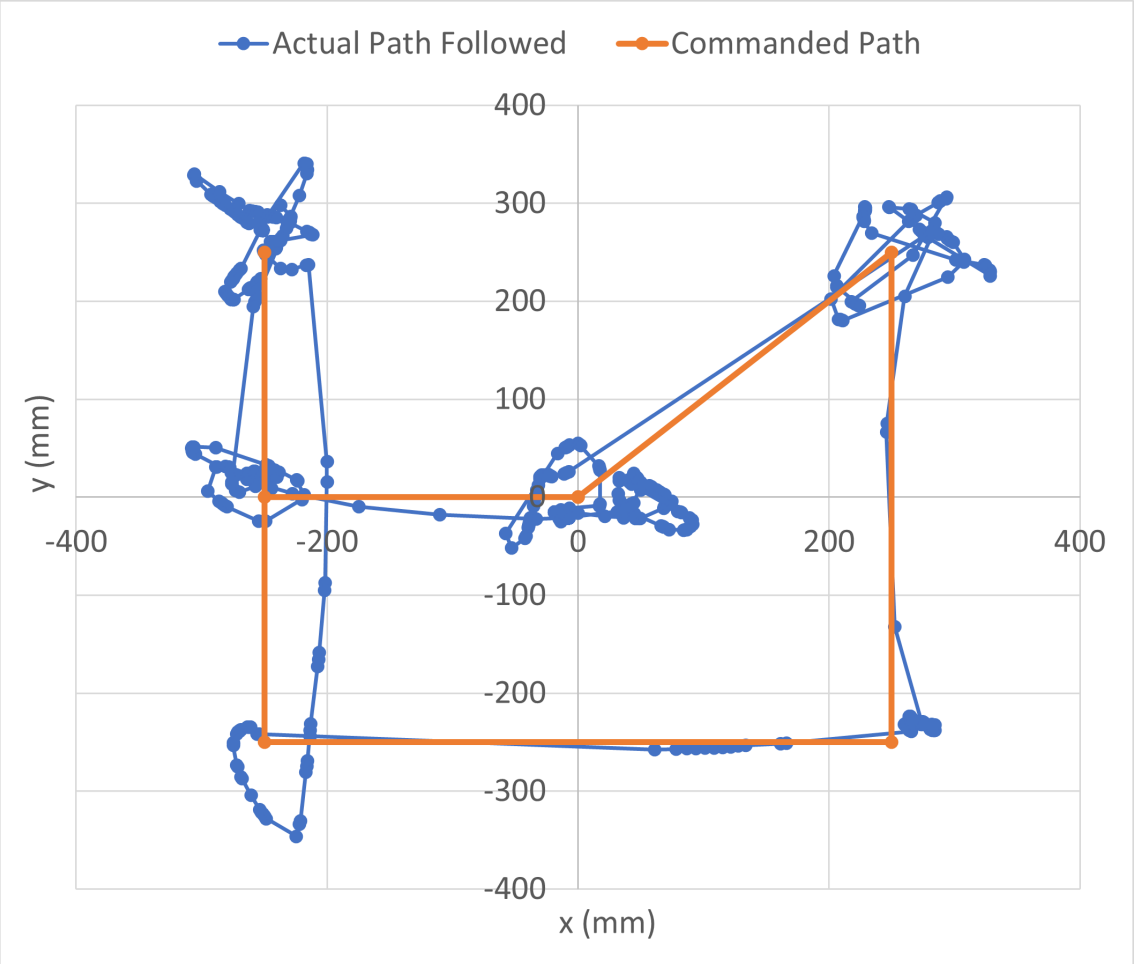


Figure 3.10: Offboard Experiment 1: Path Followed by UAV in X-Y direction

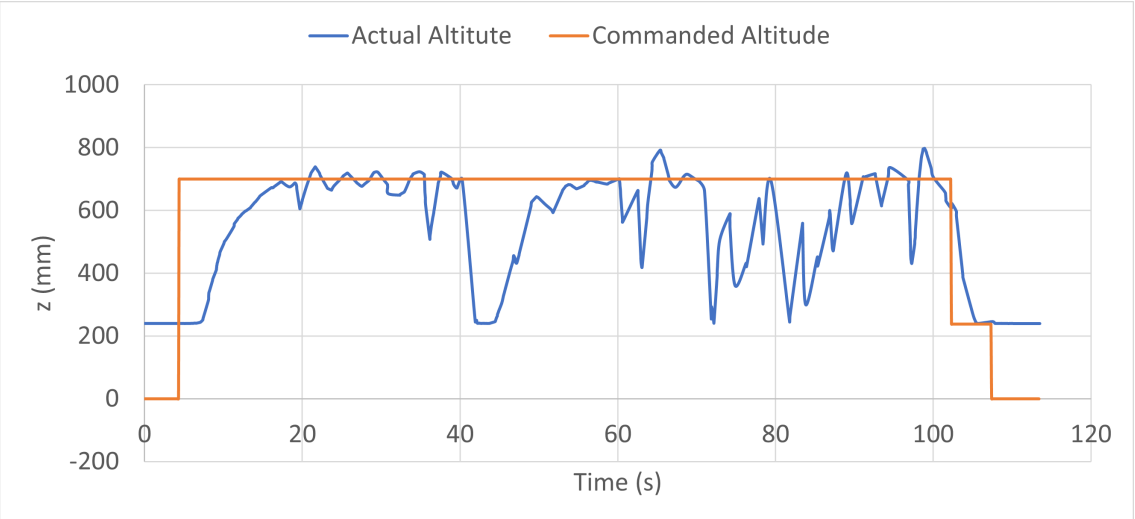


Figure 3.11: Offboard Experiment 1: Altitude followed by UAV

communication was lost and was trying to reach the commanded location as soon as the communication was regained.

Experiment 2

In order to overcome the problem of communication loss in the previous experiment, a few changes were made in the communication. One of the major changes which affected the performance was the increment in the baudrate from 115200 to 230400. Due to these changes, we could perform a flight without any sudden drops, which were observed in the previous experiment. In this experiment, the UAV was commanded to follow a path in a square box of 1 by 1 meter starting and ending at the origin. The path followed by the UAV can be seen in Fig 3.12. The altitude followed by the UAV can be seen in Fig 3.13. The video of the experiment can be seen at **Link**.

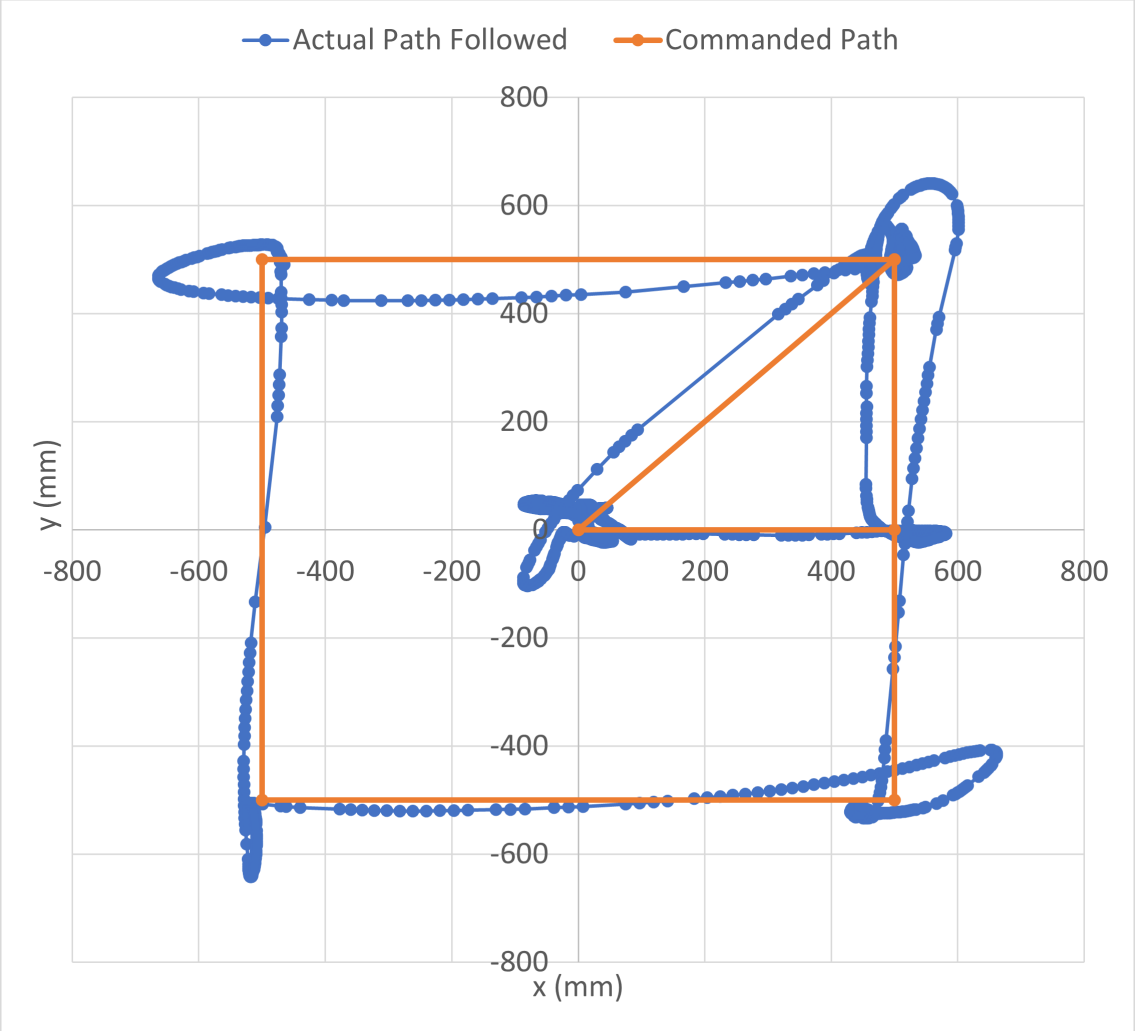


Figure 3.12: Offboard Experiment 2: Path Followed by UAV in X-Y direction

The aim of this project is to do dynamic experiments using the aerial robotic platform, such as impacting a cable suspended mass attached to the UAV with an external object, applying a force on the wall using an arm mounted on the UAV, etc. These kinds of experiments require not only the position control of the UAV but also the lower level

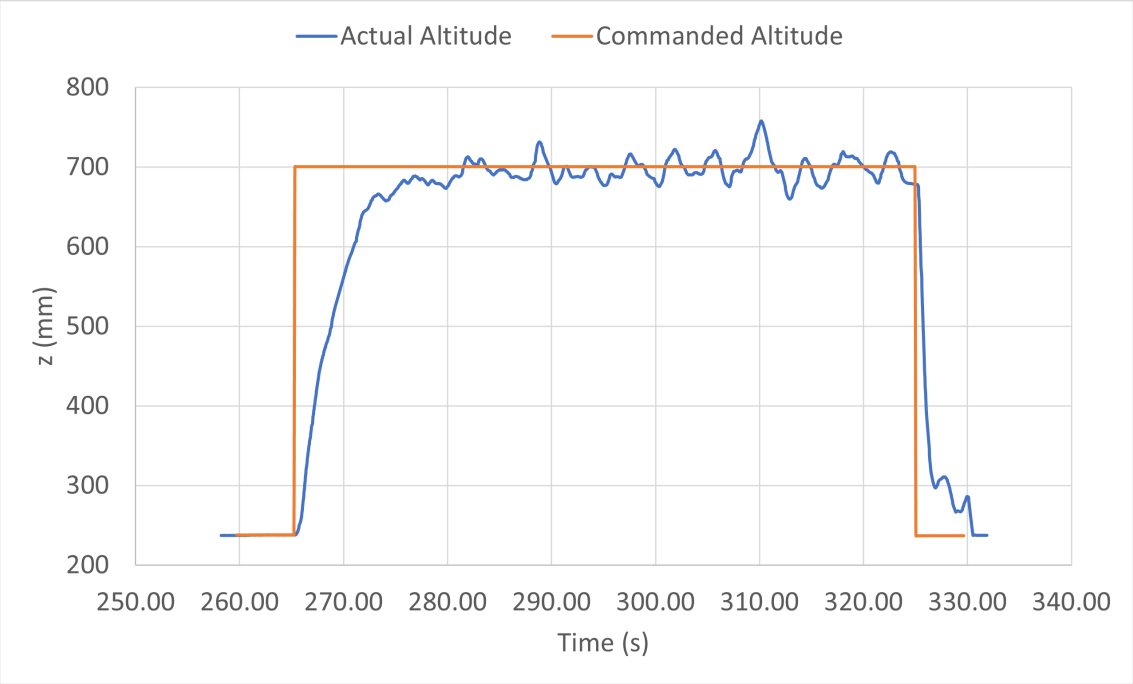


Figure 3.13: Offboard Experiment 2: Altitude followed by UAV

attitude control. As described in previous experiments, the position can be tracked using the PX4 inbuilt capability, but controlling the attitude with that will not be possible with the ready-made code of the PX4. In order to achieve the required tasks, the offboard mode with the outerloop running on an external system is utilized.

In the Offboard outerloop code, some calculations regarding the UAV's control are done on the system which is not onboard the UAV. For these experiments, the Linux system running Robotic Operating System (ROS) is used to calculate the desired attitude and total thrust commands for the UAV. The control diagram of the system for offboard outerloop control is shown in Fig. 3.14.

In the offboard system, the calculation of the required thrust is first done in physical unit (N). However, the PX4 offboard mode requires the command in the form of relative thrust (value from 0 to 1). The relationship between the relative thrust and the physical thrust is dependant on the motors, ESCs, propellers used and the battery voltage. In order to determine this relationship, thrust calibration experiments were done for various combinations of motors, ESCs and propellers. The detailed explanation about these experiments is described in the section 3.2.

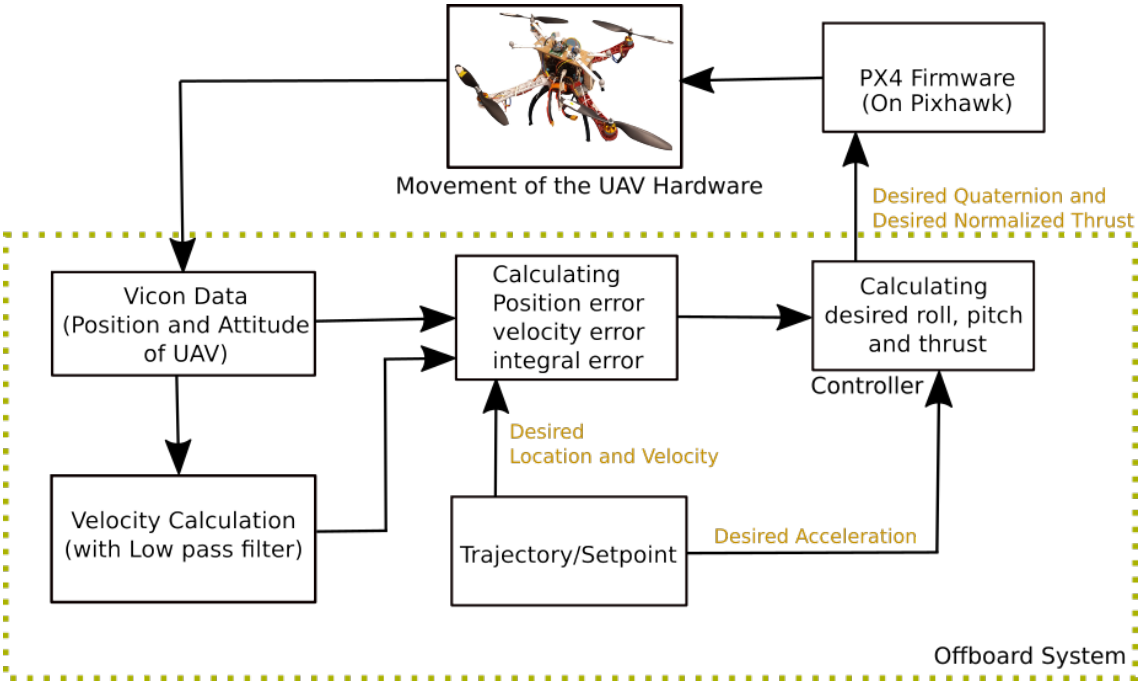


Figure 3.14: Offboard Outerloop Control Schematic

3.2 Thrust Calibration

There is an inherent non-linearity between the PWM signal sent by the flight controllers and the thrust generated by motors due to the dynamics of the system. So we will not be able to get the linear relationship between command thrust (0 to 1) and the physical thrust if this non-linearity is not compensated. In order to overcome this problem, the PX4 firmware has the parameter called "THR_MDL_FAC". This parameter is used to model the non-linear relationship between motor control signal (PWM) and static thrust. The model is: $rel\_thrust = factor * rel\_signal^2 + (1 - factor) * rel\_signal$. Here rel_thrust is normalized thrust between 0 and 1 and rel_signal is the relative motor control signal between 0 and 1. (22).

In order to empirically determine this parameter for our setup, an experiment was conducted. As shown in Figure 3.15, one of the motors (with the propeller) of the UAV is attached to the thrust measurement setup. The UAV is put on an almost horizontal surface. Now using the offboard attitude control, the input thrust command can be varied and we get the corresponding thrust reading on the thrust measurement setup. The attitude command which needs to be sent to PX4 has to contain both the desired thrust and desired attitude data. The UAV is resting on the test setup. If a certain desired attitude is commanded to the UAV, it will try to reach that attitude. Hence it will constantly produce non-equal thrust in 4 motors and the relationship derived will be invalid. In order to avoid this problem from occurring, the current attitude data is read from the PX4 itself (using

/mavros/imu/data topic) and is fed back as the command attitude. Using this methodology, an equal amount of thrust will be generated for all 4 motors and the thrust generated by the motors will depend only on the input thrust. Figure 3.16 shows the ROS graph of such a setup.

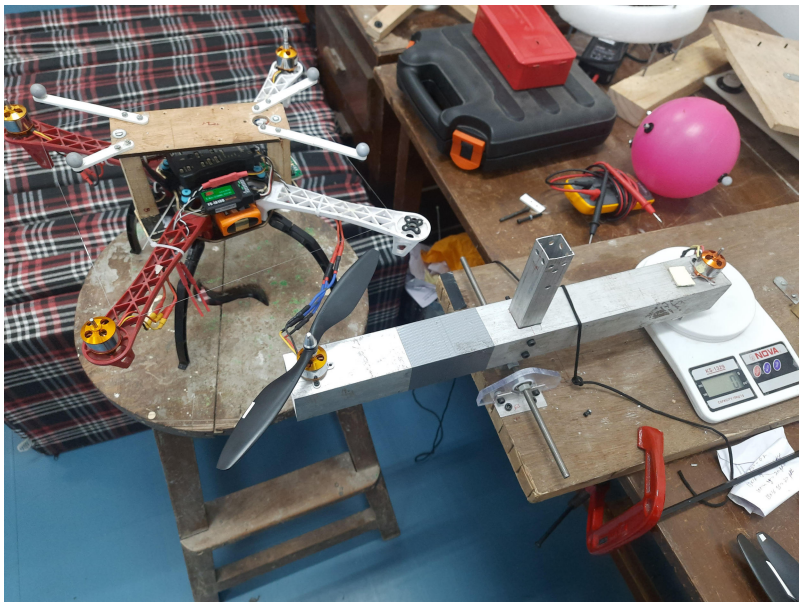


Figure 3.15: UAV on Thrust Measurement Setup

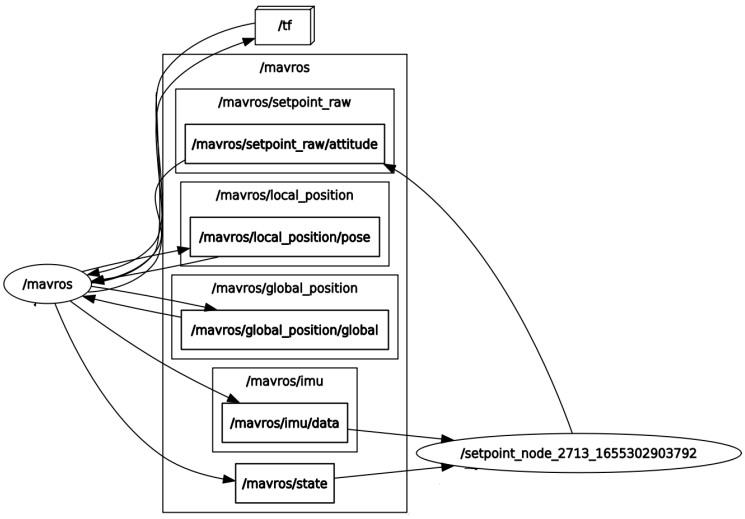


Figure 3.16: ROS Graph for Thurst Measurement Experiment

Using the experimental setup explained above, the input command thrust was varied from 0.0 to the input value till we get about maximum of thrust from the motor (around 600 g). This experiment was done for various THR_MDL_FAC values and linear trendline was plotted over the readings. The THR_MDL_FAC value corresponding to the best

linear fitting curve was considered, which came out to be 0.25. The plots of these various experiments and their trendlines are shown in Figure 3.17.

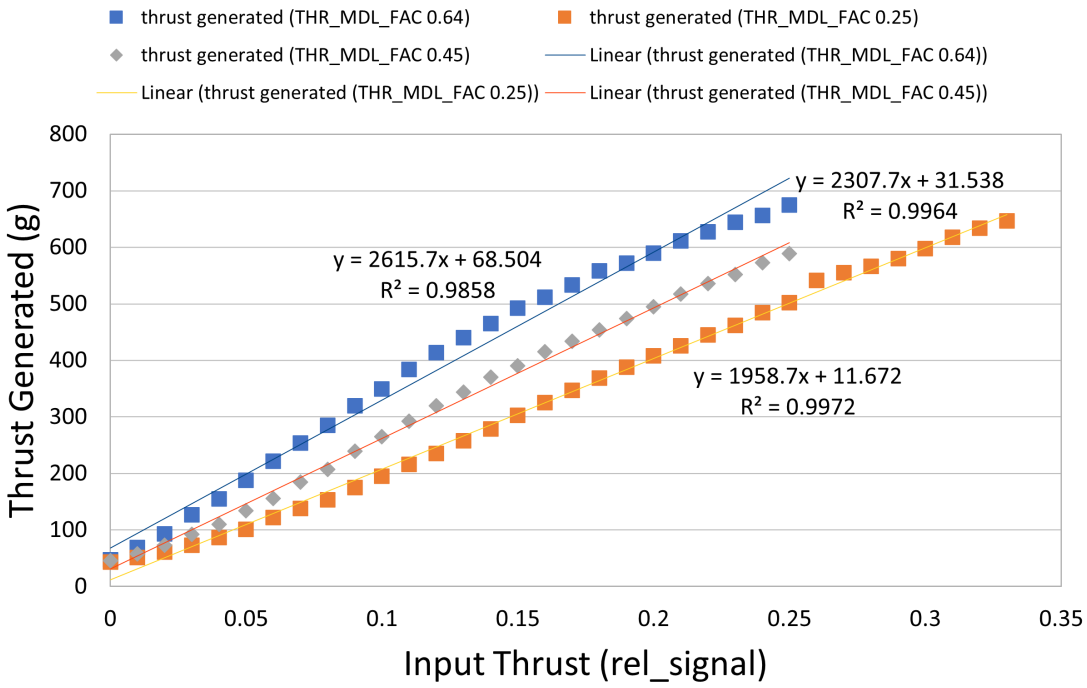


Figure 3.17: Experiment to determine THR_MDL_FAC parameter

After changing this parameter on PX4 firmware, the input thrust signal to output thrust is almost linear ($R^2 = 0.9972$). So for various component combinations, the linear relationship between the input command thrust to the physical thrust can be inverted to find out the desired input command thrust. In the following sections, we will explore this relationship for various ESC/Motors/Propellers setups.

Generic Motors with SimonK ESC

For this setup (22012 1000 kV motors with SimonK ESCs), the thrust measurement experiment was done for all 4 motors to ensure repeatability. As shown in Figure 3.18, we can see all 4 motors are having an almost identical relationship. The small drop in thrust generated is related to the drop in the battery voltage. These small uncertainties can be taken care of by the integral term used in the PID controller of the offboard outerloop. After the repeatability was confirmed, the thrust in N corresponding to 4 motors combined was plotted as shown in Figure 3.19.

The relationship derived using this experiment is:

$$Total\ Thrust = 26.91 \cdot rel\_signal + 1.427 \tag{3.1}$$

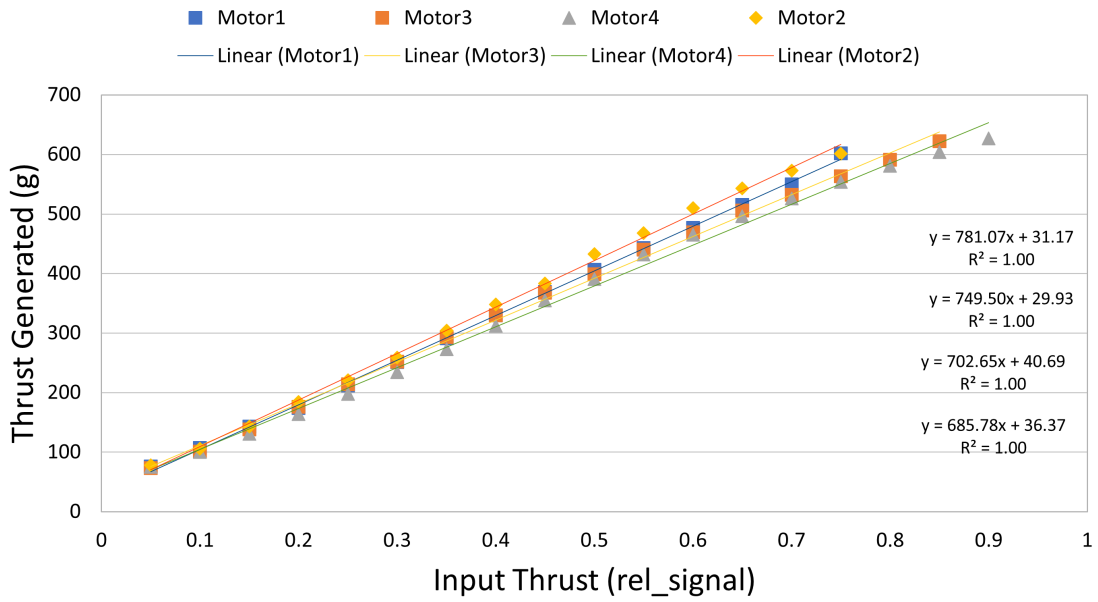


Figure 3.18: 2212 1000 kV motors with SimonK ESCs

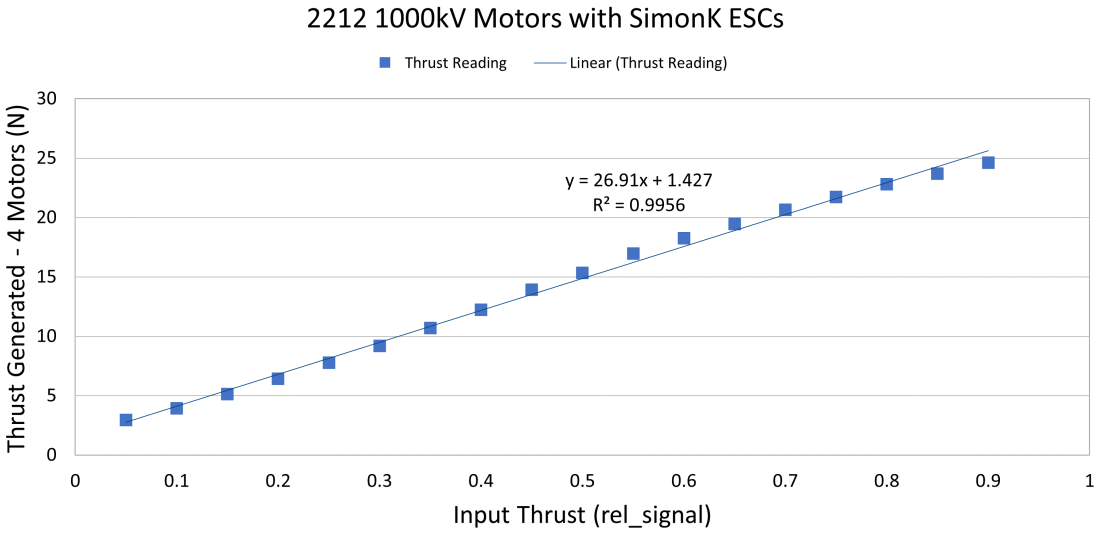


Figure 3.19: 2212 1000 kV motors with SimonK ESCs

Hence, in order to calculate the required input rel_signal (command thrust), the above relationship is inverted:

$$rel\_signal = \frac{Total\ Thrust - 1.427}{26.91} \tag{3.2}$$

Using the above equation, the command thrust is calculated in the control algorithm based on the required total thrust value.

Emax Motors with SimonK ESC and Carbon Fibre Propellers

This experiment was done with the Emax Motors (2213 935 kV) with SimonK ESCs. These motors were expected to have a better response to input and were expected to have better thermal properties. The ESCs were kept the same to see the effectiveness of the motors only. The propellers tried were carbon fiber propellers of 1045 size. The relationship derived using this experiment is (Fig. 3.20):

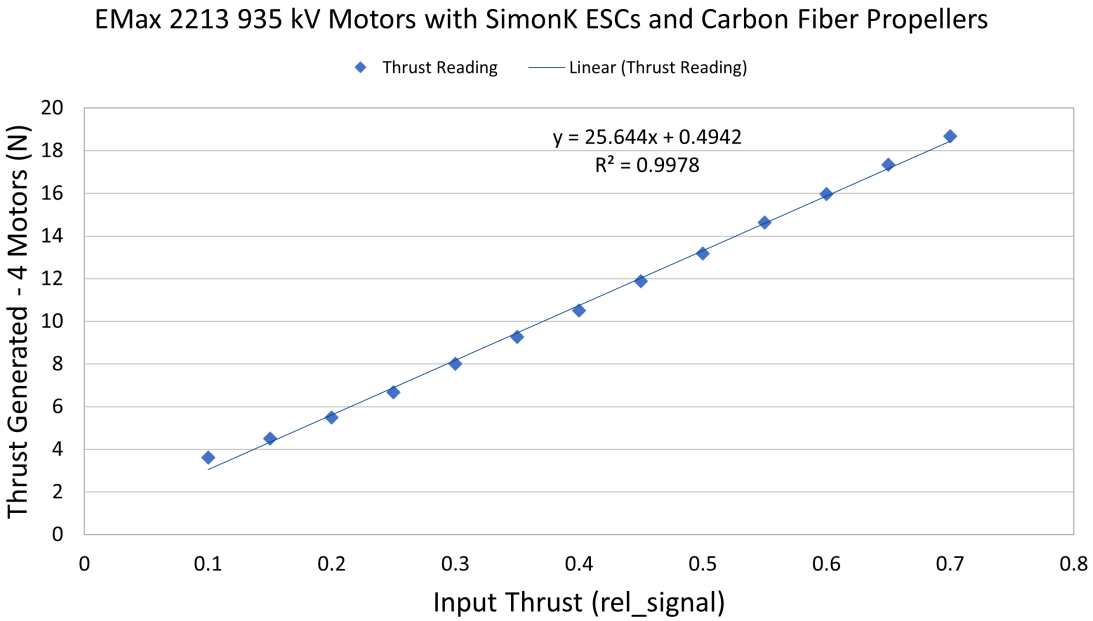


Figure 3.20: Emax Motors with SimonK ESC and Carbon Fibre Propellers

$$Total\ Thrust = 25.644 \cdot rel\_signal + 0.4942 \tag{3.3}$$

Hence, in order to calculate the required input rel_signal, the above relationship is inverted:

$$rel\_signal = \frac{Total\ Thrust - 0.4942}{25.644} \tag{3.4}$$

Emax Motors with SimonK ESC and Plastic Propellers

The carbon fiber propellers were used with the Emax motors as they were expected to give better performance. But they were not fitting properly on given motors and were causing vibrations for certain frequencies. Hence, the carbon fiber propellers were replaced with plastic propellers which came with the motors. The relationship derived using this experiment is (Fig. 3.21):

$$Total\ Thrust = 22.379 \cdot rel\_signal + 1.5652 \tag{3.5}$$

Hence, in order to calculate the required input rel_signal, the above relationship is inverted:

$$rel\_signal = \frac{Total\ Thrust - 1.5652}{22.379} \tag{3.6}$$

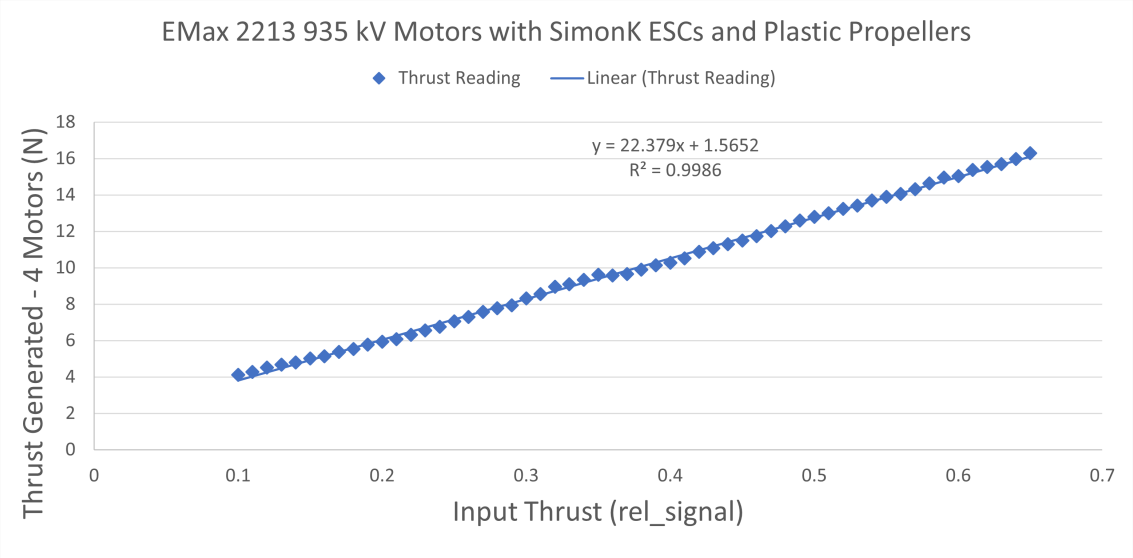


Figure 3.21: Emax Motors with SimonK ESC and Plastic Propellers

Emax Motors with Emax BLHeli ESC and Plastic Propellers

After improving the performance with the Emax Motors, in order to have even more accurate control, the ESCs were changed. These ESCs use the BLHeli protocol, which also supports the oneshot signal. So the control frequency (from the flight controller to the ESCs) can be much higher (max 4000 Hz) (23). The relationship derived using this experiment is (Fig. 3.22):

$$Total\ Thrust = 40.422 \cdot rel\_signal - 2.6821 \tag{3.7}$$

Hence, in order to calculate the required input rel_signal, the above relationship is inverted:

$$rel\_signal = \frac{Total\ Thrust + 2.6821}{40.422} \tag{3.8}$$

These derived relationships were used during various stages of the project according to the hardware used in the offboard outerloop code.

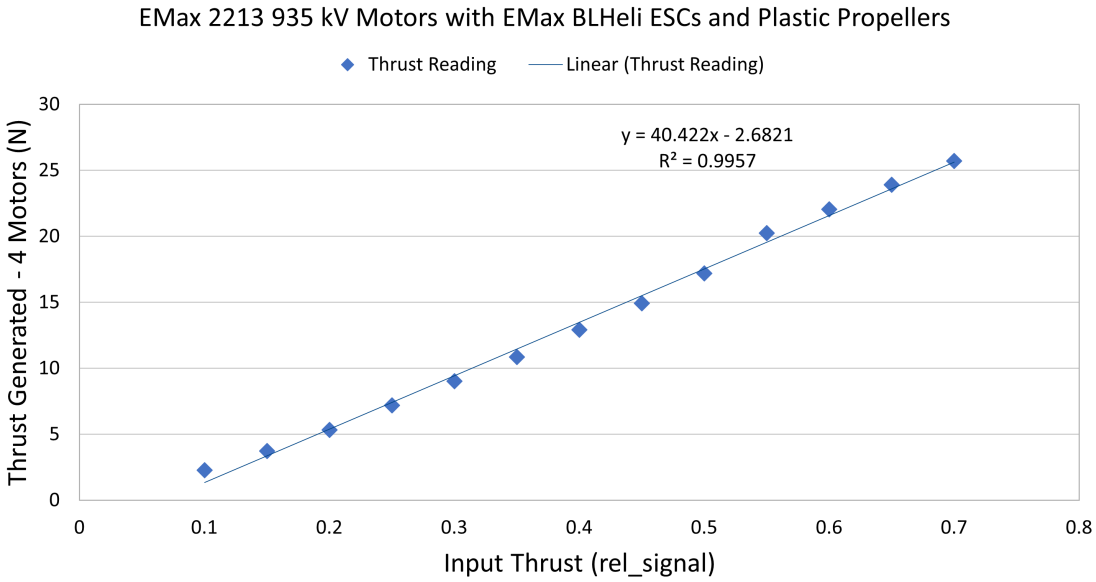


Figure 3.22: Emax Motors with Emax BLHeli ESC and Plastic Propellers

3.3 Tuning

The PX4's inbuilt innerloop and the offboard outerloop designed by us are running using a PID controller. Based on the system being used for experiments, the gains of the PID controller need to be tuned to have the desired performance. The desired performance in our case is the system should reach the desired state in minimum duration without any oscillations. These kinds of systems are known as critically damped systems in control theory. Achieving this performance requires precise fine-tuning of the gains. Several other things also need to be tuned precisely such as the cutoff frequency of the low pass filter, notch filter frequency, etc. In this section, we will explore various methodologies used for tuning purposes.

Tuning of Innerloop PID Gains

The tuning procedure described in the user guide of the px4 (24) needs moving the UAV in an open space. However, the flying arena available inside the lab is very small. So flying the UAV inside the lab unconstrained for tuning purposes is dangerous. For that purpose, the basic tuning is done in a 3 DOF setup (Fig 2.12a), in which the UAV cannot move, but can only rotate.

After attaching the UAV to the 3 DOF setup, the impulse signal is given using the RC controller in roll and pitch direction. The response of the system can be seen in the QGroundControl software as shown in Fig 3.23. We need to have a critically damped response out of the system, which can be done by comparing the setpoint and response

graphs in this interface. We can determine whether to increase or decrease the gains based on the plots available in the QGroundControl interface. If the oscillations are present, then P gain needs to be decreased and if the system takes too much time to reach the P term can be reduced. Similarly, the D term should be as minimum as possible so as to not have the oscillations. If the D term is too high, then the vibrations of the velocity measurements can cause fluctuations in the control signal.

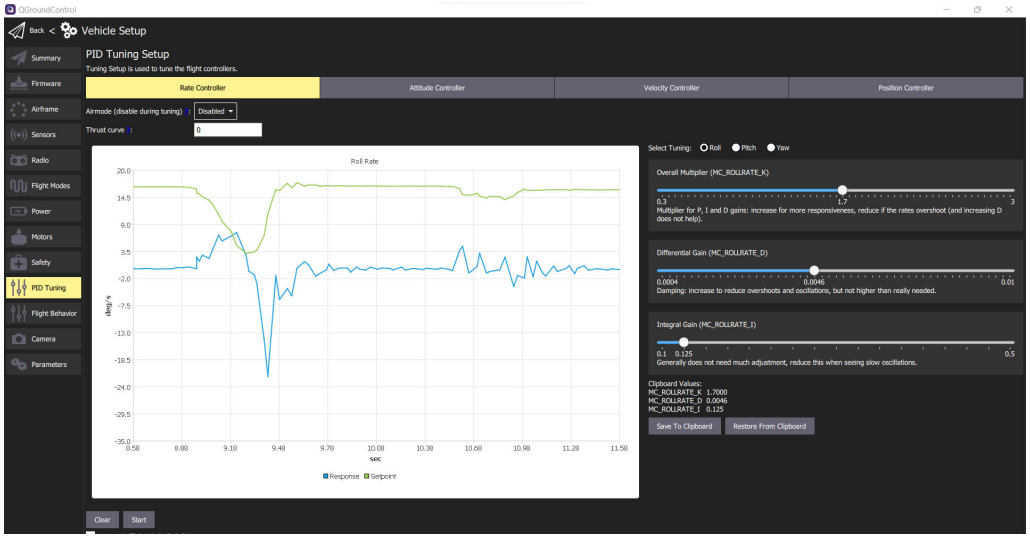


Figure 3.23: PID Tuning Interface of QGroundControl

Based on this methodology, the inner loop PID gains are tuned. The tuning is done for Rate Controller (Roll and Pitch) and Attitude Controller (Roll and Pitch). After the tuning on the ground setup, the final tuning has to be done while the UAV is in flight as the dynamics of the system attached to the ground setup is not the same as the dynamics of the system flying in the air.

The 3DOF setup did have one issue, there was no arrangement to hold the UAV in level orientation before starting the tuning procedure, which required manually handling the UAV initially. In order to solve this issue, it was decided to use the 1DOF setup to tune roll and pitch separately. In order to save the timing of design and manufacturing, it was considered to attach the UAV to the existing thrust measurement setup (Fig 2.9b), which had 1 degree of freedom. The UAV was attached on top of the vertical support channel. However, it was realized that the center of mass of the UAV is too far away from the pivot of the rotation and hence the inertia of this system is very much higher than the UAV's inertia. So it was decided not to use this method and continue the 3DOF setup only.

The tuning of PID gains was first done on a 3DOF setup first for various motor/ESC combinations and then it was done while flying inside the lab. During various tuning procedures, high-frequency oscillations of the UAV and the accuracy of flying in a hover

position were observed to determine the tuning effectiveness. The results of these tuning procedures will be shown in section 4.1.

Tuning of the Filters

The IMUs inside the Pixhawk are directly attached to the Pixhawk body, which is attached to the frame of the UAV through a vibration absorber (Fig 3.24). The frame of the UAV suffers from many vibration frequencies, such as vibrations generated by the rotation of the motors, vibrations of the frame at the natural frequency of the structure, etc. The vibration absorber helps in reducing the amount of vibration noise reaching the IMUs, still this noise cannot be completely eliminated at the hardware level. If the control action is directly done on the measurements of the IMUs, the UAV will try to act upon the noisy signals, which might result in high-frequency oscillations. In order to avoid this problem, the PX4 firmware has low pass filters and a notch filter. The cutoff frequency of the low pass filters and the frequency/bandwidth of the notch filter need to be tuned based on the setup being used.

Figure 3.24: Vibration Absorber Base for Pixhawk (Image source: (6))

The tuning for these frequencies is done using the data logged by the PX4 firmware on the SD card. The PX4 provides the facility to analyze the logged data on their website (25). The logged data of the PX4 can be uploaded on this website, which will generate various analysis plots for the flight. The logging on the SD card needs to be done at a high frequency (can be enabled using SDLOG_PROFILE parameter in the QGround-Control). For cutoff frequency and notch filter frequency tuning, the Actuator control FFT is used. In Fig 3.25, we can see that there are useful frequencies recorded by the IMU under 20 Hz. But the dominant frequencies are near 50 Hz and 80 Hz. These frequencies are unnecessary noisy data recorded by the IMUs from the vibrations of the

Cutoff (Hz)	Delay Approx. (ms)
30	8
60	3.8
120	1.9

Table 3.1: Approximate Delay for Various Cutoff Frequencies (source: (26))

frame. If not filtered, the UAV will not perform as desired and will be having oscillations. In the example FFT plot, a notch filter is not applied. After applying the notch filter at 80 Hz with 30 Hz bandwidth, the performance of the UAV improved drastically. Also, the other 3 cutoff frequencies IMU_DGYRO_CUTOFF, IMU_GYRO_CUTOFF, and IMU_ACCEL_CUTOFF are tuned by trial and error. If these cutoffs are kept very low, it will induce the delay Table (3.1) in the measurement and if they are kept very high, the unnecessary frequencies will not be filtered out. So these cutoffs need to be tuned in order to have the optimal performance. For the Emax Motors and Emax BLHeli ESCs, the optimal values for these cutoffs were 25, 20, and 25 respectively. The full log analysis of an example flight can be seen at (7), which includes many other interesting plots such as Roll estimated vs Roll setpoint, Actuator Outputs, and readings from various sensors, etc.

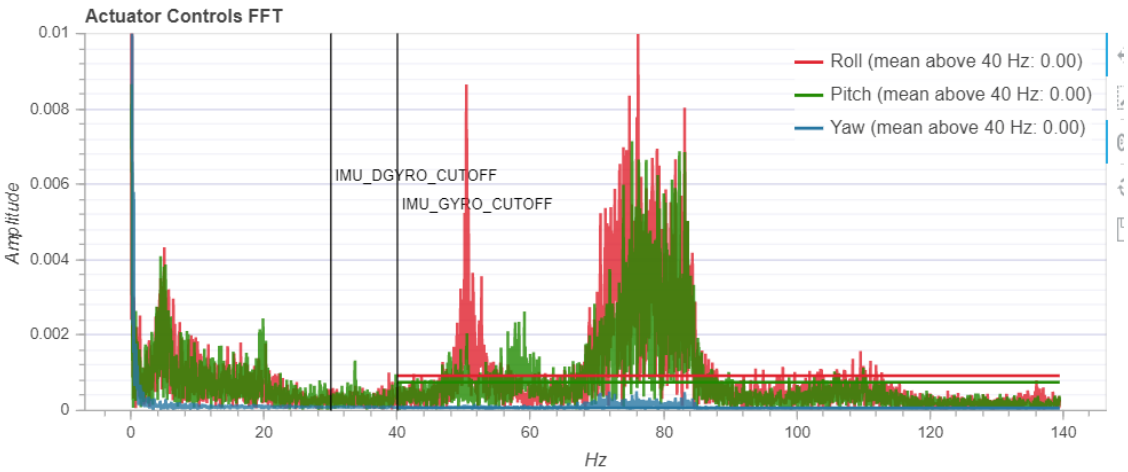


Figure 3.25: Sample Actuator Control FFT created using logged data on Pixhawk (source: (7))

3.4 Derivative Error

Derivative Kick

In the offboard outerloop, the derivative error is calculated in order to limit the velocity (or velocity error) of the UAV. This term acts similar to a damper on the spring-mass system. From theory, the derivative error term is calculated as

$$\frac{d}{dt}(error) = \frac{d}{dt}(setpoint - position) = \frac{d}{dt}(setpoint) - \frac{d}{dt}(position) \tag{3.9}$$

While doing the setpoint-based control (when discrete setpoints are provided manually rather than the continuous trajectory), the $\frac{d}{dt}(setpoint)$ term remains constant except when the setpoint is changed. Also, when the setpoint is changed, the $\frac{d}{dt}(setpoint)$ term becomes very high making the derivative error very high for that particular single iteration and thus making the control input very large. This phenomenon is called derivative kick. In order to overcome this problem, we can simply take the negative of the derivative of the current position of the UAV rather than taking the derivative of the error (basically making the $\frac{d}{dt}(setpoint)$ to be 0 always). This way, the contribution of the setpoint change is neglected from the derivative error, which is anyways 0 when the setpoint is not changing. The derivative kick was removed from the code of the offboard outerloop code after realizing it was causing the control issues.

Low Pass Filter

In the previous section, we explored the solution to the problem of derivative kick. However, there is one more issue associated with the derivative errors. As the position measurements can be noisy, the derivative of it (change in position divided by time interval of the discrete loop) can jump by a large amount. This is not desirable as the system will act based on these noisy values and will have oscillations. In order to resolve this, the low pass filter is introduced while calculating the derivatie error or the velocity in the offboard outerloop.

The first naive strategy for this purpose was to just average out the current and previous 3 values of the derivative error calculations as shown in the equation 3.10.

$$\frac{d}{dt}error(t) = \frac{\frac{position(t-1)-position(t)}{dt} + \frac{d}{dt}error(t-1) + \frac{d}{dt}error(t-2) + \frac{d}{dt}error(t-3)}{4} \tag{3.10}$$

As we can see in Fig 3.26, many spikes of the unfiltered derivative error have been reduced to a smaller amplitude, but still the effect of the noise can be seen. In order to

improve the filtration, the proper filter was introduced as discussed in (27). In this filter the current derivative error is calculated as below:

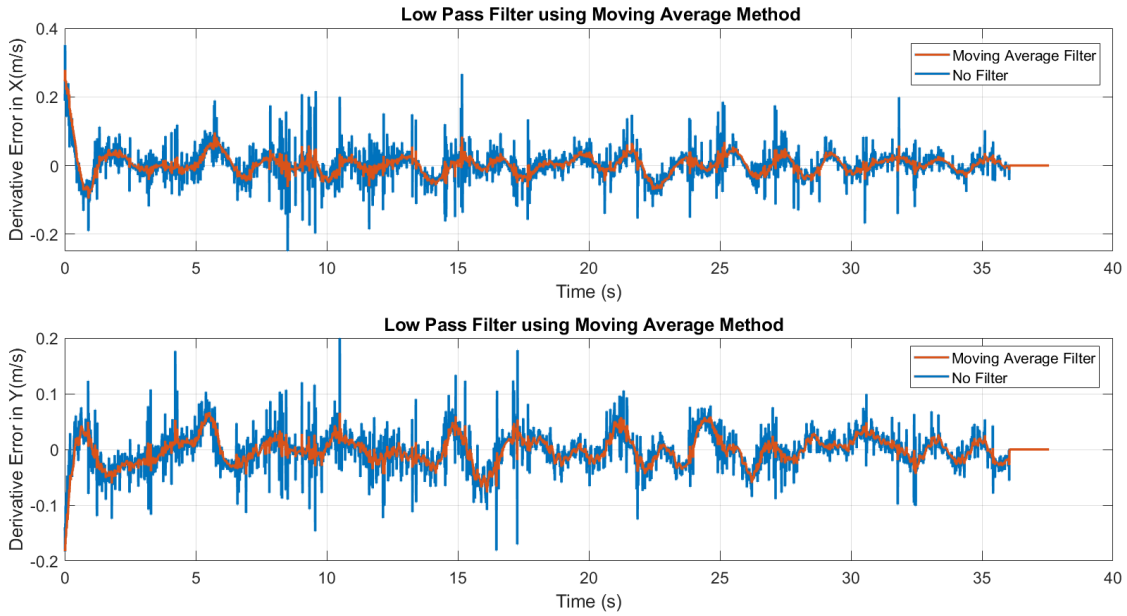


Figure 3.26: Low Pass Filter based on Average of 4 Values

$$\frac{d}{dt}error(t) = n * \frac{d}{dt}error(t-1) + \left(\frac{1-n}{2}\right) \cdot \frac{position(t-1) - position(t)}{dt} \tag{3.11}$$

$$+ \left(\frac{1-n}{2}\right) \cdot \frac{position(t-2) - position(t-1)}{dt} \tag{3.12}$$

In this equation, the factor n needs to be tuned. The higher the value of n, the more the error derivative will rely on history, making the filter aggressive. The lower the value of n, the less aggressive the filter will be. After trial and error, the value of n was decided to be 0.97 for the x and y directions and 0.5 for the z-direction. The comparison between filtered derivative values using this filter and non-filtered values can be seen in Fig. 3.27

While taking off or landing the UAV, the UAV moves very fast and if n is kept high, it introduces the delay in velocity calculations, which creates oscillations of the UAV in a vertical direction. Hence the value of n is kept lower in the z-direction. The Fig 3.28 shows the difference between both the filters for one of the Gazebo simulation runs.

3.5 Hardware Improvements

While doing various offboard outerloop experiments, many hardware limitations were observed. In order to resolve these limitations, some hardware improvements were made periodically to the setup. These improvements are listed below:

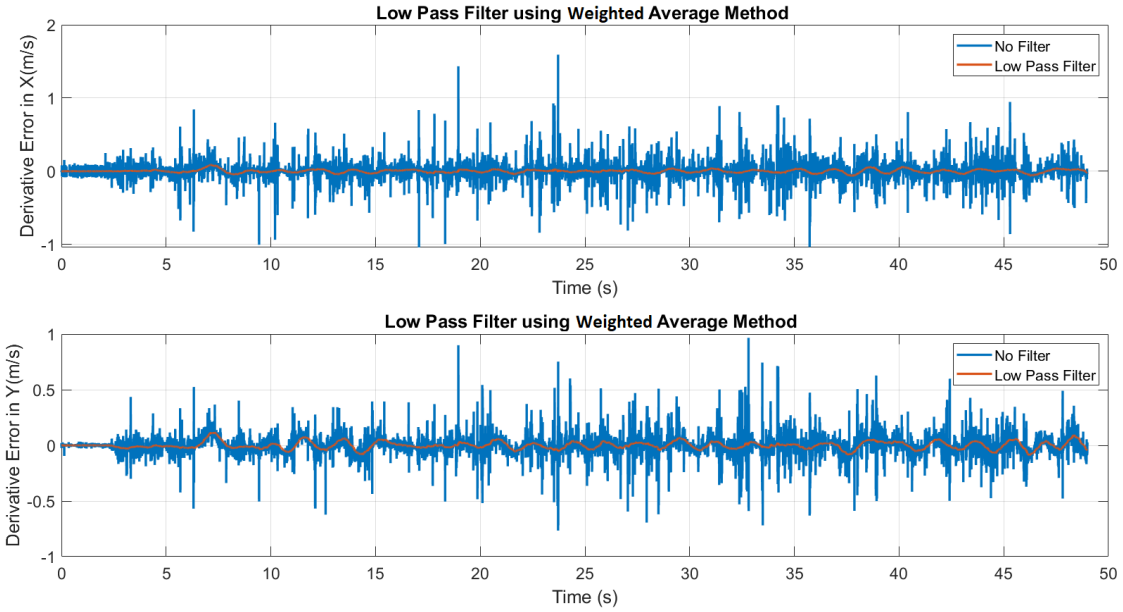


Figure 3.27: Low Pass Filter using Weighted Average Method

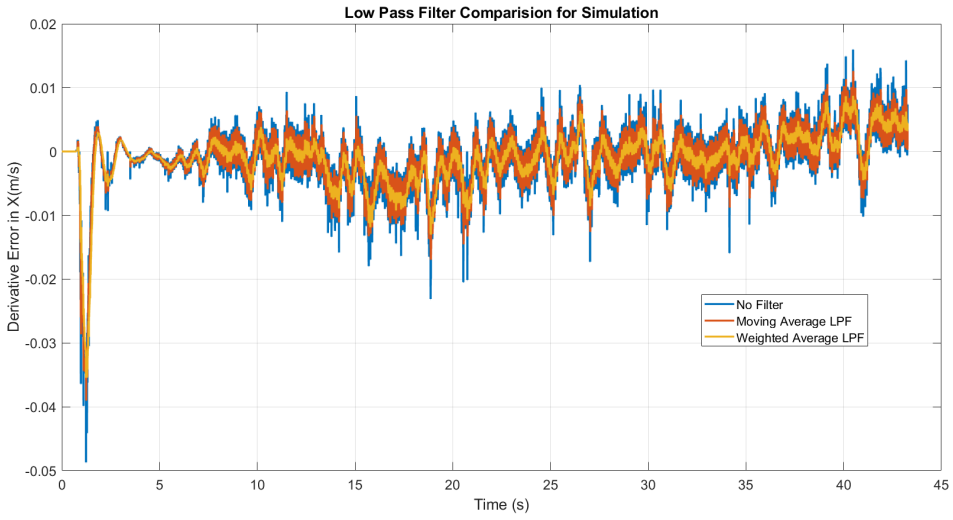


Figure 3.28: Low Pass Filter Comparison

- Many times, it was found that the frequency of the propeller rotation was matching with the natural frequency of the UAV frame's arms. This was causing the resonance of the UAV frame. In order to resolve this, the nylon cable was tied to the UAV passing through all 4 arms. This way, if the vibration is being faced by one of the arms, the other arms will try to stop the vibrations through the cable. This methodology was found effective and resonance was not observed after implementing this solution. The photograph of this solution is shown in Fig 3.29.

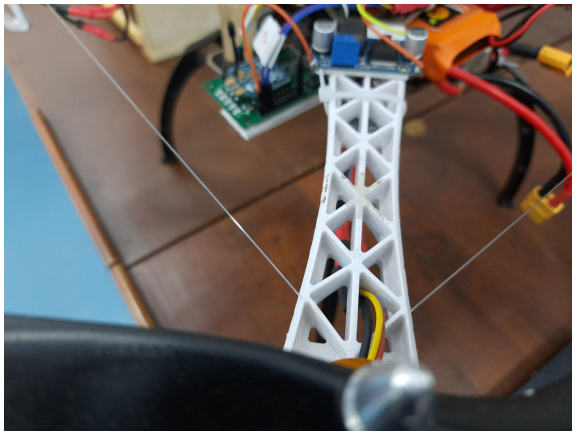


Figure 3.29: Nylon Cable to dampen the vibrations of the arms

- After a few runs, it was found that the connection of one of the motors with the ESCs was not proper. This was due to damage to the soldering of the bullet connectors which are used to connect motors with the ESCs. In order to resolve this problem, the loose bullet connectors were re-soldered. The bullet connector example is shown in Fig. 3.30.

Figure 3.30: Bullet Connector between Motors and ESCs

- The connection between Pixhawk and XBee is made using a custom cable. Initially, jumper cables were used for this connection, which was getting disconnected frequently causing the communication loss. In order to resolve it, the crimping of the cable was done using the JR-type servo connectors and the heat shrink was applied on top to make the connection mechanically strong. The final connection is shown in Fig. 3.31. After making this connection, no communication loss due to a loose connection was observed.

- The propellers which come from the factories have an imbalance of the mass (and hence inertia) about the axis of rotation. If the balancing is not done, it can cause

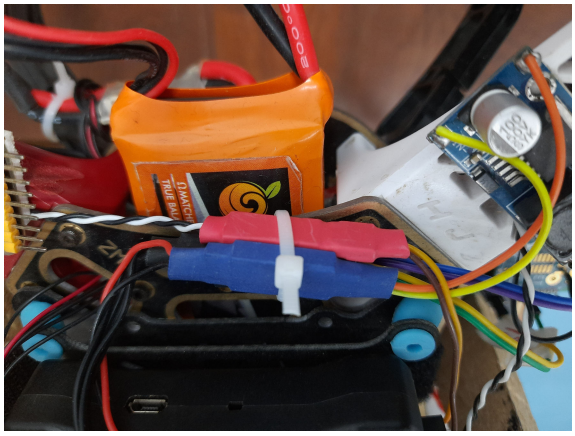


Figure 3.31: Crimped and Mechanically Stronger Connection for XBee

vibrations in the frame. In order to resolve this, the propeller balancer as shown in Fig. 3.32 was used. This propeller balancer is attached to the base via a magnetic connection, making it almost frictionless. So after putting the propeller on this balancer, the side which moves above relative to the other side is lighter and is made a little heavier with the use of stickers. After balancing, both sides remain at the same height on the propeller balancer.

Figure 3.32: Propeller Balancing

- The position of the UAV is determined using the Vicon motion capture cameras, which are in the Vicon world frame. But the roll/pitch of the UAV are in the body frame, which the pixhawk uses to move in x-y directions. If the motors of the UAV are not aligned with the Vicon cameras' axes, then there will be errors in the movement of the UAV in x-y directions (Fig. 3.33). For example, if the motion is required in the X direction, the UAV will initially move towards its perceived X direction, but after the movement, there will be an error in the position of the UAV and the controller will correct the motion. Due to this, the UAV will not move

in a perfectly straight line. In order to overcome this problem, the motors of the UAV need to be aligned with the Vicon camera reference frame while defining the object. For that purpose, the Vicon markers were attached on top of the shafts of the motors. These markers create a square, which can be seen in the Vicon tracker software. In order to align the motors with the Vicon world frame, the square made by 4 motors is visually aligned with the gridlines of the Vicon tracker software and then the object is defined in the tracker. This way, the motors of the UAV are exactly aligned with the frame of the Vicon world. Fig. 3.34 shows the markers attached on top of the motors for this purpose.

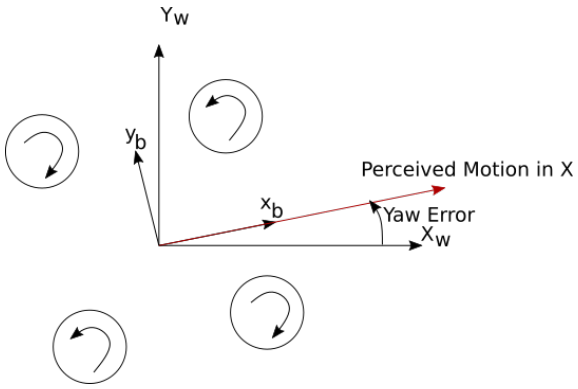


Figure 3.33: Yaw Error between actual world frame and perceived frame by the UAV

Figure 3.34: Alignement of the Pixhawk Body Frame with Vicon World Frame

- The frame of the Vicon world is also calibrated using the 5 markers wand provided with the Vicon cameras. Initially, the calibration was done without making sure of the alignment of the wand with the gravity. During the flight runs, it was realized that the frame of the Vicon world is not perfectly horizontal, which was causing the errors in z-direction for the UAV. In order to resolve this issue, the calibration wand

was put on the perfectly horizontal base (using the spirit level) and then the Vicon world calibration was performed.

- In order to calibrate the accelerometer, magnetometer and gyroscope of the Pixhawk, the calibration procedure is described in the QGrounControl software, in which we need to hold the UAV in various orientations (Fig 3.35). Initially, the calibration was done approximately, but the accuracy of the hover flight was affected due to improper calibration. In order to resolve this, the calibration of the Pixhawk was done properly using the right angle protractor as shown in Fig 3.36.

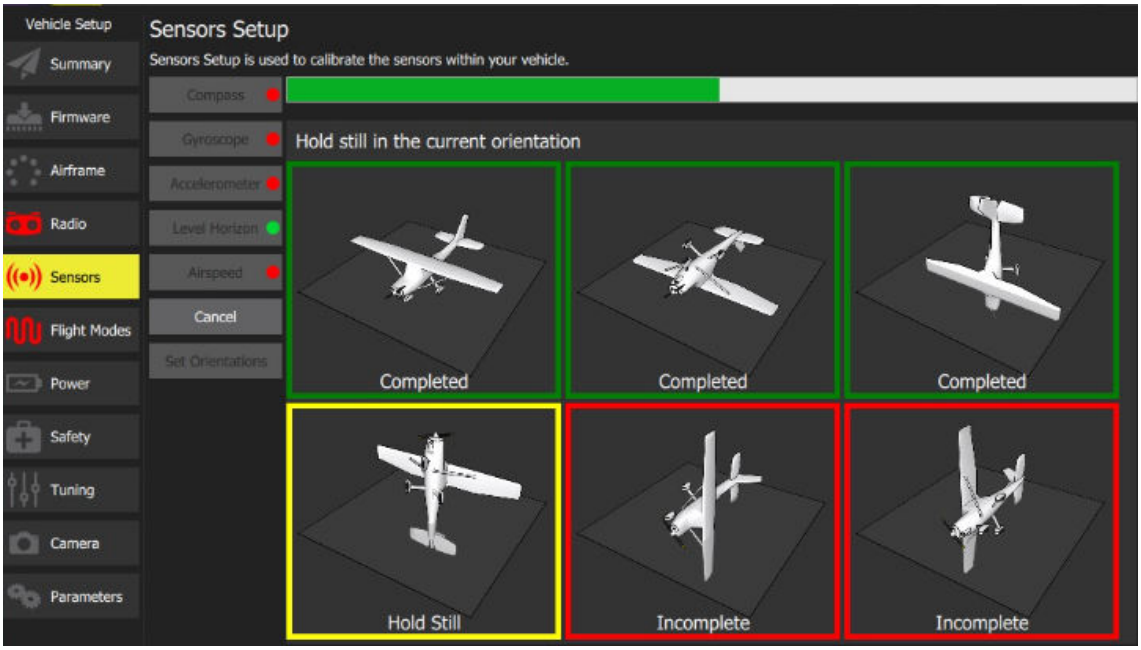


Figure 3.35: Calibrating Pixhawk Sensors in QGroundControl

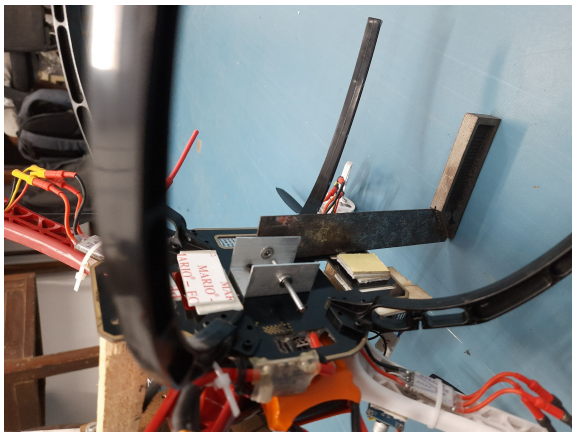


Figure 3.36: Calibrating Pixhawk Sensors using Right Angle Protractor

- After a few flights, it was observed that the markers detected in the Vicon cameras oscillate too much. It was observed if the Vicon frequency is kept a little lower (125

Hz), these oscillations are less. So it was considered to use 125 Hz frequency in the future.

- When we define the object of the UAV in the Vicon tracker software, the center of mass of the UAV is defined automatically at the center of the markers. But the center of the mass of the UAV is at a different location. If this offset is not corrected, then the tilt of the UAV will be considered as the movement in x-y direction by the Vicon tracker causing the slow oscillations. In order to resolve this, the center of mass need to be offsetted in the Vicon tracker software (Fig. 3.37).

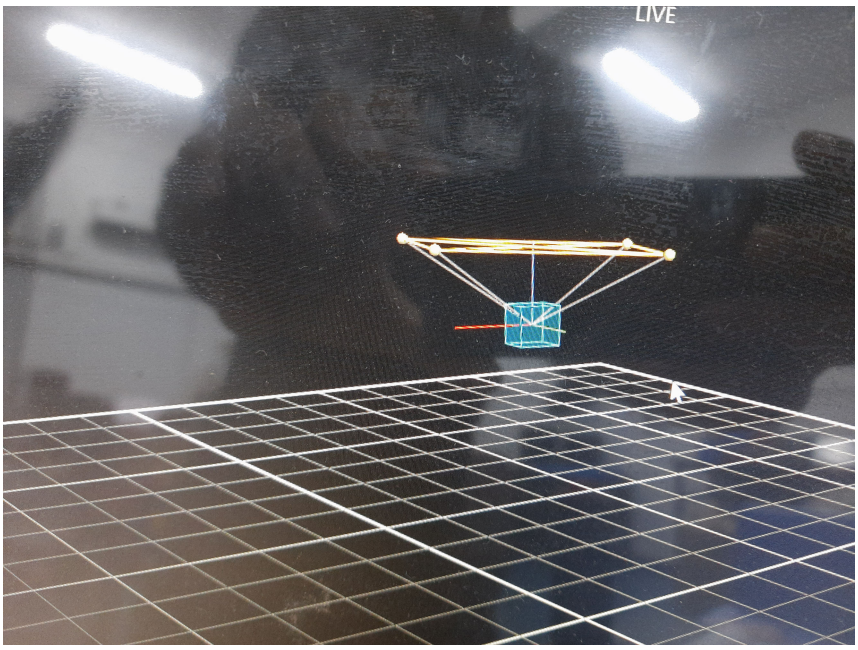


Figure 3.37: Center of Mass Offset for Vicon Object

- During one of the experiments, it was observed that the frame of the UAV was vibrating too much. After the diagnosis, it was found that the the frame was cracked at one location (Fig. 3.38). If crack was corrected using a super-glue, the vibrations vanished. From this, it can be concluded that even if the crack does not look like it will affect the strength, it can still cause the vibrations.

- A few times during the flight, the Vicon cameras were obstructed by various objects when they were mounted on the bottom rail. In order to avoid this problem, the Vicon cameras were shifted to the rails located at a higher position.

oneshot125 protocol

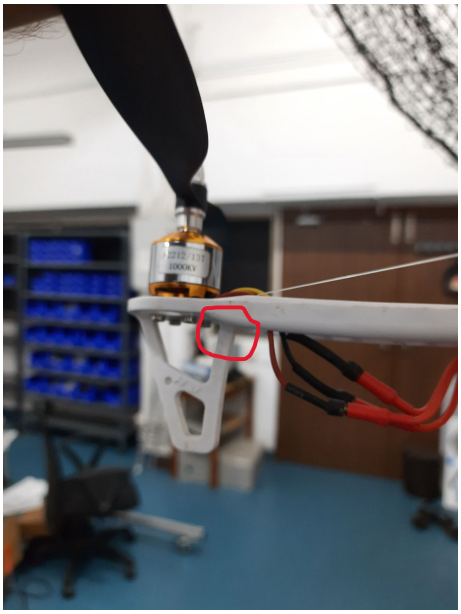


Figure 3.38: Crack location in the Frame

3.6 Gazebo Simulation

PX4 is a flight controller software that is used on Pixhawk flight controller hardware. The same software can be used in Software-In-The-Loop (SITL) simulation also. Doing this enables us to test the control algorithms in the software before actually flying the UAV. We can also test novel algorithms without any risk. This avoids any potential damage to the vehicle and the surroundings caused by mistakes in the control algorithms. During this project, the SITL simulation was used with the Gazebo simulation environment. The choice was made considering the compatibility of the Gazebo with the ROS environment, which is extensively used for doing the hardware experiments. In this simulation, the commands can be sent to the topics of the mavros the same way they are sent to the UAV Hardware. So this enables us to check even the syntax errors in the offboard codes.

Setup

In order to setup the Gazebo simulation for PX4, the following procedure needs to be followed:

1. Install Ubuntu 18.04 (with dual boot if required) on your system
2. Install the package of ROS, MAVROS, MAVLINK using the Ubuntu Development Environment ((28)). Follow the instructions under ROS/Gazebo section.
3. Install python-rosdep2

4. Install Python3 if not already installed
5. Repeat step 2 until all the errors are removed
6. Install QGroundControl
7. Install PX4 source code inside the src folder of the catkin_ws (Follow the instructions under "Gazebo, JMAVSim and NuttX (Pixhawk) Targets" section of the (28))
8. Go to PX4-Autopilot directory in terminal
9. run *$ make px4_sitl jmavsim* and resolve for any errors. If no error is present, the simulation window will run in jmavsim
10. Exit the simulation
11. Now we will be running the simulation with Gazebo. For that, follow the instructions under *Launching Gazebo with ROS Wrappers* section of (29).
12. The simulation by default will run with "iris" model, but for our hardware, the "solo" model is more closure to the actual hardware. So change the model inside the mavros_posix_sitl.launch file.
13. Adjust the inertias inside the solo.sdf file found in PX4-Autopilot/tools/sitl_gazebo/models directory.
14. In the simulation, the UAV will not arm due to failsafe of no RC input. In order to resolve that, change the parameter COM_RCL_EXCEPT to 4 using QGroundControl.
15. In order to make the background of the simulation simple, modify the empty.world file inside the PX4-Autopilot/tools/sitl_gazebo/worlds directory
16. Now simulation is ready, in order to run the offboard outerloop code, the outerloop ("ol") package installation needs to be done, which will be discussed in chapter B.

Introduction of Real life Latency

There are communication delays in all the steps in the hardware setup. This was discussed in details in the section 2.1. Though the simulation contains the noise introduction in the IMU readings, it does not have any communication delay. In order to simulate the real-life experiment in the simulation, the delay needs to be introduced in the simulation.

Figure 3.39: Methodology to introduce the Delay in the Simulation

For this purpose, the delay was introduced using the first in first out the principle in the offboard outerloop code.

In order to introduce the Vicon latency (approximately 12 ms in real life), the list is created inside the Gazebo Callback function of the offboard outerloop code (Fig. 3.39). The current gazebo reading of the position/orientation of the UAV is inserted at the first index of the list. In order to calculate the errors, the value from the last index of the list is used and then deleted. This way, we always use the value which came first to the list. The size of the list is decided by the amount of delay we want to introduce and the frequency at which the Gazebo data is being received. For example, in order to introduce the delay of 12 ms in the Gazebo callback function being updated at 250 Hz (new data on each 4 ms), we make the list of the size 4. So the delay of the Gazebo will be $(4-1)\times 4 = 12$. Using this method, we can introduce the delay only at the intervals of 4 ms each for the loop running at 250 Hz, but it is good enough as the hardware delay is also not always constant.

Similarly, in the hardware, we have a delay in the communication between off-board computer and the Pixhawk. This delay is approximately 10 ms. So using the same method, the delay is introduced inside the controller function (which is running at 250/3 Hz) in our case. So if we make the list of the size 2, then the delay will be $(2-1)\times 12ms$. Which is close to an approximate delay of 10 ms.

The code (offboard_outerloop_1.py inside the scripts of (30)) contains the provision to modify the delay value inside the configuration parameter section. It automatically calculates the required size of the list in order to introduce the delay value closest to the input value.

Attaching the Pendulum Link

In the section 4.4, we discuss doing the experiments with the link attached to the UAV. But before doing the experiments on the hardware, it is necessary to first do the simulation study in order to make sure there is not any obvious problem with attaching the pendulum link. Also, various algorithms which were tried with the pendulum such as oscillation increase, oscillation swing control, etc were first tested on the simulation and many errors were corrected in the simulation studies. The UAV would have suffered many crashes if the simulation was not done.

The standard models available with the PX4 simulation do not have a pendulum link attached to them. So in order to include the pendulum link, the solo.sdf file was modified to include the pendulum link. The additional code which was added to the sdf file is shown in Appendix A. In the code, an extra 1DOF pivot is added keeping the base_link as the parent and the link is added keeping the 1DOF pivot as the parent. The inertias were calculated by making the accurate model of the pendulum in Solidworks.

The results of some of the simulations for various trajectories will be discussed in section 4.1. Fig. 3.40 shows the simulation screen with the pendulum attached to the UAV.

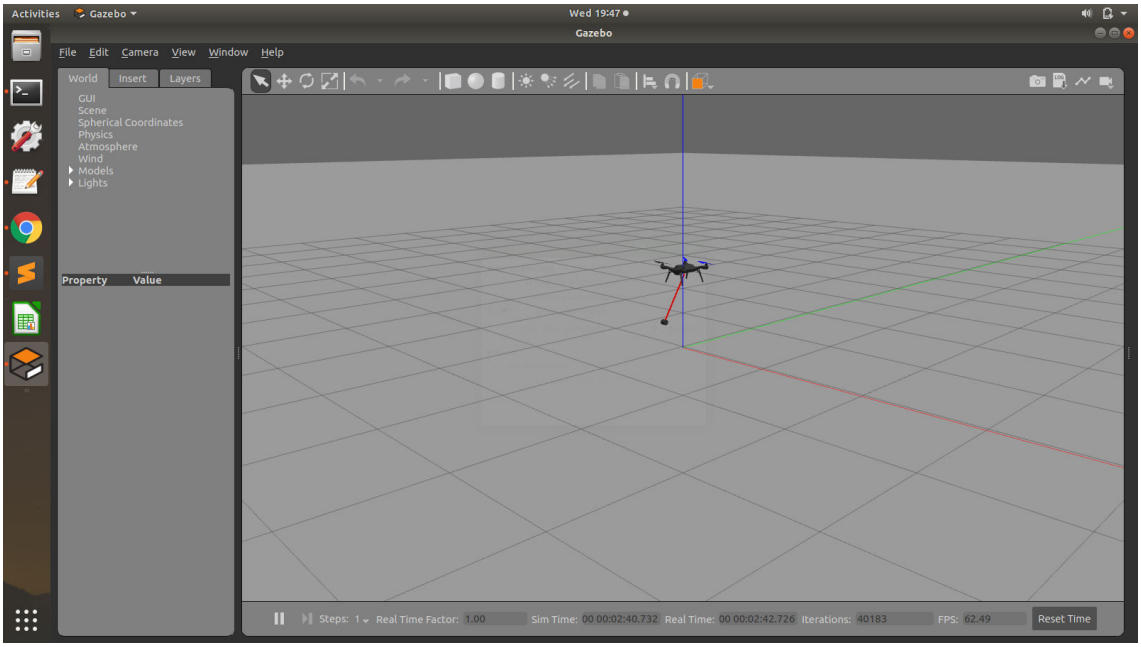


Figure 3.40: Gazebo Simulation Screen with Pendulum Attached

Chapter 4

Results and Conclusion

In chapter 3, we discussed various tasks done in the preparation of flying the hardware. In this chapter, the results of the hardware flight experiments will be discussed. We will first discuss the basic structure of the code used for offboard outerloop in the subsection 4.

Code Structure

The table 4.1 shows the structure of the code used as the offboard outerloop. Various functions and classes used in the code are described in the table. The basic flow of the data was shown in Fig.3.14.

4.1 Hover Experiments

In order to confirm the effectiveness of the controller, the hover control experiments are conducted. In the hover controller, the desired position is entered manually. The desired velocity and acceleration are considered to be 0. Using these desired values, the position error, velocity error, and integral errors are calculated and the PID controller is implemented as shown in equation 4.1. The velocity error and integral error are calculated as discussed in sections 3.4 and 4.1.

$$\ddot{\mathbf{r}}_{command} = \mathbf{k}_p \cdot \mathbf{r}_{error} + \mathbf{k}_v \cdot \dot{\mathbf{r}}_{error} + \mathbf{k}_i \cdot \int \mathbf{r}_{error} \tag{4.1}$$

From the command acceleration, the desired roll and pitch are calculated as shown in equations 4.2, 4.3 and 4.4.

$$\phi_{des} = \frac{1}{g}\left(\ddot{\mathbf{r}}_{command,x} \cdot sin(yaw_{des}) - \ddot{\mathbf{r}}_{command,y} \cdot cos(yaw_{des})\right) \tag{4.2}$$

$$\theta_{des} = \frac{1}{g}\left(\ddot{\mathbf{r}}_{command,x} \cdot cos(yaw_{des}) + \ddot{\mathbf{r}}_{command,y} \cdot sin(yaw_{des})\right) \tag{4.3}$$

Offboard Outerloop Code Structure

Import rospy, messages, csv, numpy, math

Functions Quaternion_from_Euler, Euler_from_Quaternion, Rotation Matrix from Quaternion

Class fcuModes

//Class to activate various flight modes of PX4

Class Controller

Initialize All the required variables and parameters

Function RCCb

//Callback function to update the RC controller values

Function ViconCb

//Callback function to update the Vicon values

Function LinkViconCb

//Callback function to update the pendulum related Vicon values

Function Link_Calculation

//to calculate various angles for Pendulum Link

Function errors_calculation

//to update the errors based on vicon data and current trajectory

Function StateCb

//to update the armed/disarmed state of the UAV

Function BatteryCb

//to receive the updates about battery status

Function Outerloop

//controller function to update the target attitude and thrust values

Function load_stabilization

//to calculate the correction values for pendulum swing stabilization

Function load_unstabilization

//to calculate the correction values to increase the oscillations of the pendulum

Function Emergency_land

//in case of emergency or vicon data loss, emergency landing

Function main

Subscribe mavros/state

Subscribe mavros/battery

Publisher mavros/setpoint_raw/attitude

Subscribe /vicon/quad_aligned_1/quad_aligned_1

Subscribe /vicon/link2/link2

Arm the motors

set offboard mode

call outerloop

publish the attitude and thrust

$$thrust_{des} = mg + m\ddot{\mathbf{r}}_{command,z} \tag{4.4}$$

In the code, the desired ϕ, θ, and desired yaw are converted into the quaternion. Also, the physical thrust calculated in the equation 4.4 is converted to normalized thrust as discussed in the section 3.2. The target attitude signal containing the quaternions and the normalized thrust is sent to the PX4, which uses its own innerloop controller (Fig. 1.10) to follow the desired attitude and thrust. In the following subsections, we will explore the results from the hover flights.

First Flight

Figure 4.1 shows the position of the UAV in 3 dimensions, position in XY plane, z-direction and XY position errors during the flight. This was the first-ever flight done with the offboard outerloop without any tuning. The flight resulted in uncontrolled takeoff and sideways drift and the UAV collided with the nets.

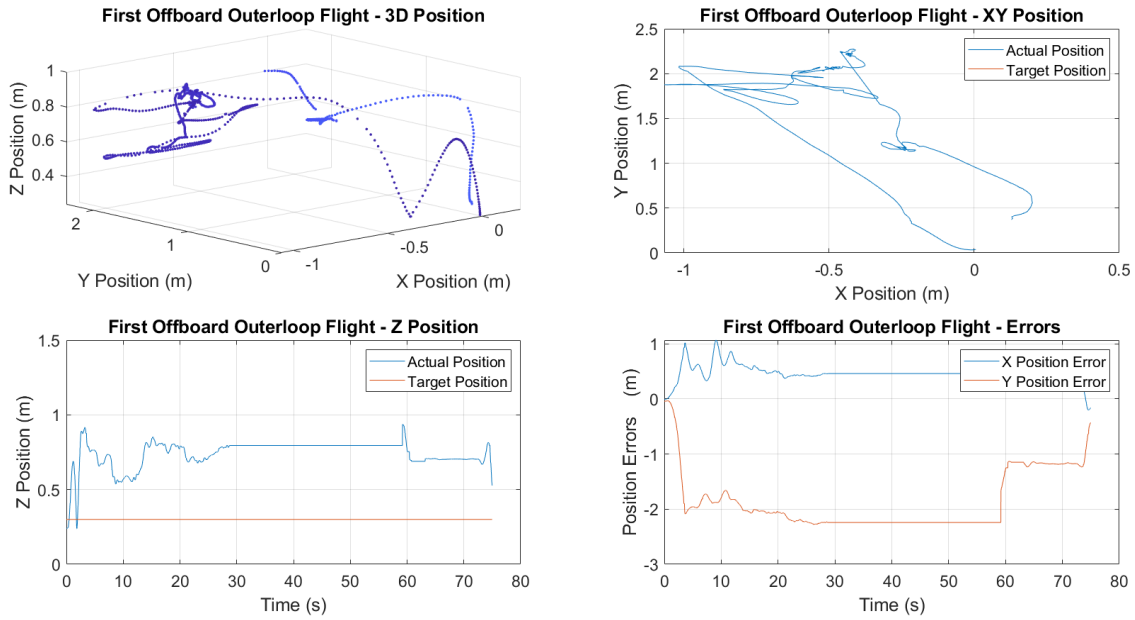


Figure 4.1: First Hardware Flight with Offboard Outerloop

After analyzing the logged data, it was found that the tuning of the outerloop was not correct, which resulted in uncontrolled flight.

First Successful Flight

After some corrections and tuning, the UAV was able to fly a stable flight. As we can see in Fig 4.2, the UAV was able to hold the position very well. However, the UAV

was holding the position centered around 20 mm in the x-direction and 2 mm in the y-direction. This was due to non-symmetries in the UAV frame and calibration errors of the Pixhawk. As there was no integral error correction, The UAV wasn't able to compensate for these errors. Also in the z-direction, many oscillations are happening, which is due to poor tuning and lack of the low pass filter. Also, initial the altitude of the UAV is more than the target, which decreases slowly. This is due to the voltage of the battery being high initially and dropping gradually. The video of the initial hover flight can be seen at (31).

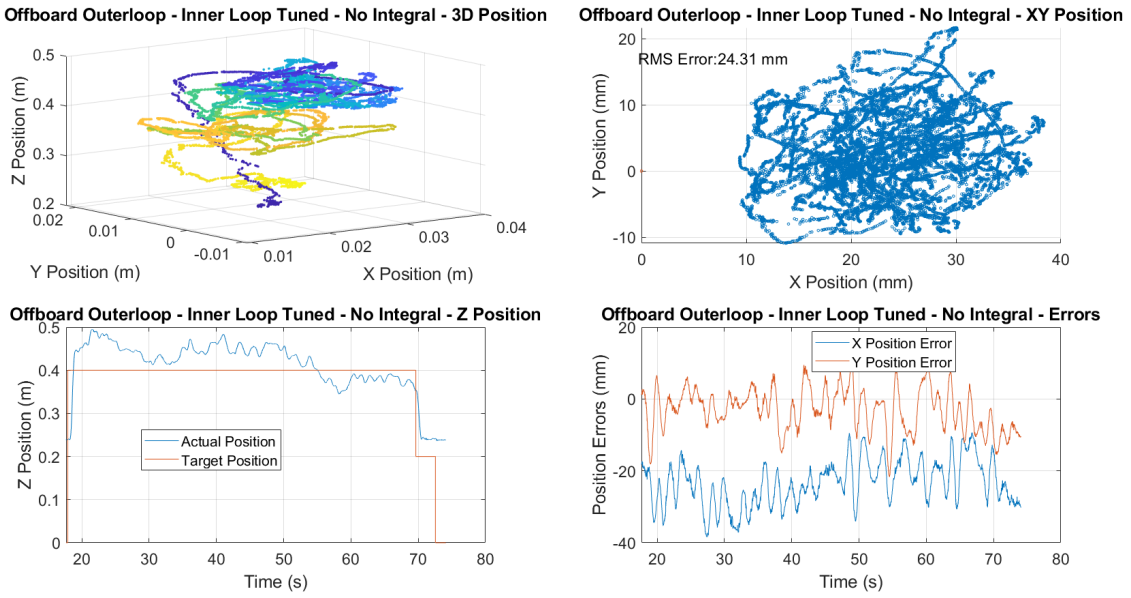


Figure 4.2: Example of the Successful Offboard Outerloop Flight

The Fig 4.3 shows the histogram of this hover flight. The histogram is plotted based on the UAV's distance from the origin in the XY plane. We can see that the UAV remains at a distance of 25 mm for the maximum amount of time. Also, the orange plot is the cumulative probability curve, which shows the probability of the UAV being inside the given distance. This cumulative probability curve can be used as a measure to determine the accuracy of the hover flight. For example, in the given flight, the UAV will be within a 28mm distance for 70% of the time. In order to compare the various flights, we can find at what distance the UAV has 70% of the probability of being inside.

Introduction of integral error and the low pass filter

Further to improve the performance, the integral error correction was introduced in the code. This integral error correction compensates for the modeling inaccuracies and non-symmetries of the system. In the discrete system, the integral error is calculated using the Equation 4.5.

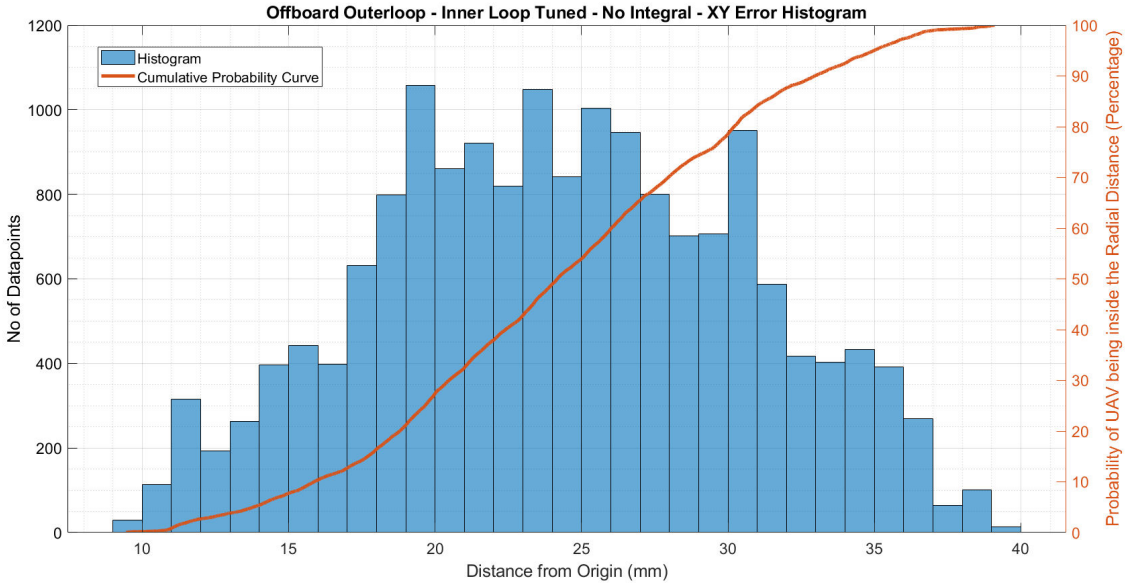


Figure 4.3: Histogram of the Successful Offboard Outerloop Flight

$$Integral\ Error(t) = Integral\ Error(t-1) + Error(t) \cdot dt \tag{4.5}$$

However, we cannot use this equation directly as the integral error suffers from the phenomena called *Integral Windup*. In our case, the integral error can start accumulating from the floor itself before takeoff, the integral error can become very large when the setpoint is changed, etc. In order to avoid this from happening, the conditions are applied to the calculations of the integral error such that the integral error starts accumulating the errors only when it reaches a certain altitude and also when it reaches closer to the setpoint. Also, the integral error is bounded on both sides to avoid the integral error becoming a very large number.

The low pass filter is also introduced as explained in section 3.4. Introducing both these features, the RMS error of the flight reduced from 24.31 mm to 10.95 mm (Fig 4.4). We can also observe an integral error in the z direction becoming higher and higher as the voltage drops. From the histogram (Fig 4.5), we can see that the UAV has more than 70% probability of being inside the distance of 13 mm from the origin.

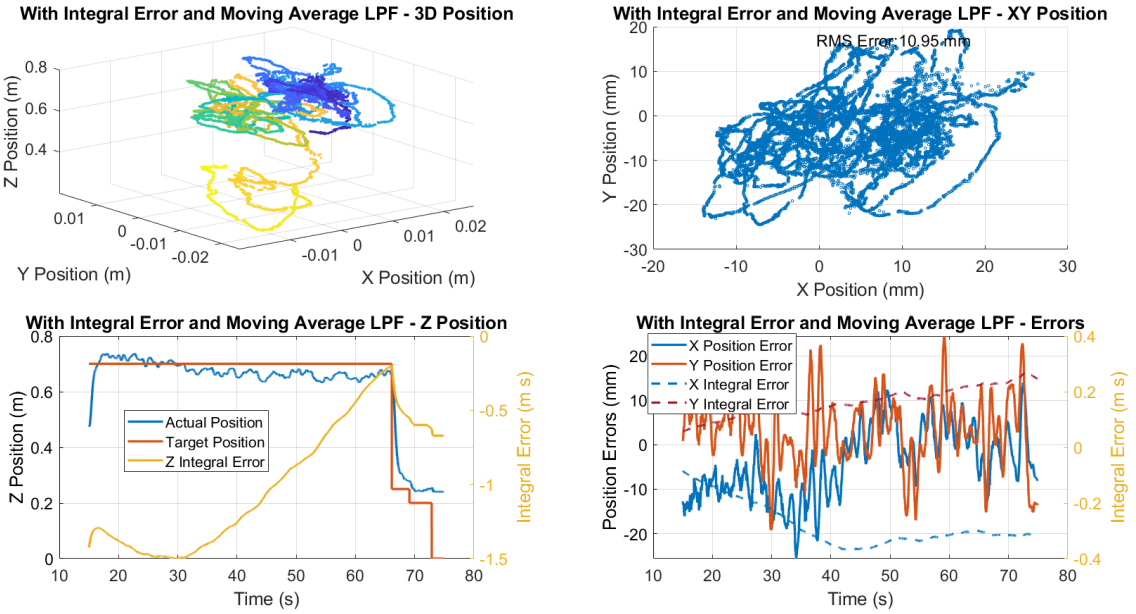


Figure 4.4: Hover flight after integral error and low pass filter was introduced

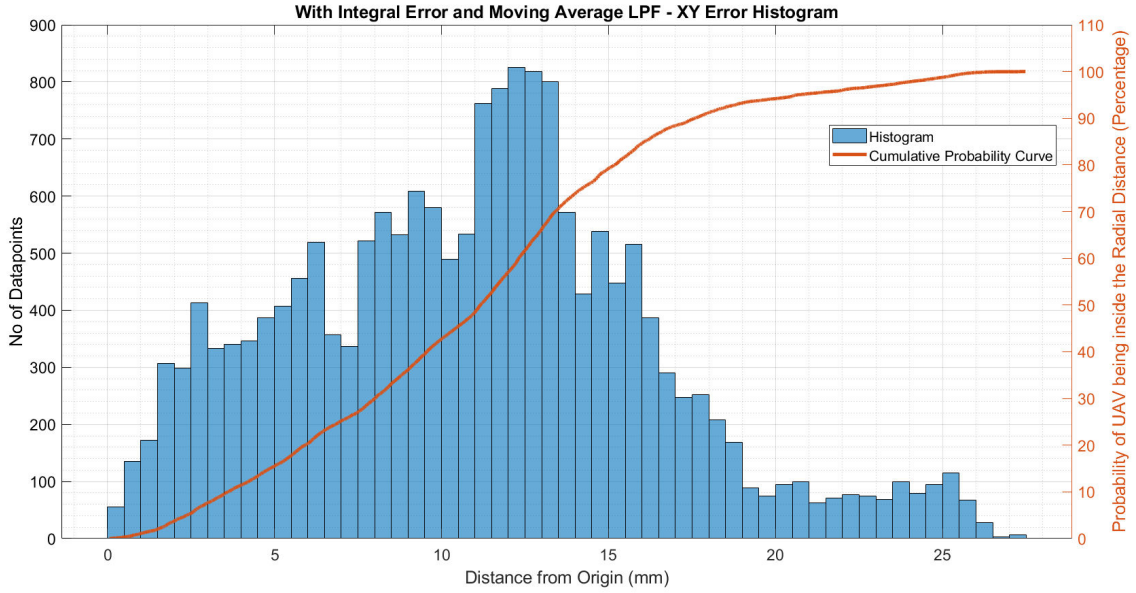


Figure 4.5: Histogram of the hover flight after integral error and low pass filter was introduced

Emax Motors

Even after introducing the corrections as described in previous subsections, the performace of the UAV did not improve beyond a certain limit. So to improve the performance further, the motors were changed from the generic motors to the Emax motors. The Fig 4.6 shows the UAV performance for the flight with the Emax motors after retuning. We can see in the results that the XY error was almost the same as the previous

motors. However, the performance was much better in the z-direction. The video of this experiment can be seen at (32).

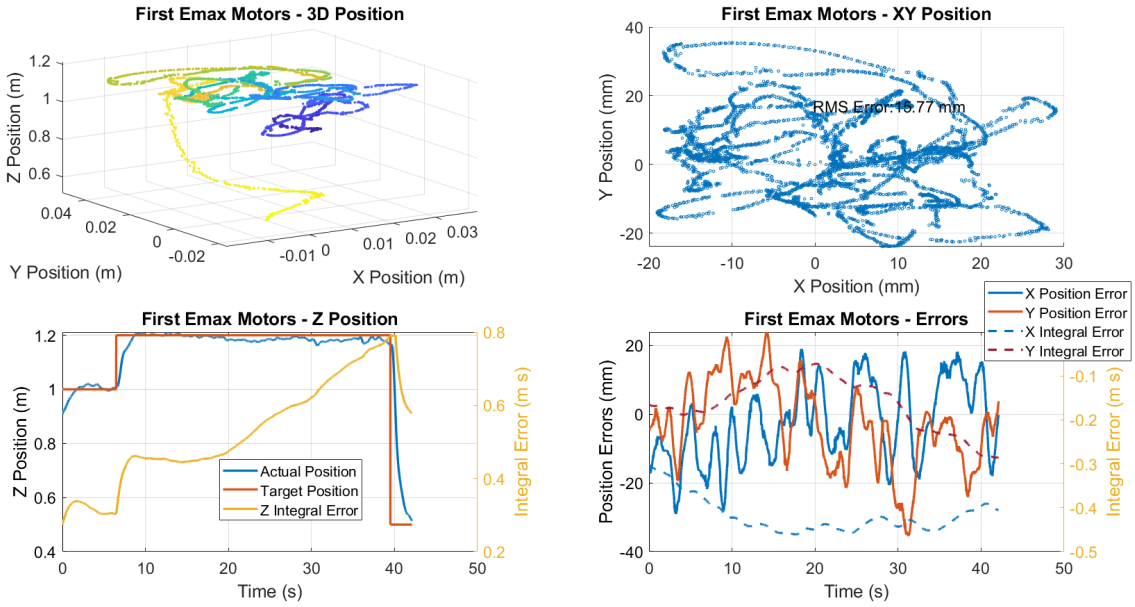


Figure 4.6: Hover flight with the Emax Motors

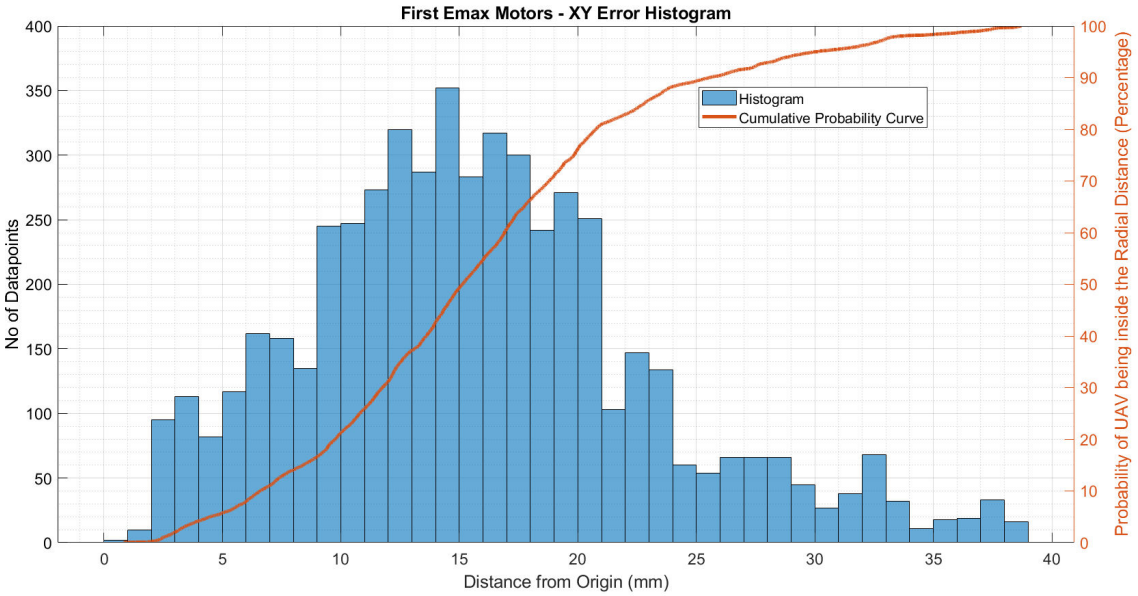


Figure 4.7: Histogram of the hover flight with the Emax Motors

Tuning of Cutoff Frequency and Introduction of Notch Filter

As described in the section 3.3, the cutoff frequencies were tuned and the notch filter was enabled to improve the hover performance. The results from this experiment are shown in Fig. 4.8 and Fig. 4.9. Though the performance of the UAV looks the same

numerically, it did not have any sudden oscillations which were present previously. The video of this experiment can be seen at (33).

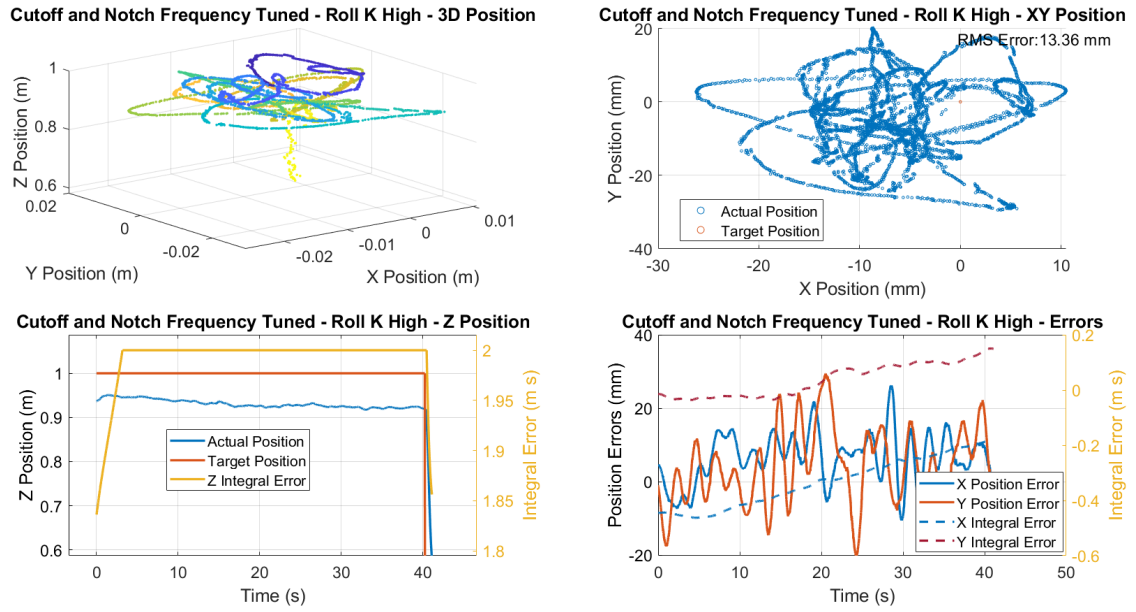


Figure 4.8: Hover flight after tuning the cutoff frequency and introduction of Notch Filter

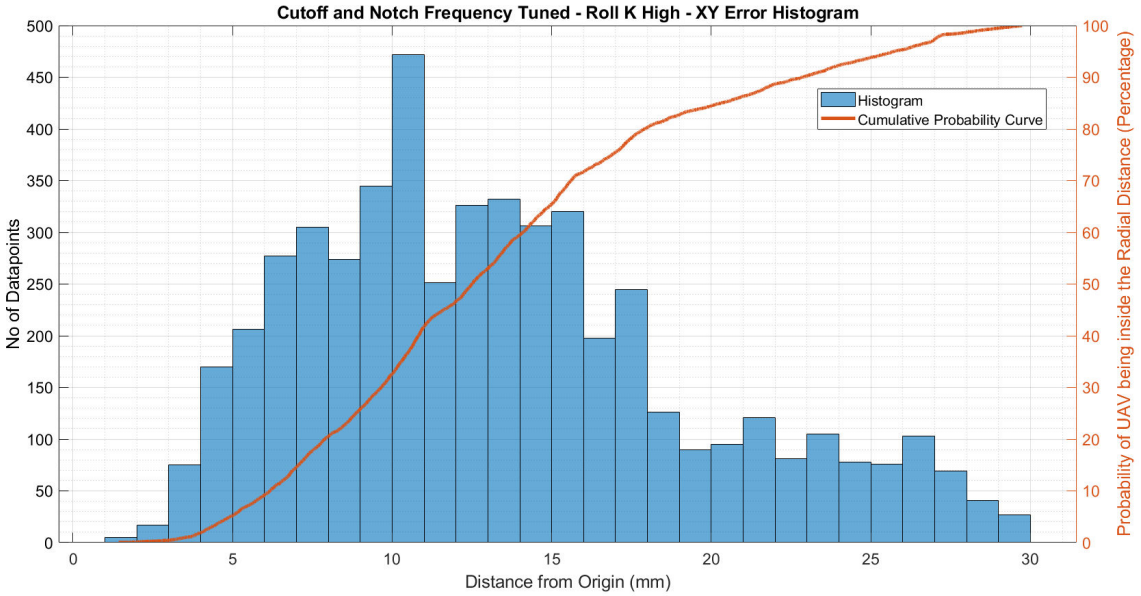


Figure 4.9: Histogram of the hover flight after tuning the cutoff frequency and introduction of Notch Filter

BLHeli ESC

Till now, the ESCs used were the cost-effective SimonK ESCs. However, there are better ESCs available that can improve the performance. In order to improve the

performance further, the BLHeli Bullet Series ESCs were used on the existing setup. These ESCs support Oneshot125 protocol (23) and have active braking. Due to this, the communication between Pixhawk and ESCs can be at a much higher frequency (up to 4 kHz in place of 400 Hz of the existing ESCs). With active braking, the reduction of the speed of the motor can be immediate when required, making the control of the UAV better. The results form this experiment can be seen in Fig. 4.10 and Fig. 4.11.

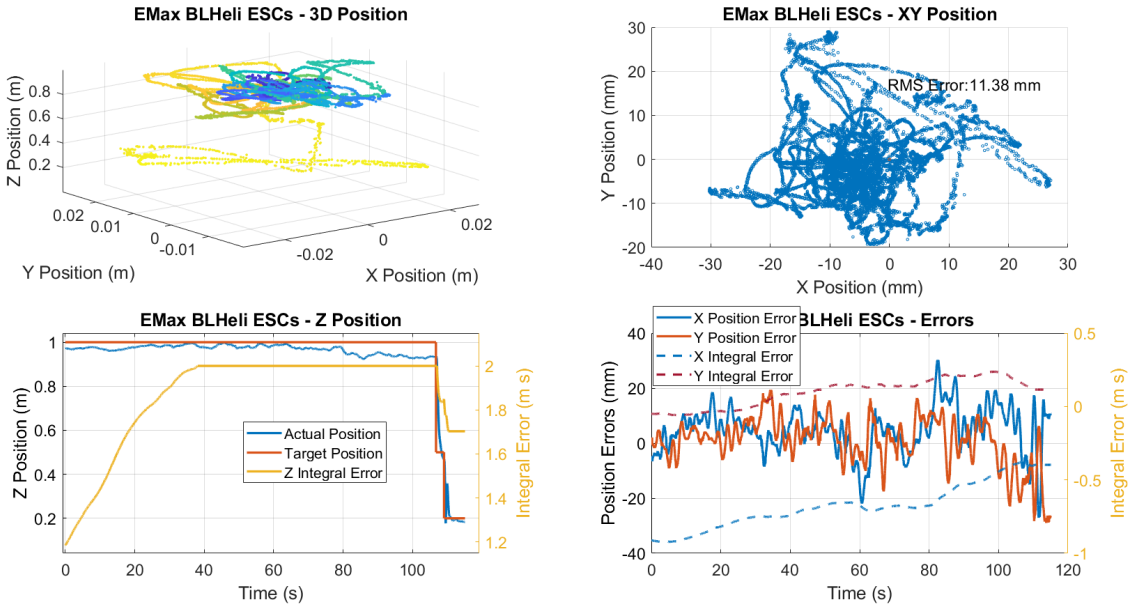


Figure 4.10: Hover flight with BLHeli ESCs

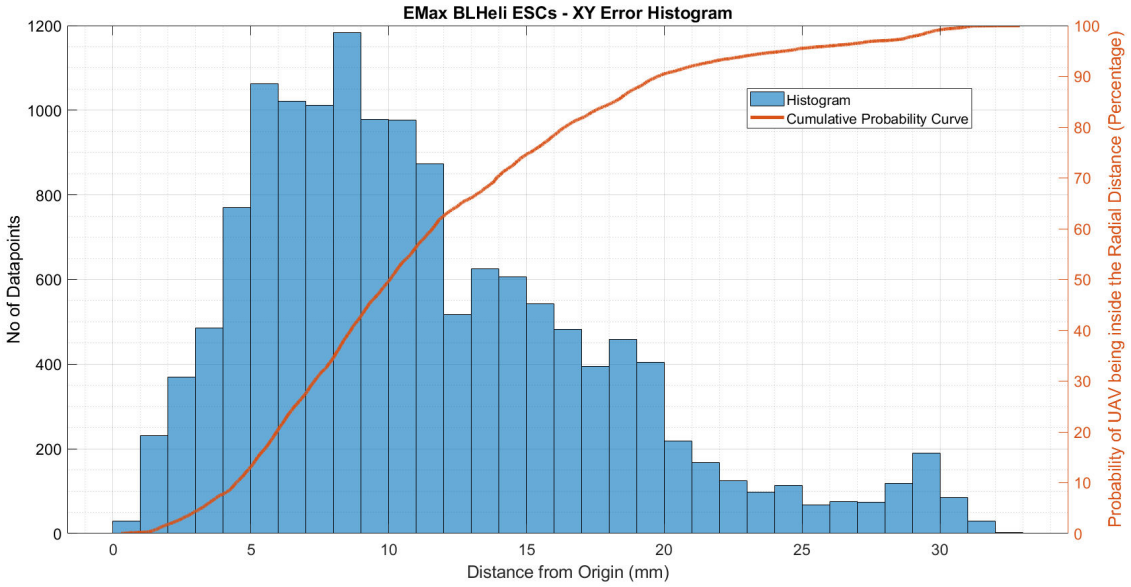


Figure 4.11: Histogram of the hover flight with BLHeli ESCs

Effectiveness of the integrator Error Corrections

We have seen the results from the experiments with different improvements for the hover duration of the flight. The Fig. 4.12 shows the UAV positions for the whole flight (including takeoff and landing). In the z position plot of Fig. 4.12, we can see that the altitude of the UAV is lower than the desired initially. Due to this, the integrator error starts increasing, and the altitude of the UAV increases. However, due to the saturation limit of the integrator value being very low, it was not able to reach the desired altitude. After this experiment, the integrator saturation value was increased from 2 to 6.

Also, in error graph of Fig. 4.12, we can see that initially x position of the UAV is on positive side (negative error). This is because of the non-symmetries or calibration errors present. But after some time, the integral error in the x-direction becomes negative and converges at one value, which makes the UAV hover about 0 position in x-direction.

In the histogram (Fig. 4.13), we can see that the UAV is at higher distance (more than 40 mm from origin) for very limited number of data points. These are the data points of the initial flight. But after integral errors converge, it reaches the hover position around the origin. It has the peak probability of being at a 10 mm distance from the origin.

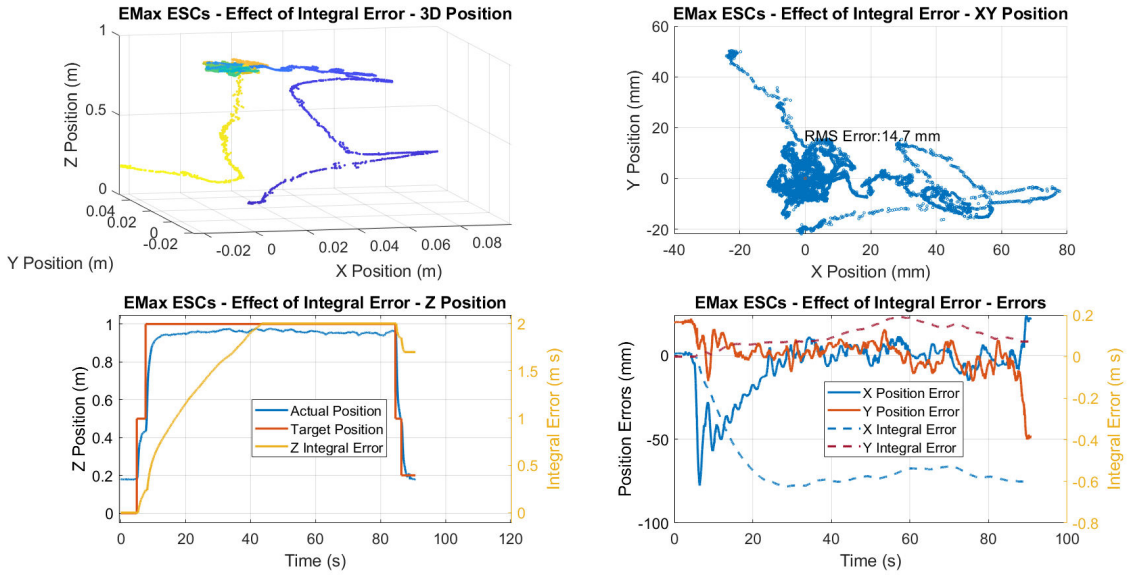


Figure 4.12: Full Flight - Effect of Integrator Error

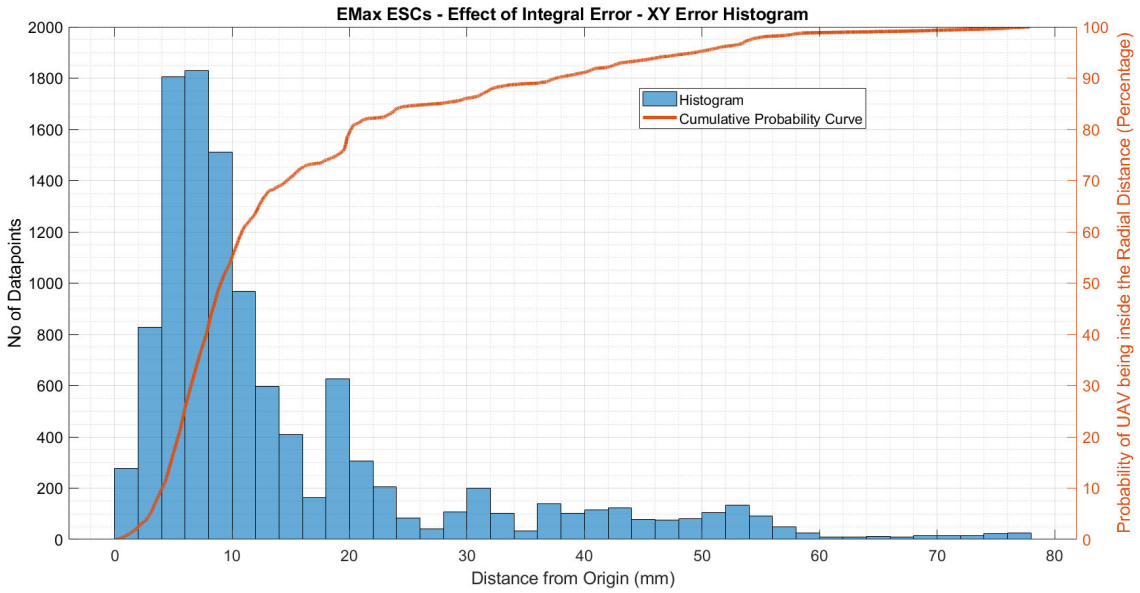


Figure 4.13: Histogram of the full flight - Effect of Integrator Error

4.2 Circular Trajectory

In the previous section, we observed the flight results for the hover flights and how they were improved step by step. In this section, we will explore the results of the circular trajectory flights conducted with the UAV. The trajectory was pre-defined using the minimum jerk trajectory method. The UAV first takes off from the origin to the desired altitude, then moves to the start of the circle. After this, it starts moving in a circle using the target position, target velocity, target acceleration provided by the trajectory generator, and error corrections from the PID controller.

The circular trajectory is being generated using a minimum jerk trajectory method (order 5 equation of time). First, the matrix **B** is generated containing the initial and final requirements of the position, velocity, and accelerations. In the equation 4.6, the initial condition for angle (of the circle) is 0 and for the altitude is $z_{initial}$. The final condition for angle is 2π and for the altitude is z_{final}. The requirements of initial velocity and acceleration are kept 0. The $z_{initial}$ and z_{final} can be defined before each run of the code.

$$\mathbf{B} = \begin{bmatrix} 0 & z_{initial} \\ 0 & 0 \\ 0 & 0 \\ 2\pi & z_{final} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{4.6}$$

Then the matrix M is generated as shown in equation 4.7. Here, t_0 indicates the start time of the trajectory and t_f indicates the end time of the trajectory.

$$\mathbf{M} = \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2 \cdot t_0 & 3 \cdot t_0^2 & 4 \cdot t_0^3 & 5 \cdot t_0^4 \\ 0 & 0 & 2 & 6 \cdot t_0 & 12 \cdot t_0^2 & 20 \cdot t_0^3 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2 \cdot t_f & 3 \cdot t_f^2 & 4 \cdot t_f^3 & 5 \cdot t_f^4 \\ 0 & 0 & 2 & 6 \cdot t_f & 12 \cdot t_f^2 & 20 \cdot t_f^3 \end{bmatrix} \tag{4.7}$$

The coefficients of the 5th order polynomial trajectory in time are found using the equation 4.8.

$$\mathbf{A} = \mathbf{M}^{-1} \cdot \mathbf{B} \tag{4.8}$$

Using the coefficients found in the equation 4.8, we can find the desired output equation, differential of the output equation and double differential of the output equation as shown in equations 4.9, 4.10 and 4.11. The $A(1,:)$ indicates all the values of the first row of matrix A.

$$\mathbf{O} = \mathbf{A}(1,:) + \mathbf{A}(2,:) \cdot t + \mathbf{A}(3,:) \cdot t^2 + \mathbf{A}(4,:) \cdot t^3 + \mathbf{A}(5,:) \cdot t^4 + \mathbf{A}(6,:) \cdot t^5 \tag{4.9}$$

$$\dot{\mathbf{O}} = \frac{d}{dt}\mathbf{O} = \mathbf{A}(2,:) + 2\mathbf{A}(3,:) \cdot t + 3\mathbf{A}(4,:) \cdot t^2 + 4\mathbf{A}(5,:) \cdot t^3 + 5\mathbf{A}(6,:) \cdot t^4 \tag{4.10}$$

$$\ddot{\mathbf{O}} = \frac{d}{dt}\dot{\mathbf{O}} = 2\mathbf{A}(3,:) + 6\mathbf{A}(4,:) \cdot t + 12\mathbf{A}(5,:) \cdot t^2 + 20\mathbf{A}(6,:) \cdot t^3 \tag{4.11}$$

Using the output equations, we can find the desired angle of the circular trajectory and their differentials as shown in equation 4.12. Similarly, we can find out desired z and their differentials as shown in equation 4.13. β is the desired angle of the circular trajectory.

$$\beta = \mathbf{O}(1,1),\ \dot{\beta} = \dot{\mathbf{O}}(1,1),\ \ddot{\beta} = \ddot{\mathbf{O}}(1,1) \tag{4.12}$$

$$z = \mathbf{O}(1,2),\ \dot{z} = \dot{\mathbf{O}}(1,2),\ \ddot{z} = \ddot{\mathbf{O}}(1,2) \tag{4.13}$$

Using β and z and their differentials, we can find out the desired position, velocity and acceleration in 3D as shown in equations 4.14, 4.15 and 4.16.

$$x_{des} = r \cdot cos(\beta),\ y_{des} = r \cdot sin(\beta),\ z_{des} = z \tag{4.14}$$

$$v_{x,des} = \frac{d}{dt}x = -y\dot{\beta},\ v_{y,des} = \frac{d}{dt}y = x\dot{\beta},\ v_{z,des} = \frac{d}{dt}z = \dot{z} \tag{4.15}$$

$$a_{x,des} = \frac{d}{dt}v_{x,des} = -x\cdot\dot{\beta}^2 - y\cdot\ddot{\beta},\ a_{y,des} = \frac{d}{dt}v_{y,des} = -y\cdot\dot{\beta}^2 + x\cdot\ddot{\beta},\ a_{z,des} = \frac{d}{dt}v_{z,des} = \ddot{z} \tag{4.16}$$

$$yaw_{des} = ATAN2(\dot{y}, \dot{x}) - \pi/2 \tag{4.17}$$

From the desired acceleration calculated in the above equations and the errors in the position, velocity, and integral, the PID controller is implemented as shown in equation 4.18. In this equation, the term $\ddot{\mathbf{r}}_{des}$ is the feed-forward term, so the PID controller only needs to take care of the errors. After calculating the command acceleration in 3 directions, the desired roll, pitch and thrust are calculated using the same equations described in hover controller (equations 4.2, equation4.3 and 4.4).

$$\ddot{\mathbf{r}}_{command} = \ddot{\mathbf{r}}_{des} + \mathbf{k}_p \cdot \mathbf{r}_{error} + \mathbf{k}_v \cdot \dot{\mathbf{r}}_{error} + \mathbf{k}_i \cdot \int \mathbf{r}_{error} \tag{4.18}$$

Simulation with Circular trajectory

Before implementing the circular trajectory on the hardware, it was first tested in the Gazebo simulation. The Fig. 4.14 shows the path followed by the UAV in the simulation. This simulation was done before doing any tuning and hence we can see that it made a circle of a much smaller size than the desired one. As the gains were very low, the UAV was not correcting the errors aggressively.

After correcting the gains with trial and error, the UAV was able to follow the circular trajectory very well. The Fig. 4.15 and Fig. 4.16 show the results from these simulations. We can see that the UAV was making a smaller circle than the desired one in the faster trajectory (Fig. 4.16). This is also due to PID controller gains being lower than required and the UAV not correcting the errors aggressively.

First Hardware Flight

After checking everything in the simulation, the circular trajectory was implemented on the hardware. The Fig. 4.17 shows the results from the first hardware flight. This experiment suffered a crash and after the analysis of the logs, it was found that the velocity update term was commented out in the code. So essentially there was no damping in the

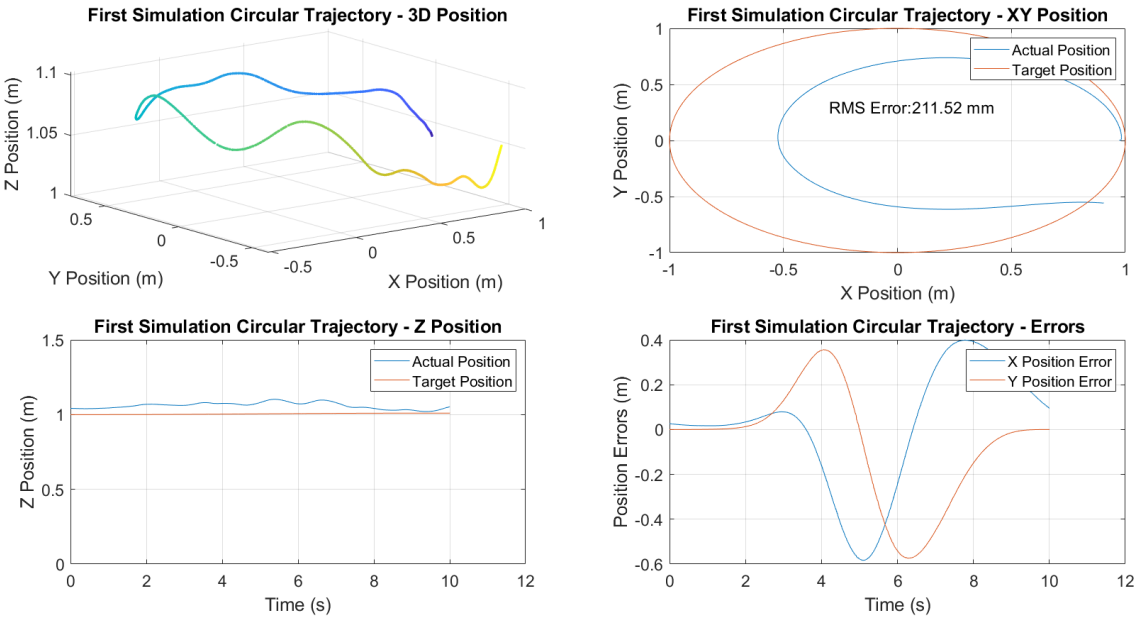


Figure 4.14: First Gazebo Simulation with Circular Trajectory

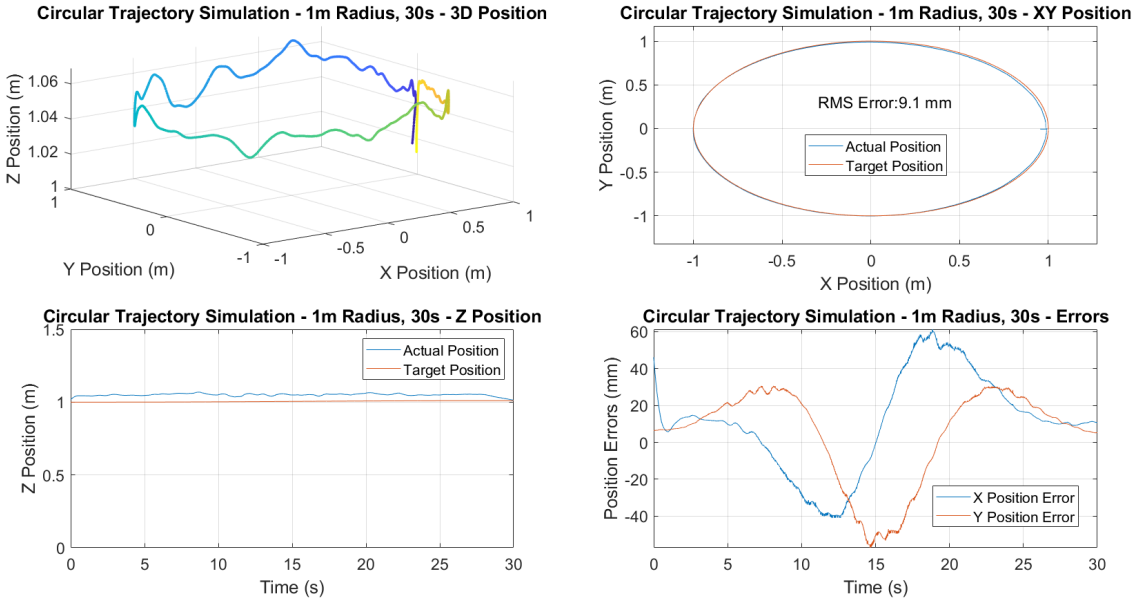


Figure 4.15: Gazebo Simulation of Circular Trajectory - 30s circle with 1m radius

controller (d term) and it started oscillating about the desired path, which increased with time, and UAV collided with the nets. Due to this crash, the importance of the damping term was understood.

First Successful Flight

After correcting the mistake made for the velocity error, a successful flight was achieved. Fig. 4.18 shows the results from this experiment. We can see that the UAV is

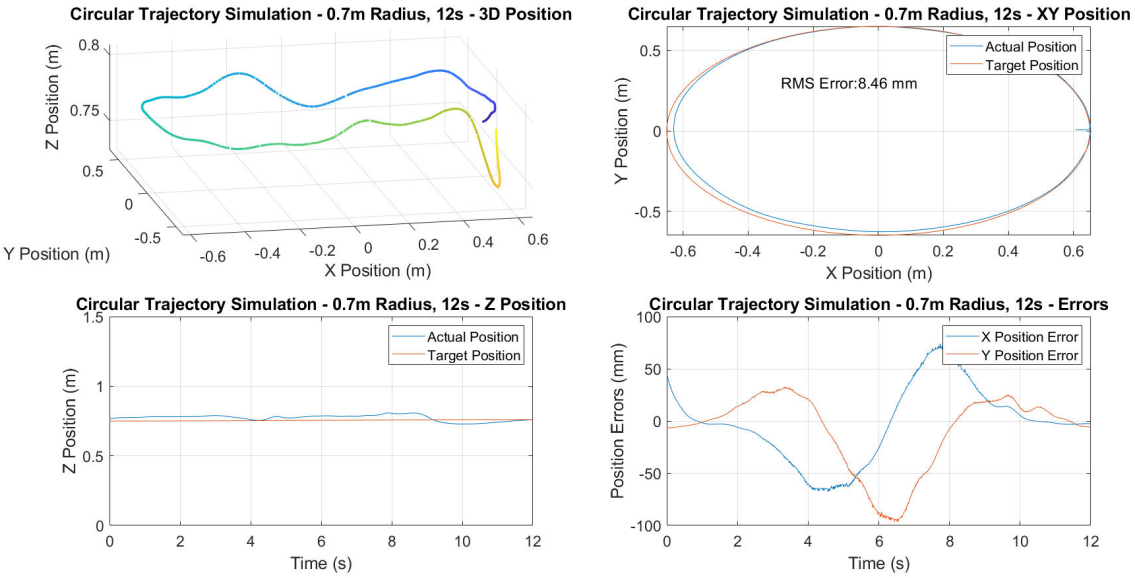


Figure 4.16: Gazebo Simulation of Circular Trajectory - 12s circle with 0.7m radius

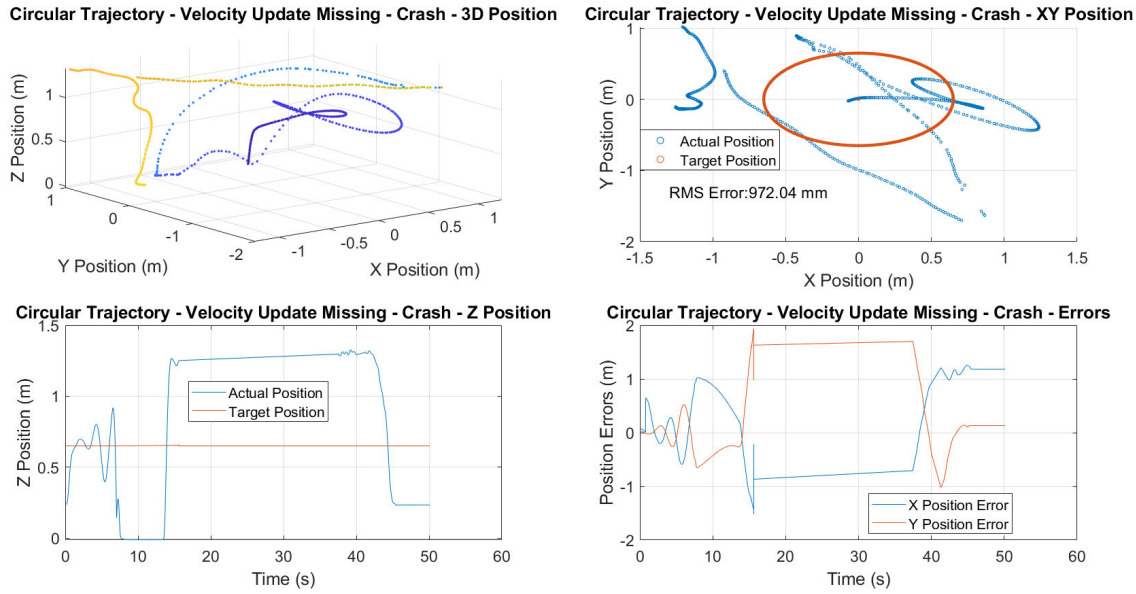


Figure 4.17: First Circular Trajectory with Hardware

more or less able to follow a circular path. But the circular path is a little on the right side of the desired path as no integral error correction term was present in the controller.

First Successful Flight with Yaw

The Fig. 4.19 shows the results from the experiment in which the yaw was changing. The yaw was commanded such that the UAV is always moving in a forward direction. We can observe that the UAV was making a bigger circle than the desired one due to roll and

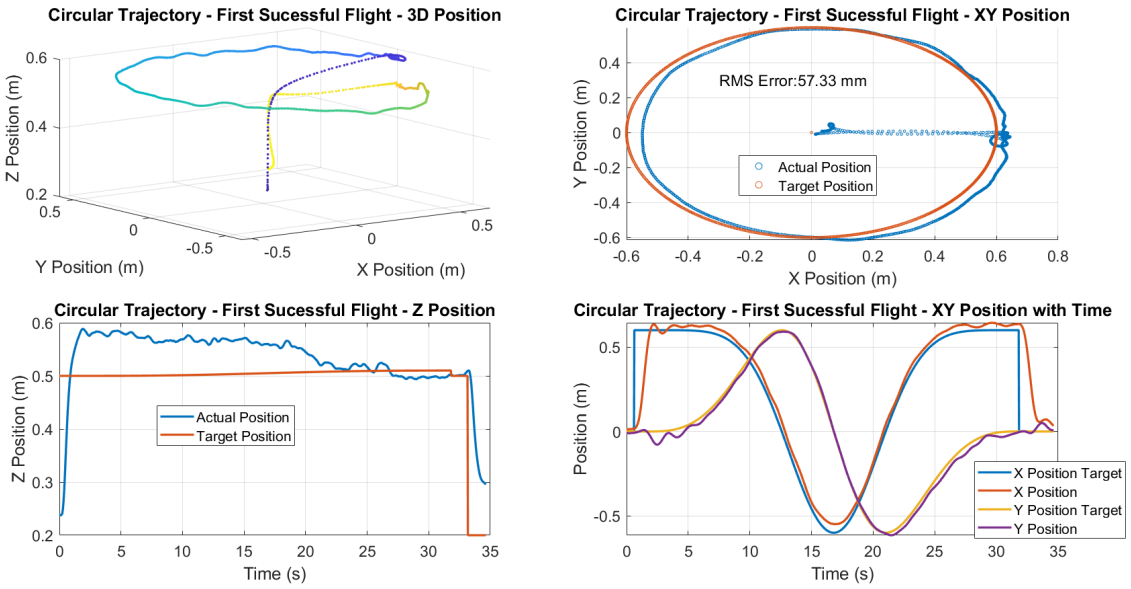


Figure 4.18: First Successful Circular Trajectory Flight with Hardware - with Constant Yaw

pitch control being affected by the yaw control. The video of this experiment can be seen at (34).

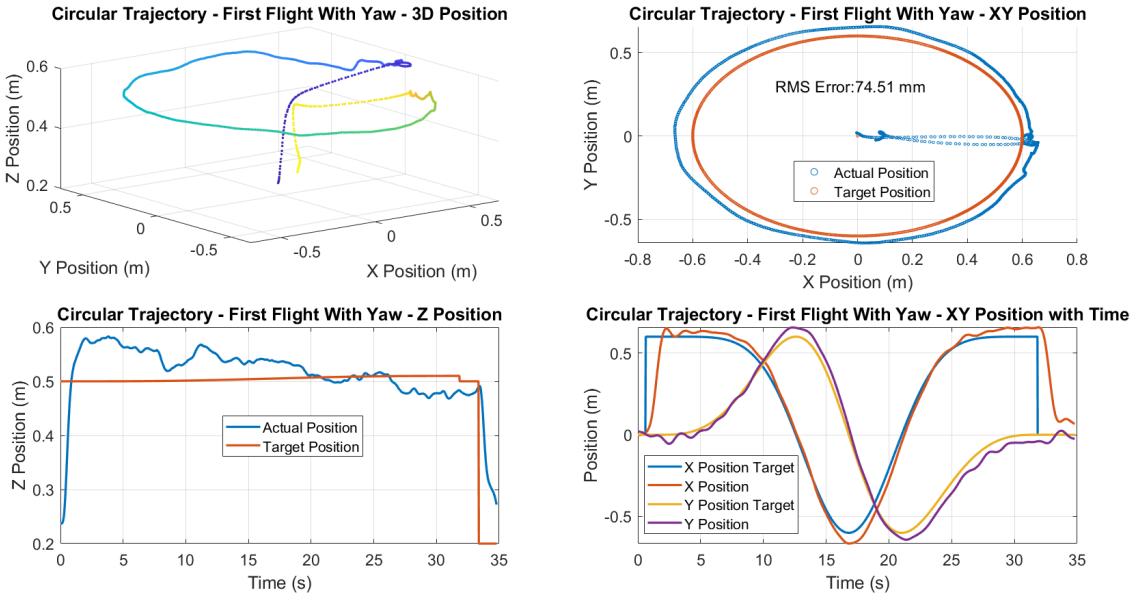


Figure 4.19: First Successful Circular Trajectory Flight with Hardware - with Changing Yaw

Integral Error Included

After finding out that the UAV is following the path towards +x than the desired one, the integral term in the controller was introduced. The Fig. 4.20 shows the path followed by the UAV with integral error term. We can see that the UAV is following the path around the desired path and there is no offset. We can also see that the altitude of the UAV is near the desired value after being higher initially.

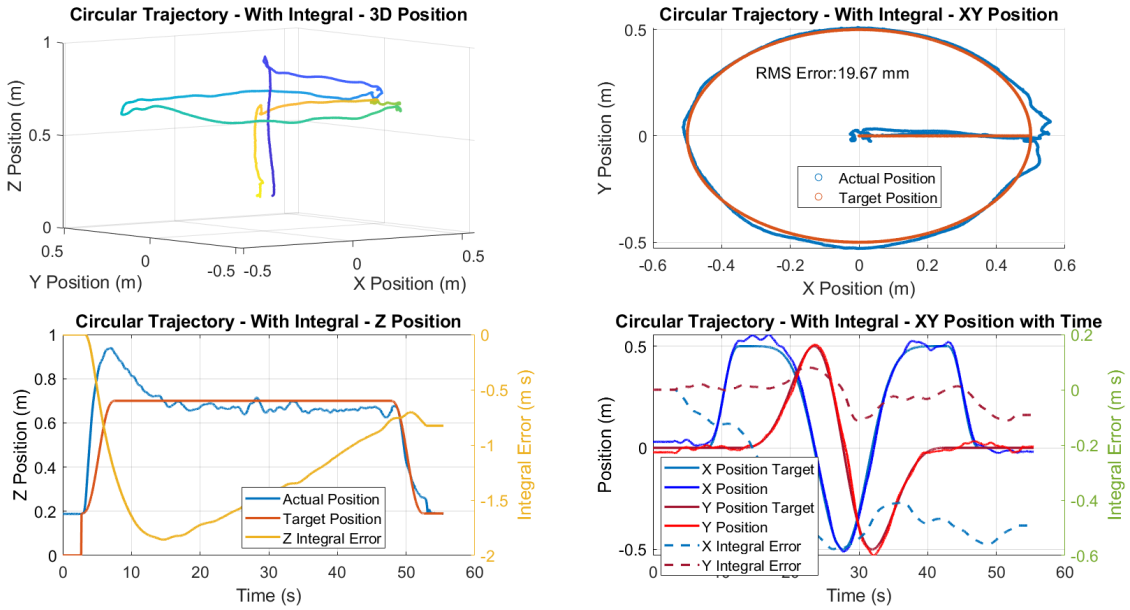


Figure 4.20: First Circular Trajectory with Integral Term

Comparison of 20s and 15s flight

After achieving the successful flights with a circular trajectory, it was decided to do the experiments with the various trajectory times. The Fig. 4.21 and Fig. 4.22 show the circular path followed by the UAV for these flights. We can observe that the path followed by the UAV is smaller than the desired one for the faster trajectory. This is the same observation made during the simulation flight also. In order to make the UAV follow the exact path, the gains need to be higher.

Emax Motors with Cutoff Frequency Tuned and Notch Filter Implemented

The experiments done till now were done with the generic cost-effective motors. After changing these motors to Emax motors, an improvement in the performance was observed (Fig. 4.23). The RMS error in the XY direction became less than 20mm Also,

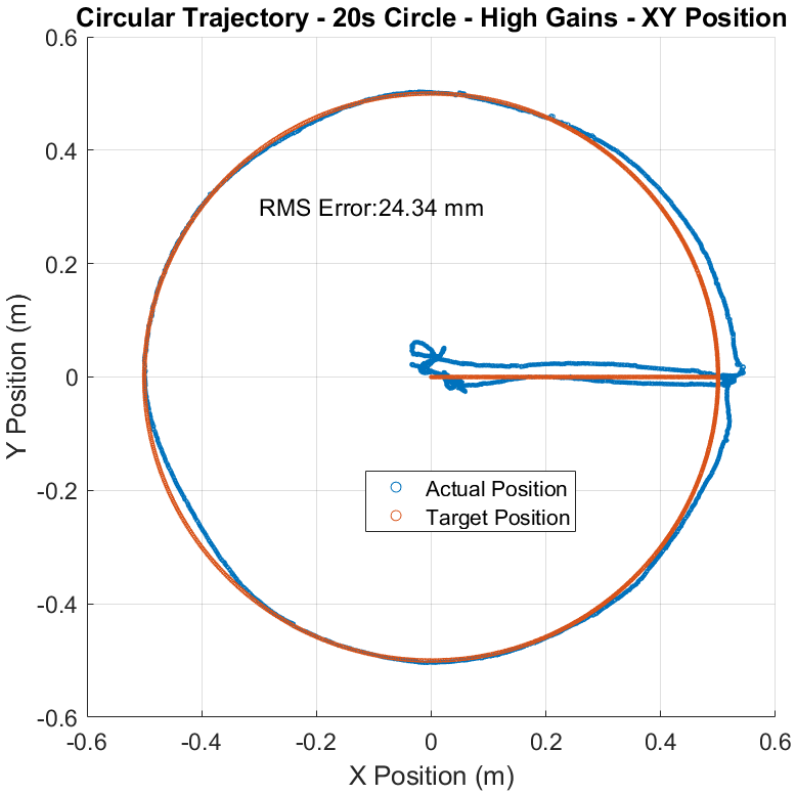


Figure 4.21: Circular Trajectory of 20s

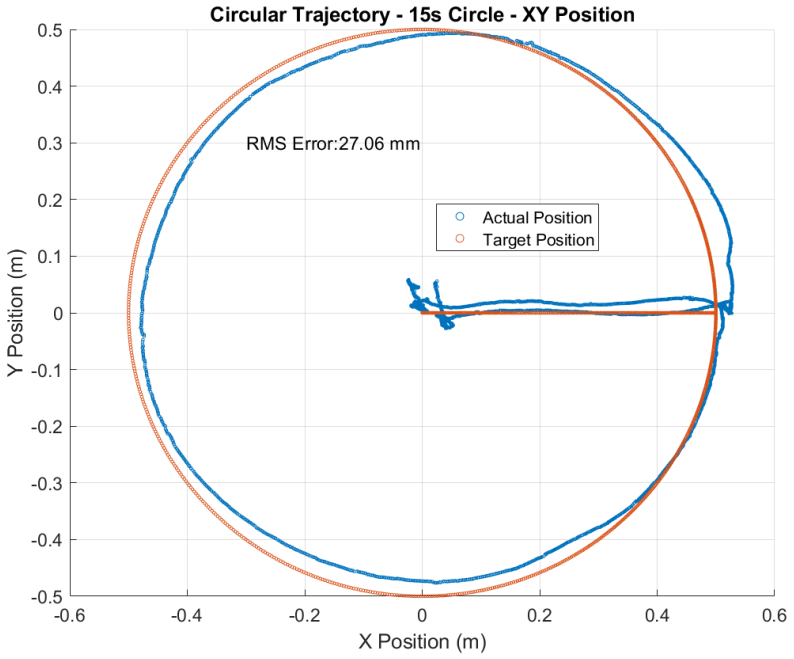


Figure 4.22: Circular Trajectory of 15s

the altitude of the UAV became much stable than the previous experiments. The video of this experiment can be seen at (35). The animation of the XY plot of this experiment can be seen at (36).

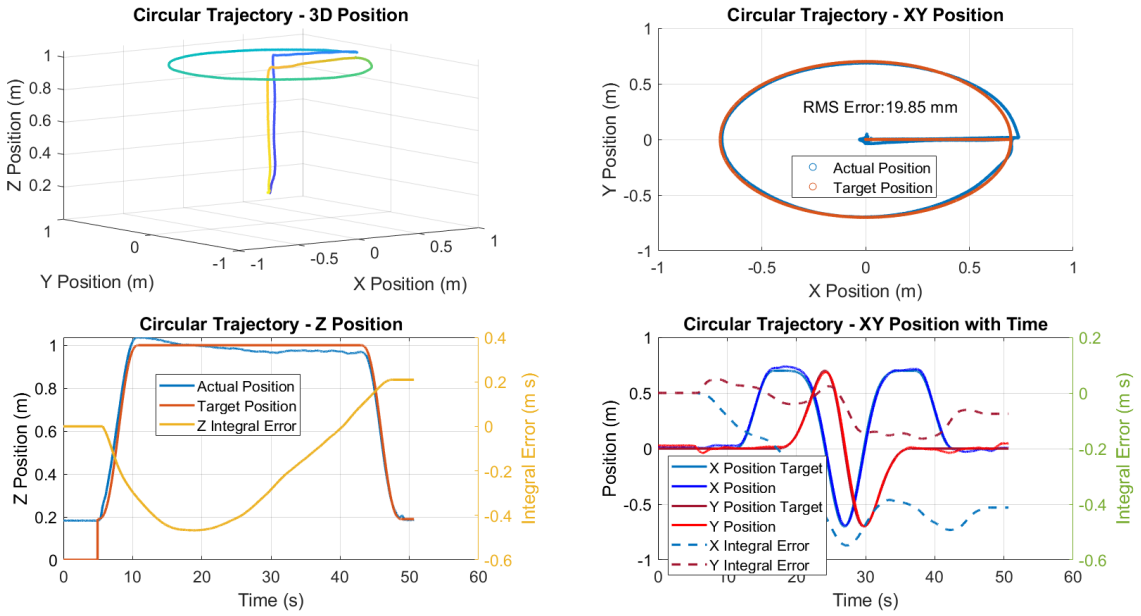


Figure 4.23: Circular Trajectory with Emax Motors

Circular Trajectory with Multiple Circles

This experiment was done to make the UAV follow the trajectory of multiple circles. The aim of this experiment was to ensure that the initial starting point errors do not affect further circular path (2nd round onwards). Also, it was considered that the integral errors will be able to converge well due to longer duration of the flight. In Fig. 4.24, it can be observed that the altitude of the UAV was same as the desired one after the integral error converged (after 40s of flight). One more observation can be made that the UAV was not following the same circular path every time. The animation of the XY plot of this experiment can be seen at (37).

Circular Trajectory with Cylindrical Coordinate System

As observed previously, the UAV was not able to follow the exact path for the faster trajectories. Also, it was observed that if the XY direction gains are made very large, then the UAV was starting the oscillations in the radial direction. Due to this, it was considered to use the control system in cylindrical coordinates (r, β, and z). The aim of this was to keep the radial and tangential gains different as the higher gains were required for the tangential direction.

The required β, required z, and their derivatives are calculated similarly as described in the equations from 4.6 to 4.13. From those values, the desired position, velocity, and acceleration in cylindrical coordinates are calculated using the below equations:

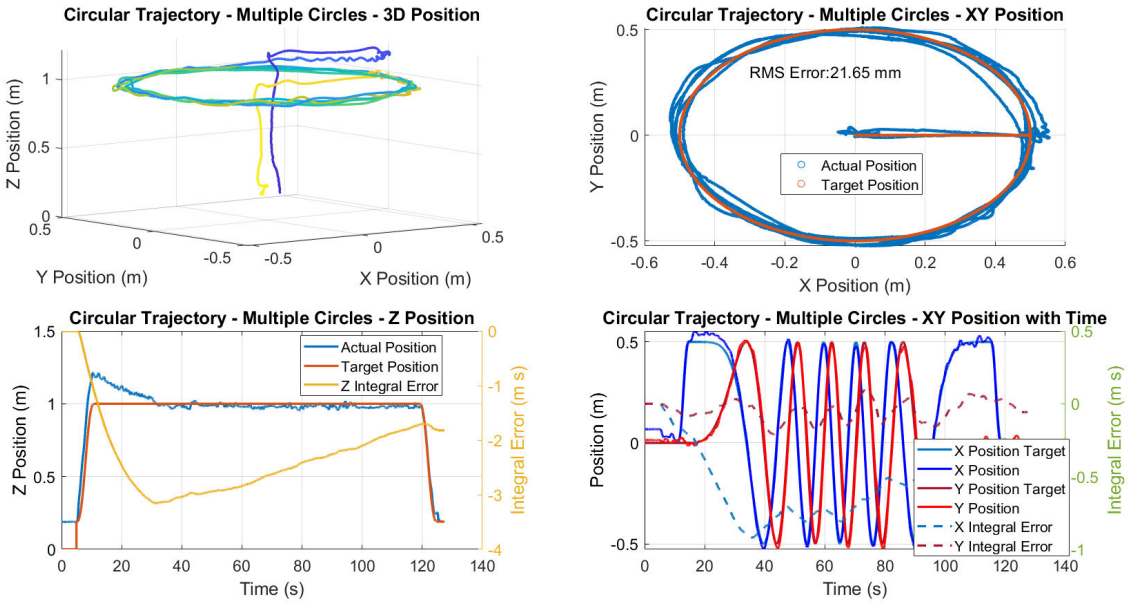


Figure 4.24: Multiple Circle Trajectory

$$r_{des} = r,\ \beta_{des} = \beta,\ z_{des} = z \tag{4.19}$$

$$v_{radial,des} = 0,\ v_{tangential,des} = r \cdot \dot{\beta},\ v_{z,des} = \frac{d}{dt}z = \dot{z} \tag{4.20}$$

$$a_{radial,des} = -r \cdot \dot{\beta}^2,\ a_{tangential,des} = r \cdot \ddot{\beta},\ a_{z,des} = \ddot{z} \tag{4.21}$$

Based on these values, the command acceleration is calculated similarly to the equation 4.18. The only difference will be the use of radial and tangential directional accelerations instead of x and y directional accelerations. We will be able to tune the gains of radial and tangential directions separately with this method.

One of the problems faced with this method was the error calculation becoming incorrect when the tangential position is near 0 or 2π angles. If the desired and current positions were in different quadrants near this condition, the error was becoming incorrect. This was solved by putting the condition of them being in a different quadrant and then correcting one of the angles so that errors are calculated correctly. A similar condition was implemented for velocity error also.

Fig.4.25 shows the path followed by the UAV in such a flight with multiple circles. We can see that the path followed by the UAV is not at all circular due to gains not being tuned properly.

After tuning the gains, the results of the flight are shown in Fig. 4.26. Still, we can see that the RMS error was very high compared to the XY directional controller.

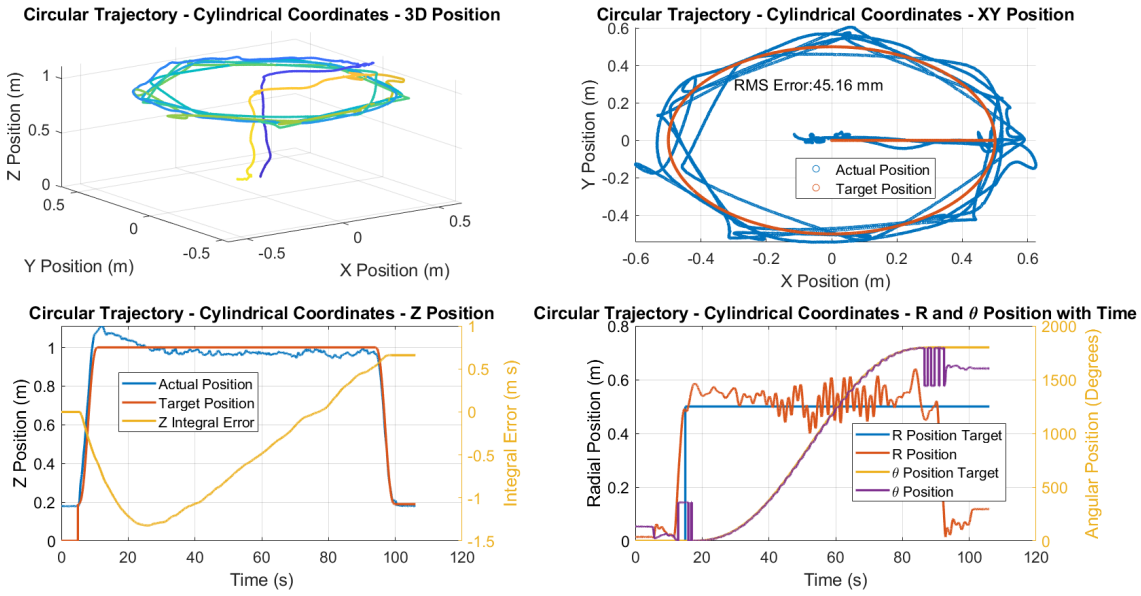


Figure 4.25: First flight with Circular Trajectory and Cylindrical Coordinate Control System

After some flights, it was decided that this method might not work very well and it was discontinued.

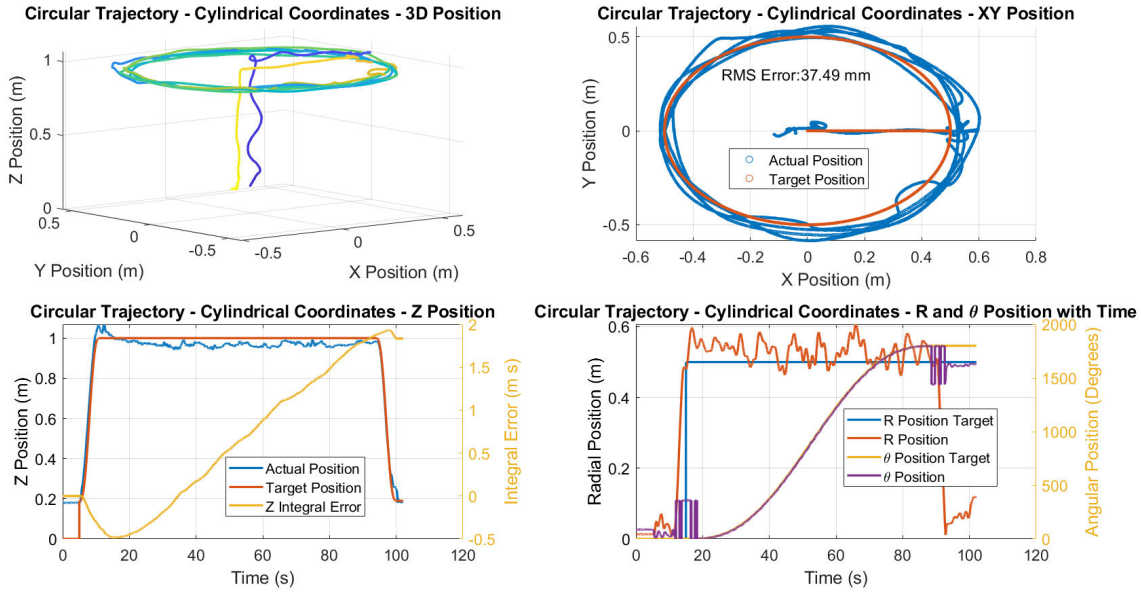


Figure 4.26: Post tuning flight with Circular Trajectory and Cylindrical Coordinate Control System

4.3 Helix trajectory

In the previous experiments, the initial and final altitude of the UAV was kept the same. However, with the same trajectory code, we can create the helix trajectory also by keeping the initial and final altitude different. One such experiment was done and the results are shown in Fig. 4.27. We can observe that the UAV is able to follow the helix trajectory very well. The altitude of the UAV is always a little lower than the desired one because the z-direction integral error could not converge in the given amount of time. The video of this experiment can be seen at (38).

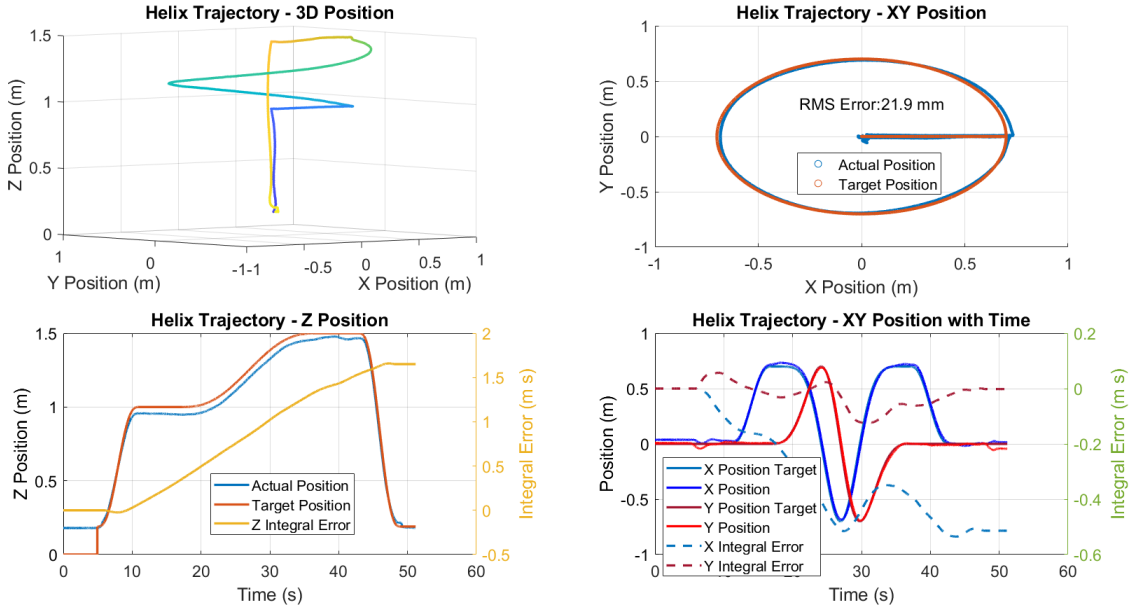


Figure 4.27: Helix Trajectory Flight

Failsafe Algorithms While doing the experiments described previously, there were many problems found in the code which made the UAV crash sometimes. For example, if the Vicon data was not received on the offboard system, it was commanding the takeoff thrust all the time (as it was not detecting the change in the altitude). This resulted in the crash and was solved by not allowing the UAV to arm if the Vicon data was not received. This can be considered a failsafe algorithm. There were many such conditions implemented during the experimental phase, which are described below:

1. The desired roll and pitch angles are calculated as described in the equation 4.2 and 4.3. However, if the errors become very large, these desired angles will become very large and the UAV can become unstable. In order to avoid this, the tilt angles were limited to some maximum values (for example 20 degrees). Similarly, the maximum thrust was also limited.

2. If the Vicon data is lost during the experimental run, the UAV was going into its own land mode. However, in this land mode, it first increases the altitude to a certain value and then lands at its home position. This was also causing the crashes. So in order to solve this problem, the emergency land mode was designed in the offboard code itself. In this code, if the Vicon data was lost during the flight, the thrust value will reduce to 90% of the hover thrust and the target attitude will be the level flight. This way, the UAV will land immediately without causing any damage.

3. Sometimes, a single Vicon data point was getting stuck within the communication. When this happened, 2 data points were coming to the offboard system almost simultaneously. Due to this, the differential error was becoming very large (as dt was very small) and was causing instabilities. In order to resolve this, the condition was put on the dt value while calculating the differential errors. The threshold of dt was decided based upon the frequency at which the Vicon was publishing the data.

4. Initially, the yaw of the UAV was determined using the magnetometer inside the Pixhawk. However, the reading of the magnetometer was susceptible to noise present in the surrounding area (for example motors). In order to resolve this, the new node was designed in the ROS to send the Vicon-based yaw data at a slow frequency to Pixhawk. Using that, the PX4 was correcting the drift in the magnetometer readings and the UAV was able to follow the desired yaw.

4.4 Experiments with 1 DOF Pendulum attached to the UAV

As described in (8), the UAV can be used to impact the objects present in the environment using the pendulum attached to the UAV. The UAV has motors that can produce a limited amount of thrust. So we cannot possibly conduct the task requiring a large amount of force, for example, to operate a lever found in the industries. However, if the energy is stored with the pendulum gradually and then used to impact the lever, it can be operated easily. The reference (8) worked on theoretical and simulation studies in this area. In this chapter, we will explore the experimental implementation of the same.

In order to do this experiment, the pendulum had to be attached to the UAV. The pendulum was made of a carbon fiber rod to make sure that the link is as light as possible. The endmass was made using the wooden cylindrical block. The holes were made inside the wooden block, in which the bolts weighing 10g each can be inserted to modify the mass of the pendulum (Fig. 4.28).

Figure 4.28: Bolts to adjust the mass of the pendulum

The other end of the pendulum is attached to the UAV using the joint as shown in Fig. 4.29. The 1 DOF joint is made using the shaft and bearings. The bearings and shafts were taken from BLDC motors themselves to ensure the smooth running.

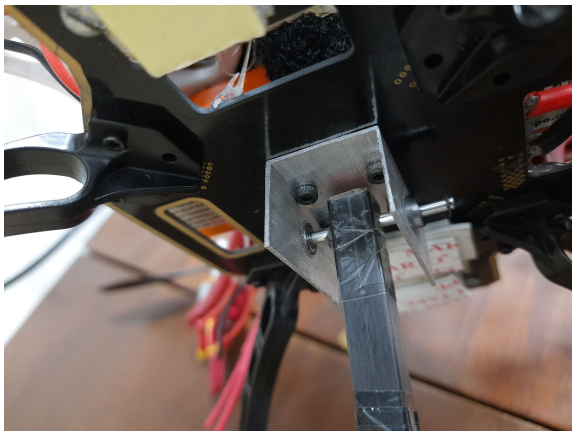


Figure 4.29: Pendulum Attachement to the UAV

One more problem needed to be solved before flying the UAV with a pendulum. The Pendulum attached to the UAV was extending beyond the landing gears, which will hamper the landing of the UAV on the ground. In order to resolve this issue, a box having a square vertical tunnel was made in which the pendulum will get inserted (Fig. 4.30). In order to land the UAV on this box, the landing has to be very precise. From the previous hover and circular trajectory experiments, we were confident that the UAV will be able to follow the required trajectory to land precisely.

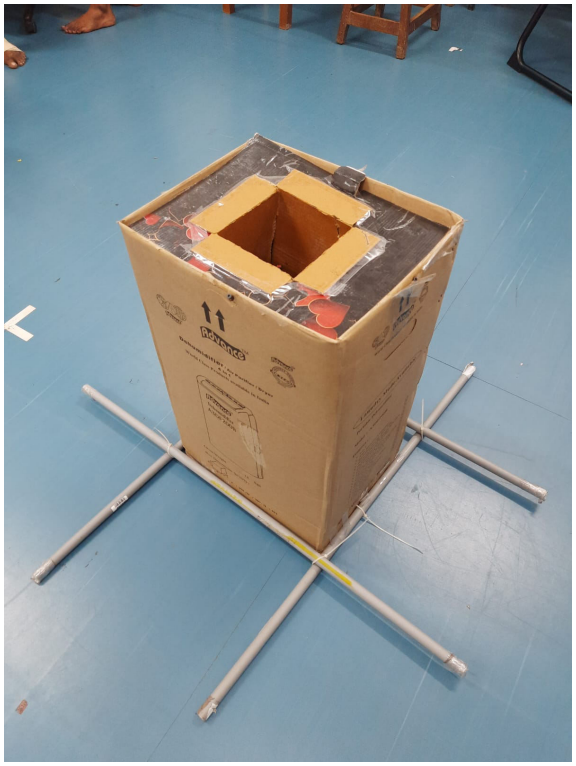


Figure 4.30: Landing Box for the UAV with Pendulum

Pendulum Swing Control

The pendulum attached to the UAV is connected via a passive joint (i.e. there is no motor). Hence, we do not have any direct control over the oscillations of the pendulum. So in the condition of oscillations going out of control or before the landing, we need to dampen out the oscillations of the pendulum. This can be indirectly achieved by moving the UAV. The strategy used for this purpose is as described in (9). For controlling the swing of the pendulum, the correction term is added to the command acceleration as shown in equation 4.22 and 4.23.

$$\ddot{r}_{command,x,corrected} = \ddot{r}_{command,x} + (k_{correction} \cdot \theta_p(t - t_p/4)) \cdot cos(yaw_{des}) \tag{4.22}$$

$$\ddot{r}_{command,y,corrected} = \ddot{r}_{command,y} + (k_{correction} \cdot \theta_p(t - t_p/4)) \cdot sin(yaw_{des}) \tag{4.23}$$

where θ_p is the angle of the link with respect to the vertical (0 degrees if the pendulum is downwards). $k_{correction}$ is the proportional constant, which is tuned empirically for the best results. t_p is the time period of the pendulum. The angle reading is taken with the delay of $1/4^{th}$ of the time period to make sure that the maximum correction is applied when the pendulum has the maximum speed (when it is at the downward position).

Fig. 4.31 shows the pendulum angle for the experiment done in order to verify the swing stabilization algorithm. In the experiment (as shown in video (39)), the pendulum was impacted with a stick to increase the oscillations. The stabilization algorithm can be turned on or off using an RC controller. We can see when the swing stabilization is on, the oscillations damped out very quickly, but when it is not on, the oscillations keep damping at a very slow rate. But as soon as it is turned on again, it dampens out very quickly. So the oscillation swing control was verified using the experiment.

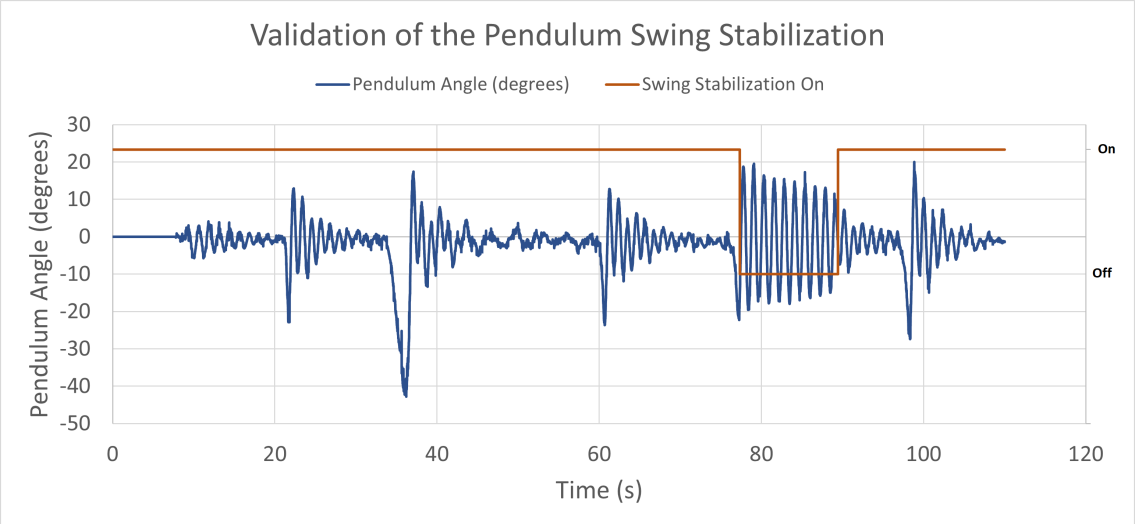


Figure 4.31: Pendulum Angle with Respect to Time for Swing Stabilization Verification

Energy Increase

The aim of the experiments with the pendulum is to increase the energy of the pendulum slowly and then impact the pendulum with the objects to utilize the energy. For this purpose, the algorithm needs to be written to increase the energy of the pendulum. As described in the previous section, we can not directly control the un-actuated pendulum and hence we again need to move the UAV itself to increase the swing. The idea about how to increase the oscillations is taken from (10). The basic idea is that the UAV is moved proportional to the link current angle such that the inertia of the pendulum makes it oscillate more. This strategy is almost the opposite of the swing stabilization discussed in the previous section, but without the delay.

$$\ddot{r}_{command,x,corrected} = \ddot{r}_{command,x} - (k_{correction} \cdot \theta_p) \cdot cos(yaw_{des}) \tag{4.24}$$

$$\ddot{r}_{command,y,corrected} = \ddot{r}_{command,y} - (k_{correction} \cdot \theta_p) \cdot sin(yaw_{des}) \tag{4.25}$$

The difference between the equations 4.22, 4.23 and 4.24, 4.25 is the change of sign from plus to minus in the correction term and the absence of the delay. The $k_{correction}$ needs to be tuned properly and will be discussed in upcoming sections. The video of the experiment to increase the oscillations can be seen at (40).

Impact Test

In order to impact the pendulum on the target object with maximum possible velocity, the trajectory is generated (Fig. 4.32). First the UAV takes off from the box to the height of 1.9 m. After that, the oscillation increase algorithm starts. One of the parameters defined in the trajectory is the threshold pendulum angle before the UAV starts moving forward. For example, if the threshold pendulum angle is kept 40 degrees, then the trajectory will switch from oscillation increase algorithm to the impact trajectory when the pendulum angle becomes more than 40 degrees. This switching is only considered when the pendulum is on the front of the UAV, so that when the UAV starts moving forward, the angle of the pendulum will increase. The hitting location is lower than the original position, which will also help in impacting the object at higher energy. After hitting the object, the UAV goes back to the original position at 1.9m altitude and activates the swing stabilization algorithm. After running the swing stabilization algorithm for 12s, the UAV starts the landing process.

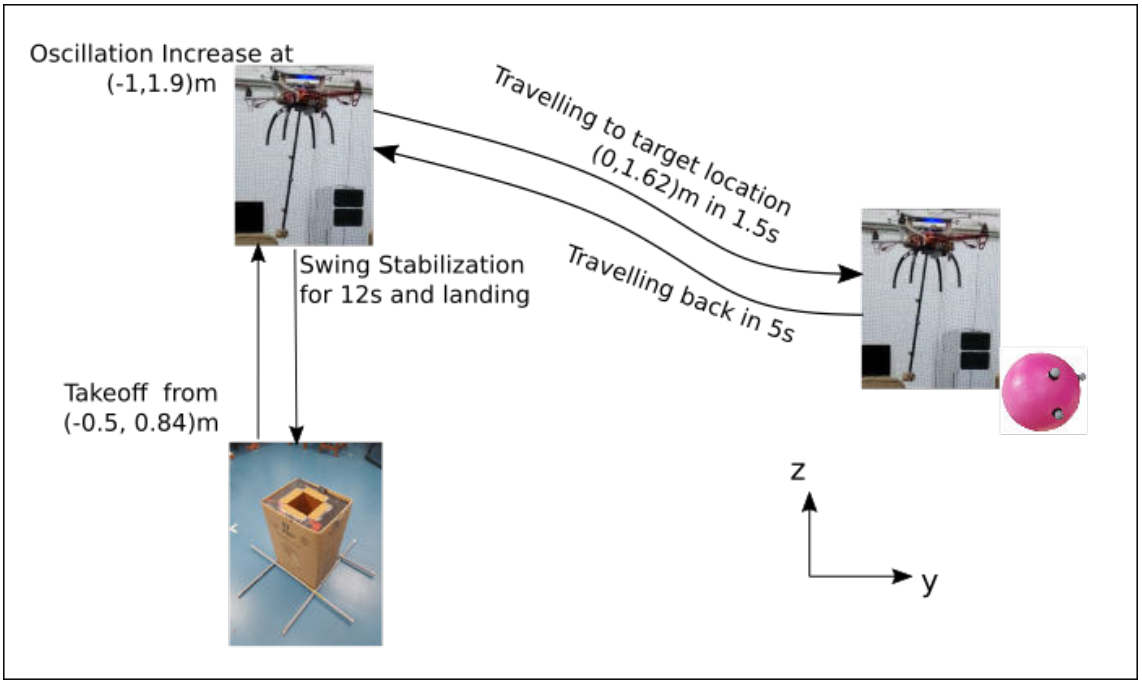


Figure 4.32: Trajectory of the Pendulum to Hit the Object

First just to test the trajectory control, the experiment was done without putting any object (to make sure there is no damage to the UAV). The video of such experiment can be seen at (41). In the video, we can see that while making the virtual impact, the UAV is following a stable trajectory. But while returning, it is facing the slow oscillations. This was due to no object being there to stop the high energy impact trial from the UAV. After this, to test the stability of the UAV with actual impact, the experiment was done to hit the object. The video of the same can be seen at (42).

Fig. 4.33 shows the comparison of the UAV's velocity and the endmass velocity. During the peak (just before the virtual impact), we can see that the maximum UAV velocity is about 1.5 m/s. However, the maximum pendulum velocity is around 5.5 m/s. So this comparison confirms that we can achieve a higher velocity and hence higher impact energy by swinging up the pendulum.

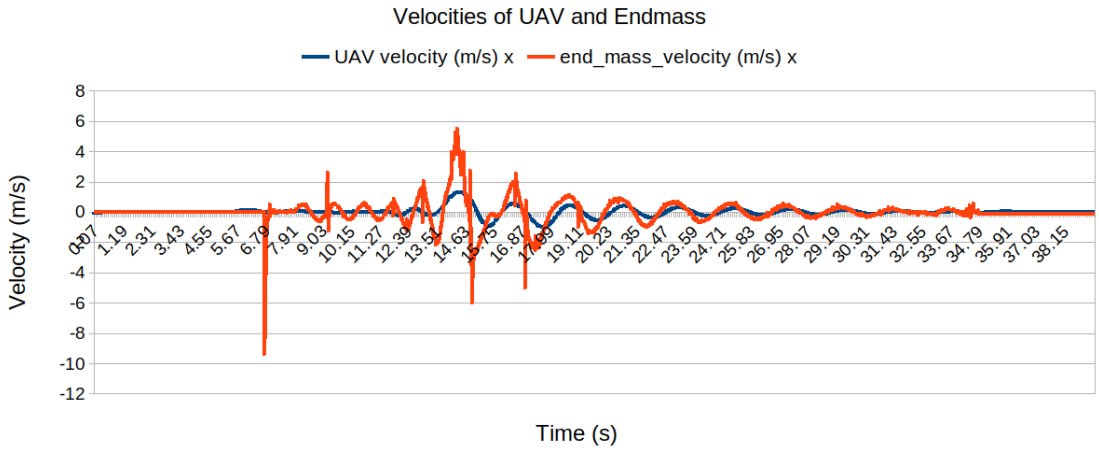


Figure 4.33: Comparison of the UAV velocity with the Pendulum Endmass Velocity

Impact Test with a Ball and a Hoop and Repeatability After confirming the stability of the trajectory, the experiment was designed to hit the ball with the pendulum and then make the ball pass through a hoop. The video of the first such experiment can be seen at (43).

After the ball passed through the hoop, the task was undertaken to increase the repeatability of the ball passing through the hoop. To understand the reasons behind non-repeatability, first the flight in which the ball passed through a hoop and the flight in which the ball did not pass through a hoop were compared. This comparison is shown in Fig. 4.34.

We can observe that the path of the UAV and the pendulum were lower in the experiment in which the ball did not pass through a hoop. From the velocity graphs also, we can observe that in the y-direction, the velocities of both the experiments were almost

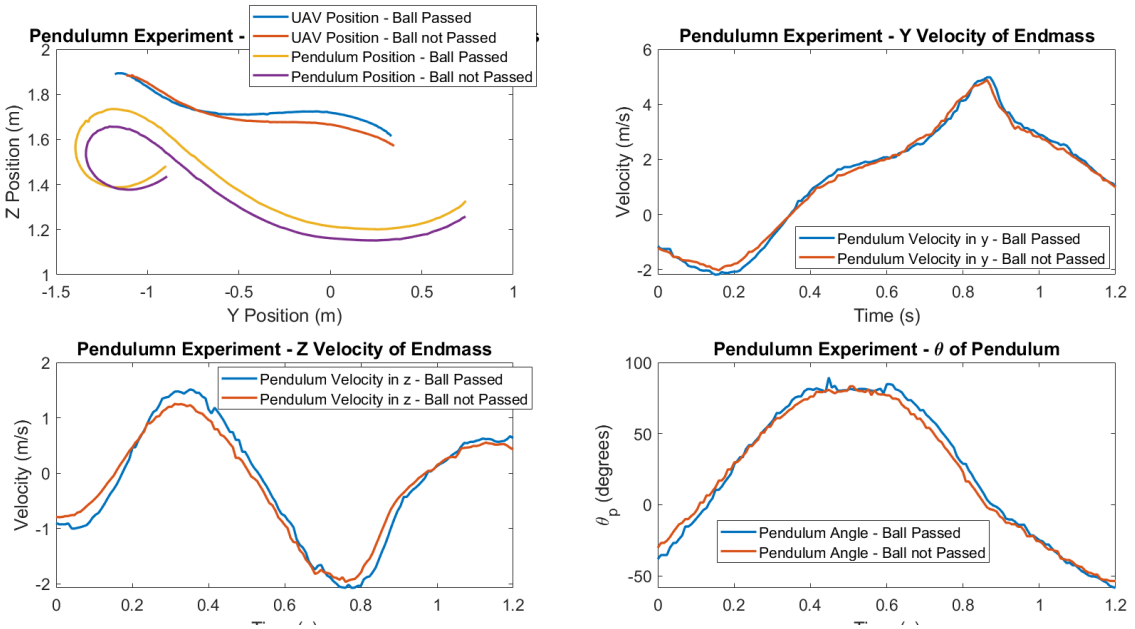


Figure 4.34: Comparison of the experiments in which ball passed through and did not pass through a hoop

the same and the difference was only in the z-direction. It was considered that this might be due to the voltage of the battery getting fluctuated and hence the amount of thrust available getting fluctuated.

Fig. 4.35 shows the total energy (potential plus kinetic) of the pendulum for both the experiments. We can see that the energy of the pendulum is always lower in the experiment in which the ball did not pass through the hoop. This can be linked to the altitude of the Pendulum being lower in the experiment in which the ball did not pass through the hoop. One more interesting observation from this plot is the sudden drop in energy when the impact happens at around 0.85s.

To resolve the problem described earlier, the gains of the PID loop were increased and multiple experiments were conducted. The results of these experiments are shown in Fig. 4.36. It can be observed that the path of the Pendulum was still varying too much in each experiment and hence repeatability of the ball passing through the hoop cannot be achieved.

Further, it was considered to reduce the proportional gain (from 7 to 0.9) of the correction term (Equation 4.24 and 4.25). In the Gazebo simulation, the improvement in the repeatability performance was clearly observed, and hence, it was tried for the hardware. The results of these experiments (without and with ball) can be seen in Fig. 4.37 and 4.38. It can be clearly observed that the pendulum is able to follow closer paths in each experiment. In the experiments with the ball, the ball passed through a hoop 2 out

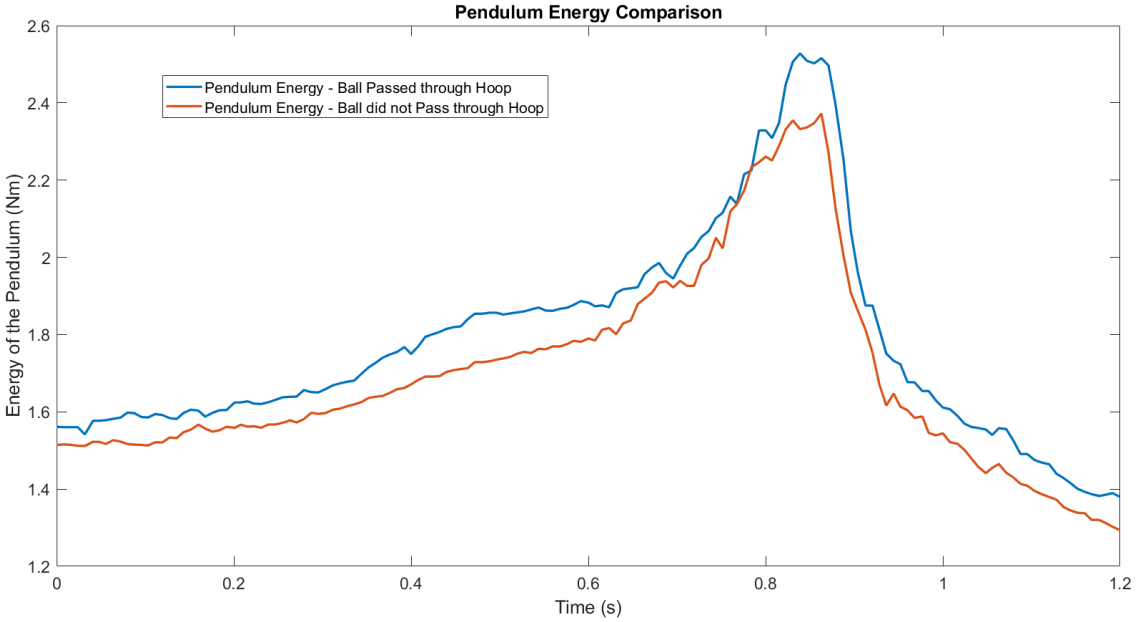


Figure 4.35: Comparison of the experiments in which ball passed through and did not pass through a hoop

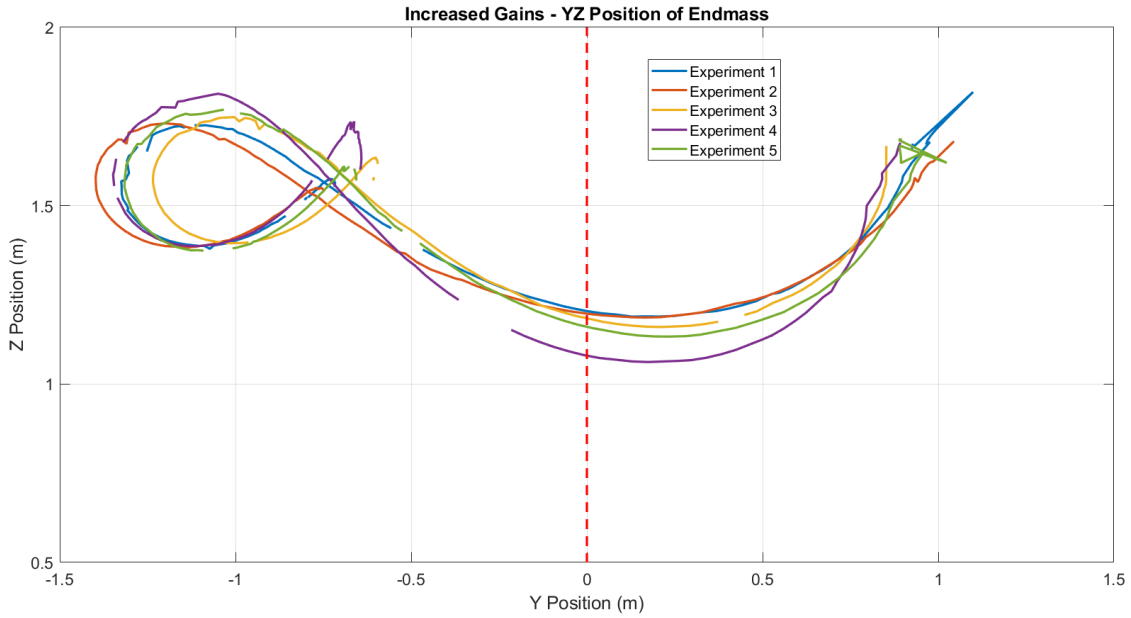


Figure 4.36: Increased PID Gains to Improve the Performance

of 5 times. The observed variation was now mainly in x-direction, which was affecting the repeatability.

In order to improve the performance in the x-direction, the link joint of the pendulum near the hinge, which became weak during previous experiments was repaired. This ensured that the movement of the pendulum does not happen in the x-direction. The results from these experiments are shown in Fig. 4.39. The performance in the YZ direc-

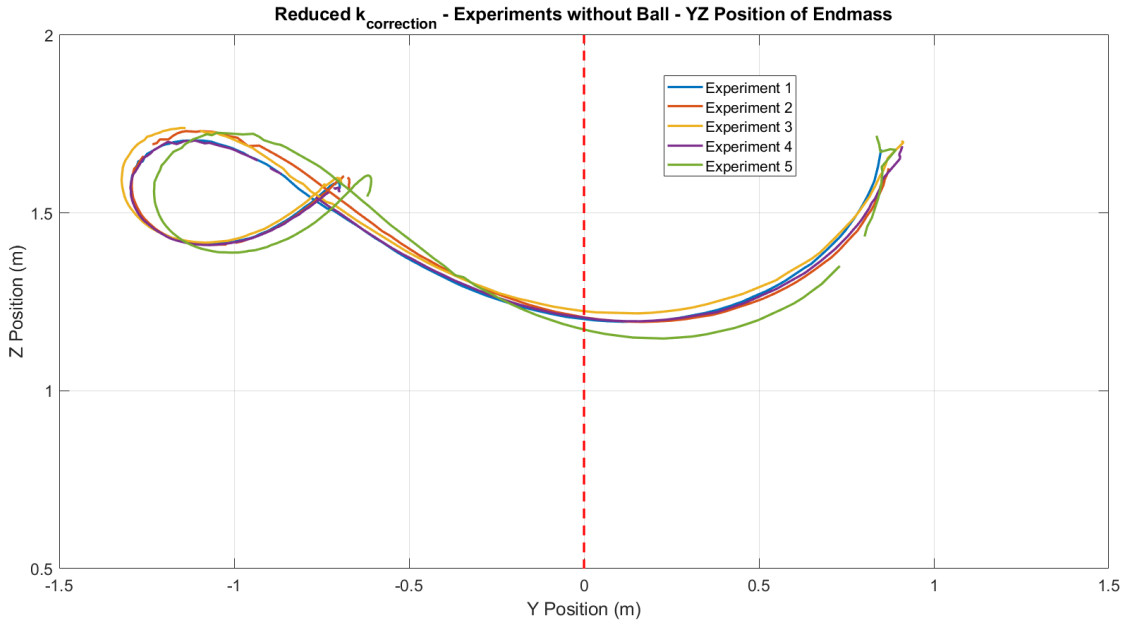


Figure 4.37: Reduced Correction Gain of the Oscillation Increase algorithm - Experiments without Ball

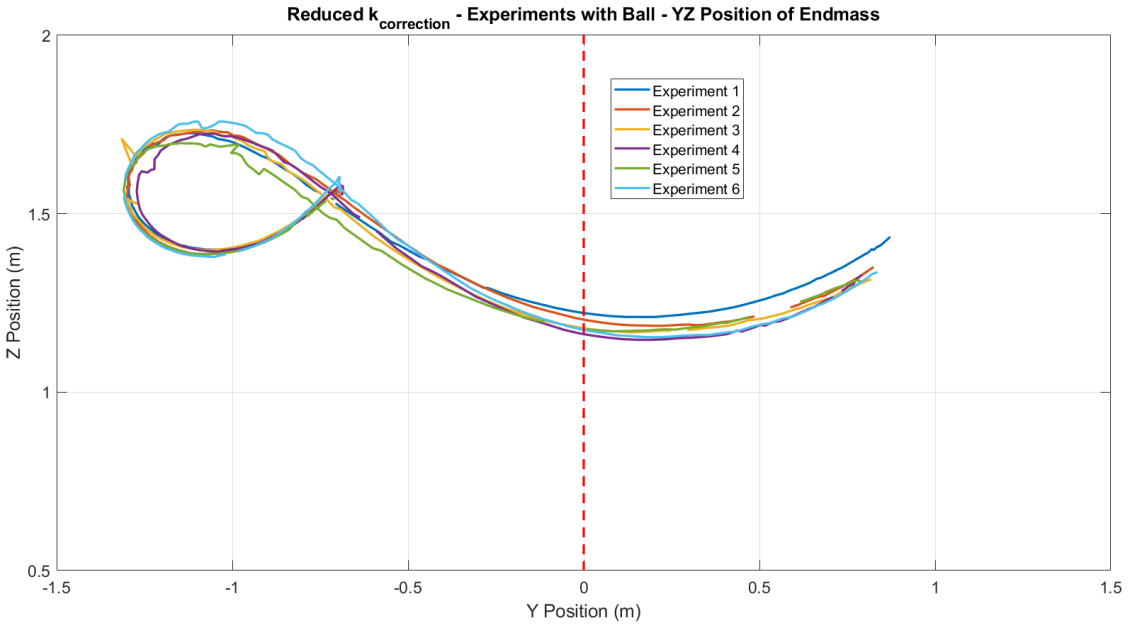


Figure 4.38: Reduced Correction Gain of the Oscillation Increase algorithm - Experiments with Ball

tion also became very stable and the repeatability in this experiment achieved was 4 out 6 experiments. This was clearly an improvement from the previous results.

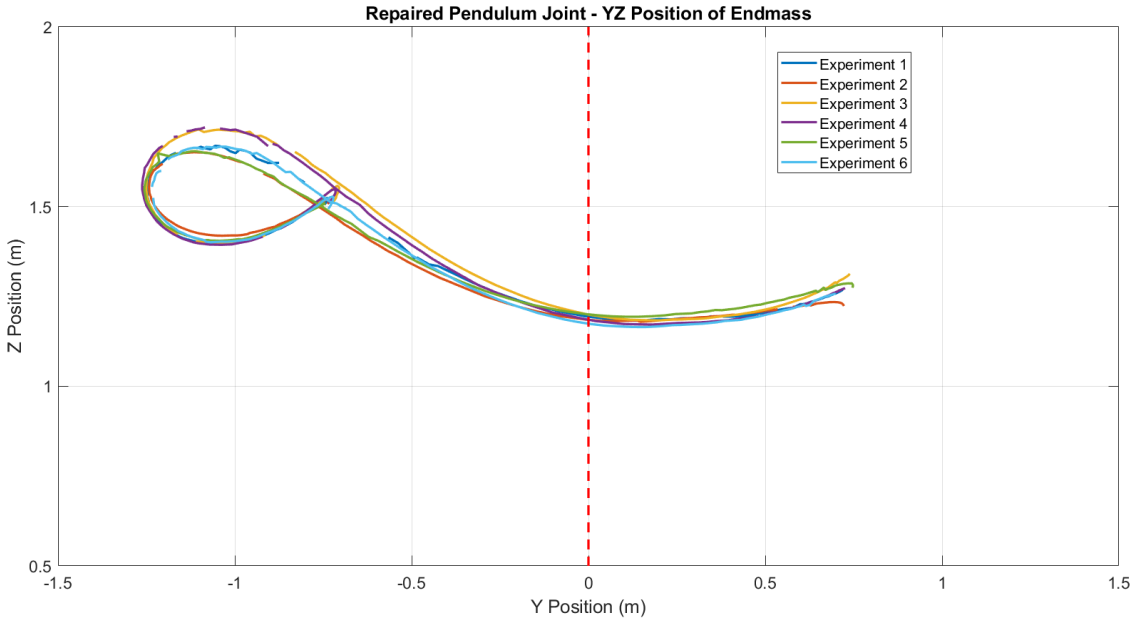


Figure 4.39: Repaird Joint of the Pendulum

Analysis of the Ball Motion

After getting the sufficient repeatability of the ball passing through a hoop, next step was to derive the relationship between impact conditions and the ball trajectory. This will help us to determine the required impact condition to hit the ball in the not pre-defined conditions. For this purpose, the Vicon markers needed to be installed on the ball so that we can track the ball using Vicon cameras. After doing some research, it was found that any retro-reflective material can be used to be detected in Vicon cameras. So the retro-reflective cloth cut in the shape of circles was attached on the ball. However, the Vicon was not recognizing them as the valid markers and it needs proper spherical markers only. So the default markers were used with ball also (Fig. 4.40). However, this will affect the motion of the ball as these markers have some mass and they will affect the aerodynamics also. However, using them was the best option to track the ball.

Fig. 4.41 shows the first experiment done with the ball tracking. We can see that the trajectory of the ball is varying too much in each experiment. With the visual observations, it was found that the trajectory of the UAV gets affected too much if the pendulum is swinging at a very high angle. So it was decided to try the experiment with a reduced swing-up threshold angle of the pendulum at which the hitting trajectory starts. The experiments were done with 35 degrees threshold (Fig. 4.42) and with 30 degrees of threshold (Fig. 4.43). We can clearly observe in these trajectories that the best performance is achieved with the 30 degrees of threshold and the ball follows almost the same path in each run.

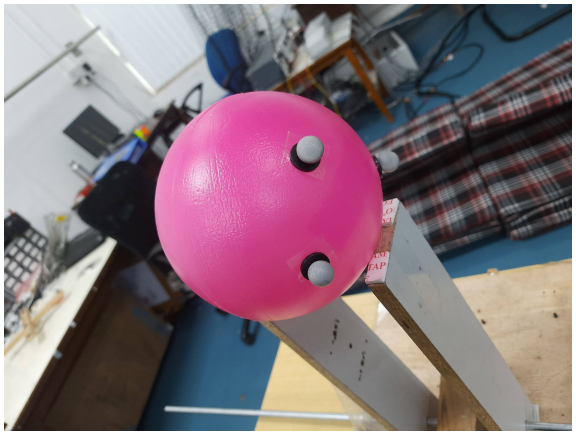


Figure 4.40: Ball with Vicon Markers

So it was concluded that with the current controller capabilities, the reduced threshold angle has the best repeatability. However, in the future with a better controller of the UAV, the repeatability can be ensured for high threshold angles also.

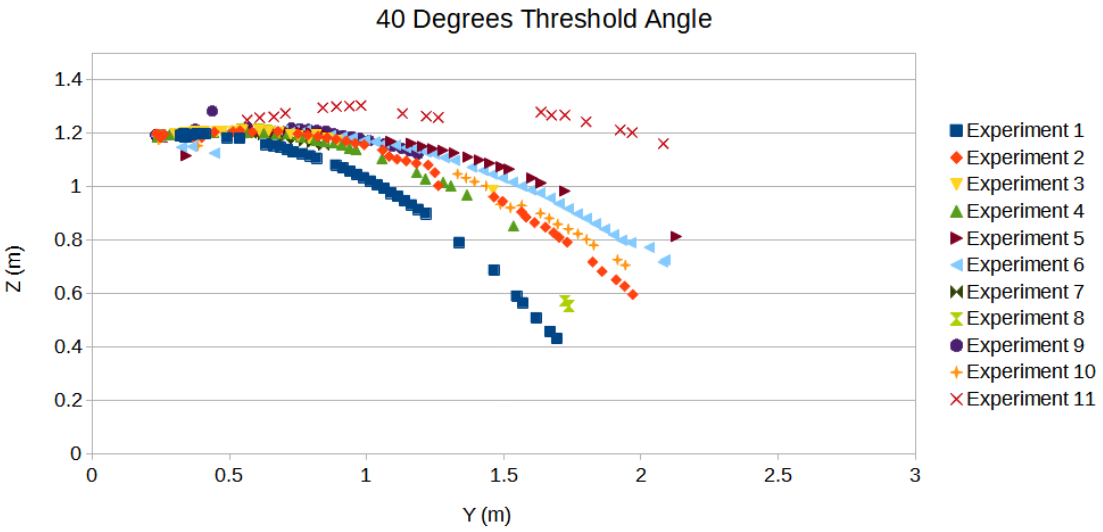


Figure 4.41: Trajectory of the Ball for 40 Degrees Threshold of Pendulum Angle

4.5 Conclusion & Future Work

During the first phase of the project, various characterization experiments were conducted: 1) Thorough analysis of the wireless communication hardware 2) Design and Manufacturing of the PWM to Thrust relationship setup 3) Design and Manufacturing of the 3DOF Attitude Testing Platform. With the Pixhawk-based quadrotor, various experiments with increasing autonomy were performed. MATLAB simulation for the 3D quadrotor was also done in order to understand the controller design.

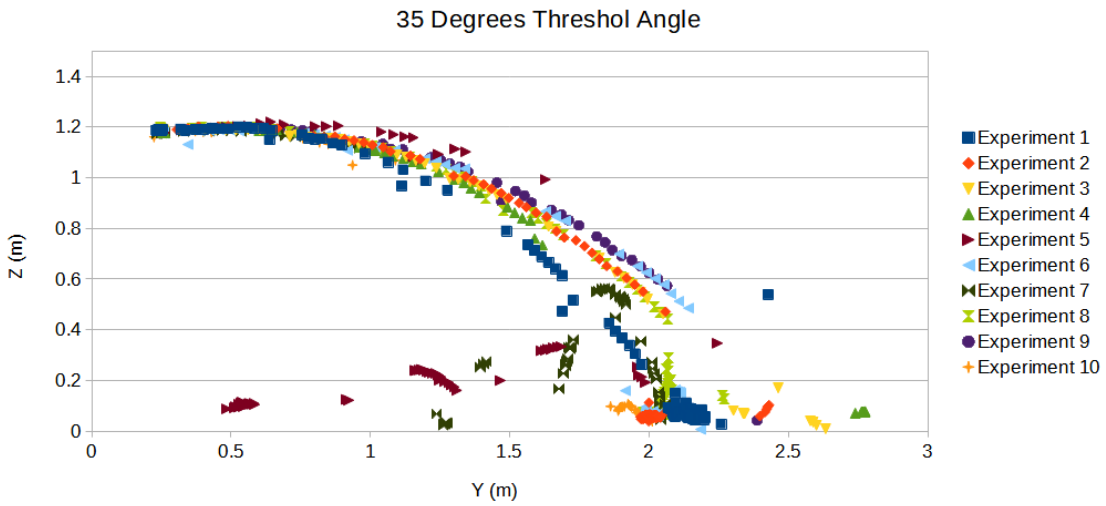


Figure 4.42: Trajectory of the Ball for 35 Degrees Threshold of Pendulum Angle

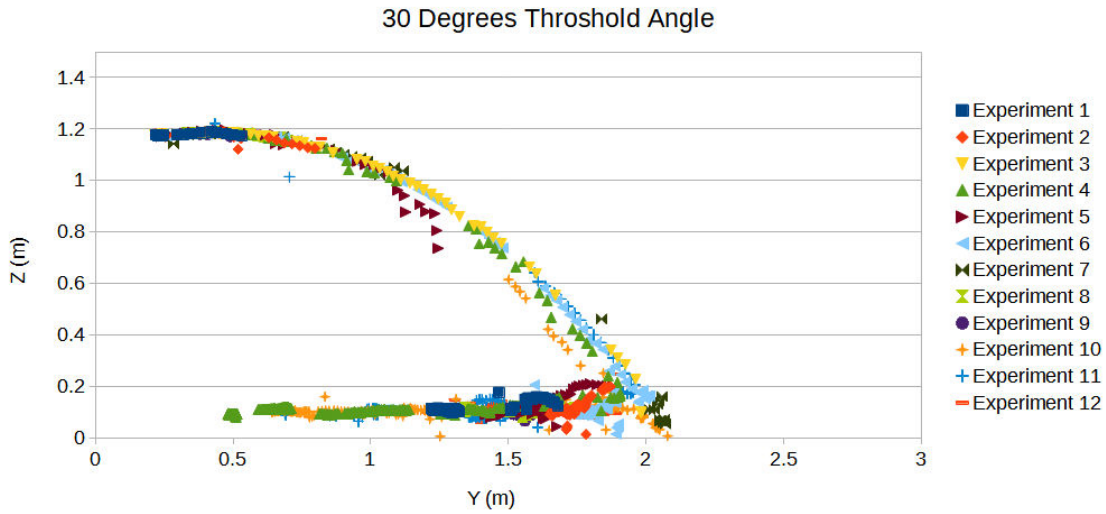


Figure 4.43: Trajectory of the Ball for 30 Degrees Threshold of Pendulum Angle

In the second phase of the project, the focus was on using the offboard mode of the Pixhawk. The offboard outerloop controller was designed using ROS. In the preparation process, the characterization was done regarding thrust to input signal relationship for various combinations of hardware. The main task was to tune the PID controllers of innerloop and outerloop controllers, which took a significant amount of time of the project. While flying the UAV for hover control and circular trajectory control, various improvements were made such as the implementation of the low pass filter, improvements in the hardware, testing the codes in Gazebo simulation, introducing integral error term etc. Gradually, the accuracy of the hover and circular trajectory increased. The best achieved

hover accuracy was 70% of the time UAV being inside 13 mm distance from the origin. The best circular trajectory had an RMS error of 14.73 mm.

After achieving the best possible performance in hover and circular trajectory experiments, the main task of doing dynamic experiments was initiated. The first experiment was done for the 1DOF Pendulum. The experiments were conducted to hit the ball using a pendulum and the goal was to pass the ball through a hoop with a fixed trajectory. Initially, the repeatability was very low, but after the improvements, the ball was passing through a hoop 4 out of 6 times.

After improving the repeatability, the experiments were conducted to determine the trajectory of the ball after being hit. Experiments with 3 different thresholds for the pendulum angles were done (30°,35°, and 40°). It was found that the ball could make almost the same trajectory for the 30°threshold angle. Hence it was concluded that with the current controller, the best perform ace will be achieved at a lower threshold angle of the pendulum.

In the future work of this project, we need to establish a mathematical model of the pendulum-based UAV system such that the UAV can determine its own trajectory given the ball and the hoop location. Also, the current PID controller needs to be improved or replaced with a better controller such that the motion of the pendulum does not affect the UAV trajectory much. After that, other dynamic experiments can be performed such as pendulum attached via string instead of a rigid link, mounting an arm on the UAV, and applying the force via an arm to the surroundings.

References

[1] D. Staff, "Configure openpilot cc3d evo." http://www.thedronesmag.com/configure-openpilot-cc3d-evo/. [Accessed May 25, 2021].

[2] N. Zimmerman, "Flight control and hardware design of multi-rotor systems," *Theory of Computing Systems Mathematical Systems Theory*, p. 17, 2016.

[3] Y. Mulgaonkar, M. Whitzer, B. Morgan, C. M. Kroninger, A. M. Harrington, and V. Kumar, "Power and weight considerations in small, agile quadrotors," in *Micro- and Nanotechnology Sensors, Systems, and Applications VI* (T. George, M. S. Islam, and A. K. Dutta, eds.), vol. 9083, pp. 376 – 391, International Society for Optics and Photonics, SPIE, 2014.

[4] PJRC, "Teensy® 4.0 development board." https://www.pjrc.com/store/teensy40.html. [Accessed Oct 11, 2021].

[5] Robu, "Pixhawk 2.4.8 drone flight controller px4 32 bit autopilot – good quality." https://robu.in/product/pixhawk-px4-autopilot-pix-2-4-8-32-bit-flight-controller/?gclid=CjwKCAjwn8SLBhAyEiwAHNTJbUU-knOlauSLr5cVYpIZcQ369YXPLtP12LLuXVkKXPfi
BwE. [Accessed Oct 22, 2021].

[6] ELECTRONICS_COMP, "Anti-vibration shock absorber for apm/kk/mwc/pixhawk." https://www.electronicscomp.com/anti-vibration-shock-absorber-for-apm-kk-mwc-pixhawk?gclid=CjwKCAjwtIaVBhBkEiwAsr7-c58fPQz3qDjTlSHKZ-bc2llDHwBnhxUn0p-vQrGYhIVYJzyNPyu
BwE. [Accessed June 9 2022].

[7] H. M. PX4_Team, "Px4 quadrotor flight." https://review.px4.io/plot_app?log=be020f25-e815-48cf-912f-4890333083c6. [Accessed June 9 2022].

[8] P. Jain and V. Sangwan, "Generation and control of impulsive forces by a planar bi-rotor aerial vehicle through a cable suspended mass," in *2020 IEEE/ASME In-*

ternational Conference on Advanced Intelligent Mechatronics (AIM), p. 868–873, IEEE Press, 2020.

[9] S. El Ferik, G. Ahmed, and H. M. Omar, "Load swing control for an unmanned aerial vehicle with a slung load," in *2014 IEEE 11th International Multi-Conference on Systems, Signals Devices (SSD14)*, pp. 1–9, 2014.

[10] K. J. Åström and K. Furuta, "Swinging up a pendulum by energy control," *IFAC proceedings volumes*, vol. 29, no. 1, pp. 1919–1924, 1996.

[11] X. Meng, Y. He, and J. Han, "Survey on aerial manipulator: System, modeling, and control," *Robotica*, vol. 38, no. 7, p. 1288–1317, 2020.

[12] Harsh Modi, "Seminar report : Design and fabrication of multirotor aerial robotic platform." https://drive.google.com/file/d/1MsOpcjutY2ajHPld5cu2TLH38mc-2HSk/view?usp=sharing, Unpublished Report.

[13] N. Michael, D. Mellinger, Q. Lindsey, and V. R. Kumar, "Experimental evaluation of multirobot aerial control algorithms," 2010.

[14] I. S. ASSOCIATION, "Ieee 802.15.4-2020 - ieee standard for low-rate wireless networks." https://standards.ieee.org/standard/802_15_4-2020.html. [Accessed Oct 11, 2021].

[15] H. Zhang, Z. Zhao, Z. Meng, and Z. Lin, "Experimental verification of a multi-robot distributed control algorithm with containment and group dispersion behaviors: The case of dynamic leaders," *IEEE/CAA Journal of Automatica Sinica*, vol. 1, no. 1, pp. 54–60, 2014.

[16] E. Ebeid, M. Skriver, K. H. Terkildsen, K. Jensen, and U. P. Schultz, "A survey of open-source uav flight controllers and flight simulators," *Microprocessors and Microsystems*, vol. 61, pp. 11–20, 2018.

[17] PX4_User_Guide, "Manual/stabilized mode (multicopter)." https://docs.px4.io/master/en/flight_modes/manual_stabilized_mc.html. [Accessed Oct 12, 2021].

[18] PX4_User_Guide, "Altitude mode (multicopter)." https://docs.px4.io/master/en/flight_modes/altitude_mc.html. [Accessed Oct 12, 2021].

[19] B. B. C. Ltd., "Tfmini infrared module specification." https://robu.in/wp-content/uploads/2017/09/benewake-tfmini-micro-lidar-module-ip65-12-m-datasheet.pdf. [Accessed Oct 12, 2021].

[20] PX4_User_Guide, "Position mode (multicopter)." https://docs.px4.io/master/en/flight_modes/position_mc.html. [Accessed Oct 12, 2021].

[21] PX4_User_Guide, "Offboard mode (multicopter)." https://docs.px4.io/master/en/flight_modes/offboard.html. [Accessed Oct 22, 2021].

[22] P. U. Guide, "Parameter reference." https://docs.px4.io/v1.12/en/advanced_config/parameter_reference.html. [Accessed May 12, 2022].

[23] Robu, "What is oneshot and multishot in esc? / oneshot vs multishot esc / esc calibration protocol." https://robu.in/what-is-oneshot-and-multishot-in-esc-difference-between-oneshot-and-multish [Accessed May 12, 2022].

[24] PX4_User_Guide, "Multicopter pid tuning guide." https://docs.px4.io/v1.12/en/config_mc/pid_tuning_guide_multicopter_basic.html. [Accessed June 9, 2022].

[25] PX4_Team, "Px4 flight review." https://review.px4.io/. [Accessed June 9 2022].

[26] PX4_User_Guide, "Mc filter tuning control latency." ttps://docs.px4.io/v1.12/en/config_mc/filter_tuning.html. [Accessed June 9 2022].

[27] C. Res, "How to design and implement a digital low-pass filter on an arduino." https://www.youtube.com/watch?v=HJ-C4Incgpw. [Accessed June 10 2022].

[28] PX4_User_Guide, "Ubuntu development environment." https://docs.px4.io/v1.12/en/dev_setup/dev_env_linux_ubuntu.html. [Accessed June 10 2022].

[29] PX4_User_Guide, "Ros with gazebo simulation." https://docs.px4.io/master/en/simulation/ros_interface.html. [Accessed June 10 2022].

[30] H. Modi, "Offboard outerloop codes." https://github.com/harshjmodi1996/Aerial_robotic_platform/tree/main/ol. [Accessed June 10 2022].

[31] H. Modi, "Initial successful offboard outerloop hover experiment." https://youtu.be/GQP5-q2I5nM. [Accessed June 13 2022].

[32] H. Modi, "Hover experiment with emax motors." https://youtu.be/m5R-ZGSf-lo. [Accessed June 13 2022].

[33] H. Modi, "Hover flight with cutoff frequencies tuned." https://youtu.be/vz1IO9XRzbM. [Accessed June 11 2022].

[34] H. Modi, "First successful circular trajectory with changing yaw." https://youtu.be/YKQAA7BvES8. [Accessed June 13 2022].

[35] H. Modi, "Circular trajectory after cutoff frequency tuning and notch filter applied." https://youtu.be/cctvNcMJjUc. [Accessed June 13 2022].

[36] H. Modi, "Xy plot with respect to time - best circular trajectory." https://youtu.be/mQ8Wt9zGOXg. [Accessed June 15 2022].

[37] H. Modi, "Xy plot of multiple circles with respect to time." https://youtu.be/xvpr9Cn_uiI. [Accessed June 15 2022].

[38] H. Modi, "Helix trajectory." https://youtu.be/dcXx76VcFvY. [Accessed June 13 2022].

[39] H. Modi, "Pendulum swing control." https://youtu.be/MWS9TsAT018. [Accessed June 13 2022].

[40] H. Modi, "Oscillation increase." https://youtube.com/shorts/936ngGOFpuo?feature=share. [Accessed June 13 2022].

[41] H. Modi, "Virtual impact with pendulum." https://youtu.be/m_K4SmcyXGA. [Accessed June 13 2022].

[42] H. Modi, "Hitting the box with pendulum." https://youtu.be/4N6sB0APQ_k. [Accessed June 13 2022].

[43] H. Modi, "Ball passing through hoop." https://youtu.be/RzogVVhaI8s. [Accessed June 13 2022].

[44] jarvisschultz, "how can i install vicon_bridge?." https://answers.ros.org/question/233177/how-can-i-install-vicon_bridge/. [Accessed June 13 2022].

[45] H. Modi, "Getting vicon position data on pixhawk using ros." https://drive.google.com/file/d/19D-9qES_PEIUNZdC44wlpaxAMLOMD9yw/view?usp=sharing. [Accessed June 13 2022].

Acknowledgements

I would like to thank Dr. Vivek Sangwan for guiding continuously throughout the project.

I would also like to thank members of INDUS Lab Prakash Suthar, Adarsh Mathai, Shubham Kumbhar, Akshay Kumar, Purushottam Mani, Shri Ishwarya S V for helping me at various stages of the project. I would like to thank Dean Student Affairs and Student Activity Center of IIT Bombay for providing a permission for the flying experiments inside the student gymkhana.

Appendices

Appendix A

SDF Code to Add Extra Link in Gazebo Simulation

```
<joint name='pendulumn_pivot' type='revolute'>
    <child>pendulumn_link</child>
    <parent>base_link</parent>
    <axis>
        <xyz>0 1 0</xyz>
        <limit>
            <lower>-1e+16</lower>
            <upper>1e+16</upper>
        </limit>
        <dynamics>
            <spring_reference>0</spring_reference>
            <spring_stiffness>0</spring_stiffness>
        </dynamics>
        <use_parent_model_frame>1</use_parent_model_frame>
    </axis>
</joint>

<link name="pendulumn_link">
        <pose>0 0 -0.05 0 0 0</pose>
        <inertial>
            <pose>0 0 -0.25 0 0</pose>
            <inertia>
            <ixx>0.02712343</ixx>
```

```
            <ixy>0</ixy>
            <ixz>0</ixz>
            <iyy>0.02713278</iyy>
            <iyz>0</iyz>
            <izz>0.00003342</izz>
        </inertia>
        <mass>0.125</mass>
    <visual name="pendulumn_link_visual">
        <pose>0 0 -0.25 0 0 0</pose>
        <geometry>
            <box>
                <size>0.01 0.01 0.5</size>
            </box>
        </geometry>
        <material>
            <script>
                <name>Gazebo/Red</name>
                <uri>file://media/materials/scripts/gazebo.material</uri>
            </script>
        </material>
    </visual>
    <visual name="pendulumn_link_endmass">
        <pose>0 0 -0.5 1.57 0 0</pose>
        <geometry>
            <cylinder>
                <radius>0.025</radius>
                <length>0.05</length>
            </cylinder>
        </geometry>
        <material>
            <script>
                <name>Gazebo/DarkGrey</name>
                <uri>file://media/materials/scripts/gazebo.material</uri>
            </script>
        </material>
    </visual>
```

```
    <velocity_decay/>
    <gravity>1</gravity>
</link>
```

Appendix B

Guidance to Setup the Experiments

As described in subsection 3.6, you would have installed ROS, MAVROS, MAVLINK, PX4-Autopilot (Only used for simulation) already. Now in order to do the hardware experiments, the following steps need to be followed:

1. Go to catkin_ws/src/mavros/mavros/launch directory and edit px4.launch file. Edit the argument "fcu_url" to the value "/dev/ttyUSB0:230400". This will ensure the mavliink communication via the XBee connected through USB.

2. In the same file, edit the value of "gcs_url" to "udp://:14560@127.0.0.1:14550". This will establish the communication of the MAVROS with the QGroundControl.

3. Inside the QGroundControl software, go to Application Settings (by clicking on the top-left Q) and untick the checkbox against "SiK Radio" under the autoconnect devices list. This will ensure that the QGroundControl does not connect to the XBee directly and hence XBee remains available for MAVROS.

4. Now, after connecting the XBee hardware, you can run the px4 node of the MAVROS using "roslaunch mavros px4.launch" command. The Pixhawk also should be powered before doing this. The data of the UAV will also become available on QGroundControl.

5. In order to receive the data from Vicon on the Offboard System, we need to install vicon_bridge package. For installing the vicon_bridge, follow the steps from (44).

6. In the lan settings of the offboard PC, change the ipv4 settings to IP Address "192.168.2.2", netmask "255.255.255.0", default gateway "192.168.2.1". Ensure that the settings on the Vicon PC are IP Address "192.168.2.1", netmask "255.255.255.0", default gateway "192.168.2.2". Using this settings, the Vicon PC and offoard PC can have local connection via Ethernet cable.

7. Inside the catkin_ws/src/vicon_bridge/launch directory, edit the file vicon.launch. In the argument datastream_hostport, put the value of "192.168.2.1:801". This will ensure that the vicon_bridge gets the data from server PC's localhost.

8. We will now create the package "ol" (shortform of outerloop) inside the catkin_ws to put our custom offboard codes.

9. Inside the src of catkin_ws, run the code "catkin_create_pkg ol rospy". This will create the necessary files for creating the "ol" package.

10. Navigate to catkin_ws directory and run "catkin build" in order to build the "ol" package.

11. Download the codes from (30) to the catkin_ws/src/ol directory. Also make "log_files" directory inside ol and create necessary logging directories inside "log_files"

12. Now, you are ready to run the experiment. First run the vicon_bridge, then px4.launch, then vision_pose_publisher, then run the necessary offboard code using rosrun.

There are also a few one time setups need to be done, which are already done on the existing setup. However, if in the future, one needs to check for them, the following steps need to be followed:

1. Calibrate the ESCs using QGroundControl. Visit "Vehicle Setup" section of the QGroundControl after connecting it via a USB cable. Then go to "Power" tab and click on calibrate ESC. It will show the necessary further steps.

2. Calibrate the Pixhawk sensors using "Sensors" tab of the "Vehicle Setup". The QGroundControl shows the necessary steps to be followed.

3. If for some reason, you need to connect the Pixhawk wirelessly directly with the QGroundControl via XBee, you will need to create communication link inside the QGroundControl. Visit "Application Settings" of the QGroundControl, go to "Comm Links" tab and click on Add. Give it a name, and change the Baudrate to 230400.

4. In order to use the PX4's own outerloop for position control, we need to send the Vicon data to Pixhawk. For this purpose, follow the guidance from (45).

Modi Harsh Jashvantbhai
IIT Bombay
17 June, 2022